webMethods EntireX

EntireX XML/SOAP Wrapper

Version 9.5 SP1

November 2013

Document ID: EXX-EEXXXMLWRAPPER-95SP1-20140628

Table of Contents

1 Introduction to the XML/SOAP Wrapper

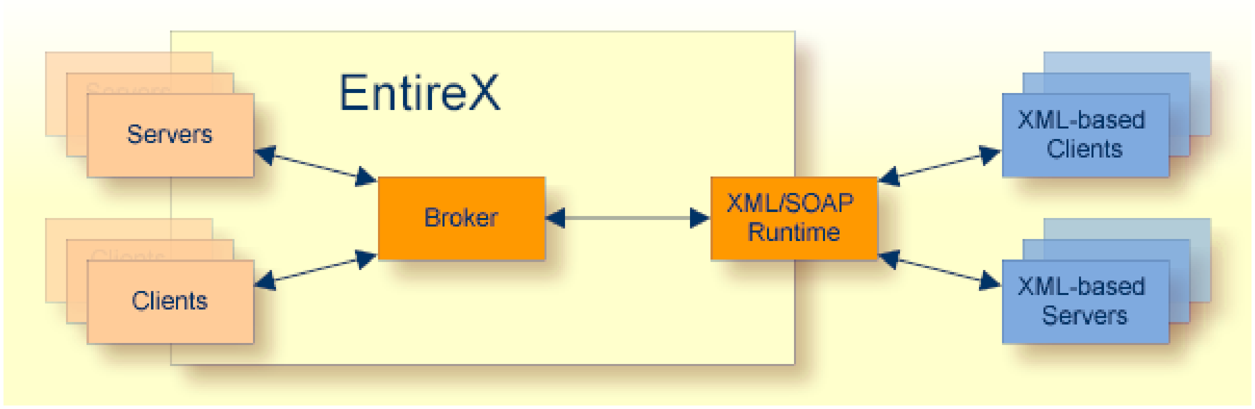
The EntireX XML/SOAP Wrapper enables XML-based communication to EntireX/Natural RPC servers and communication from EntireX/Natural RPC clients to XML-based servers.

XML/SOAP Wrapper Concepts

The EntireX XML/SOAP Wrapper and the XML/SOAP RPC Server enable XML-based communication via XML/SOAP Wrapper. An EntireX RPC server or a Natural RPC Server can communicate with XML-based clients. Similarly, an EntireX RPC client or Natural RPC client can communicate with an XML-based server via XML/SOAP RPC Server. EntireX participants (client or server) with an XML-based application (server or client) can be easily integrated with the *EntireX Workbench* and the EntireX XML/SOAP Wrapper/ XML/SOAP RPC Server.

The XML Mapping File is an XML file (file extension .xmm) generated by the EntireX XML Mapping Editor and used by the EntireX XML/SOAP Runtime, to map between XML documents and EntireX/Natural RPC parameters.

Communication is synchronous.



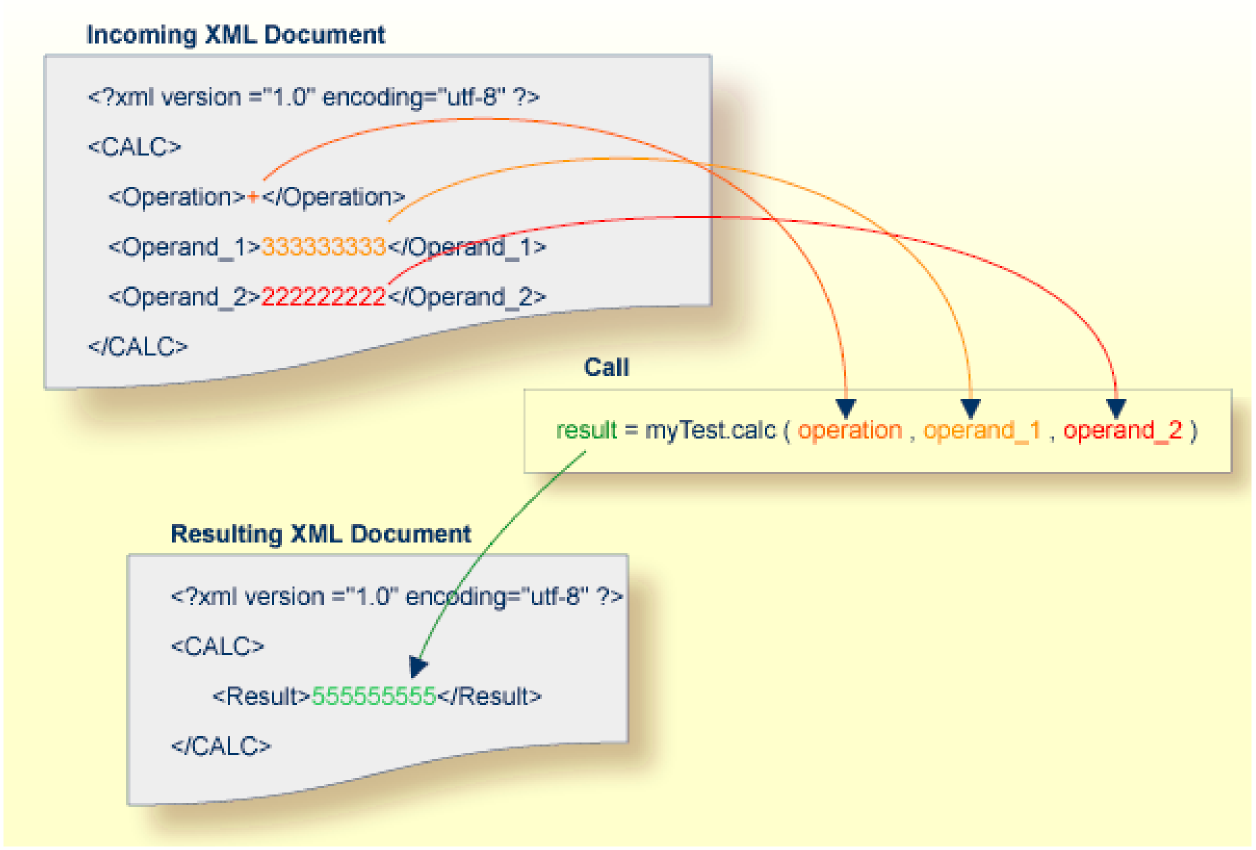
XML-based Clients Calling EntireX/Natural RPC Server

The XML-based client sends an XML document to EntireX XML/SOAP Runtime.

From an incoming XML document, the EntireX XML/SOAP Runtime extracts (using the mapping file) the information necessary to call an EntireX/Natural RPC server. It executes the RPC. The result returned by the EntireX/Natural RPC server is used to create an outgoing XML document.

The XML-based client gets an XML document containing the response.

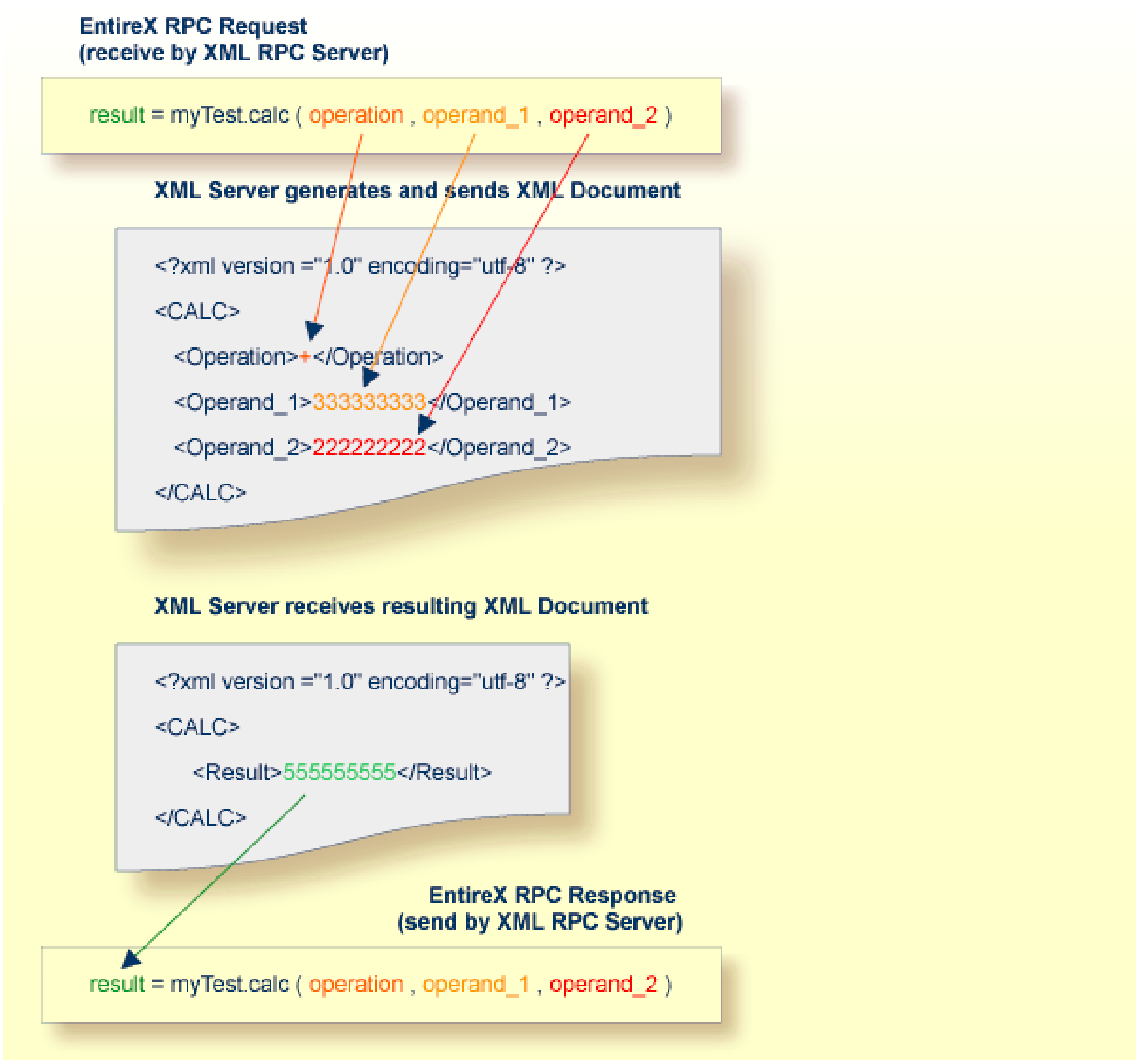
Mapping example: *calc*


Incoming XML Document
<?xml version ="1.0" encoding="utf-8" ?>
<CALC>
<Operation>+</Operation>
<Operand_1>333333333</Operand_1>
<Operand_2>222222222</Operand_2>
</CALC>
Call
result = myTest.calc (operation , operand_1 , operand_2)
Resulting XML Document
<?xml version ="1.0" encoding="utf-8" ?>
<CALC>
<Result>555555555</Result>
</CALC>

A mapping editor is provided to map the XML structures to those of the server application.

EntireX/Natural RPC Client Calling XML-based Server

The RPC client sends a request to the XML/SOAP RPC Server. The XML/SOAP RPC Server translates (using the mapping file) the information of an EntireX client call to XML (so it can be understood by an application with an XML interface). The XML/SOAP RPC Server sends the XML document to the XML-based server via HTTP(s) and receives an XML document from the XML-based server. It translates the received XML document and sends the response to the RPC client.


EntireX RPC Request
(receive by XML RPC Server)
result = myTest.calc (operation , operand_1 , operand_2)
XML Server generates and sends XML Document
XML Server receives resulting XML Document
EntireX RPC Response
(send by XML RPC Server)
result = myTest.calc (operation , operand_1 , operand_2)

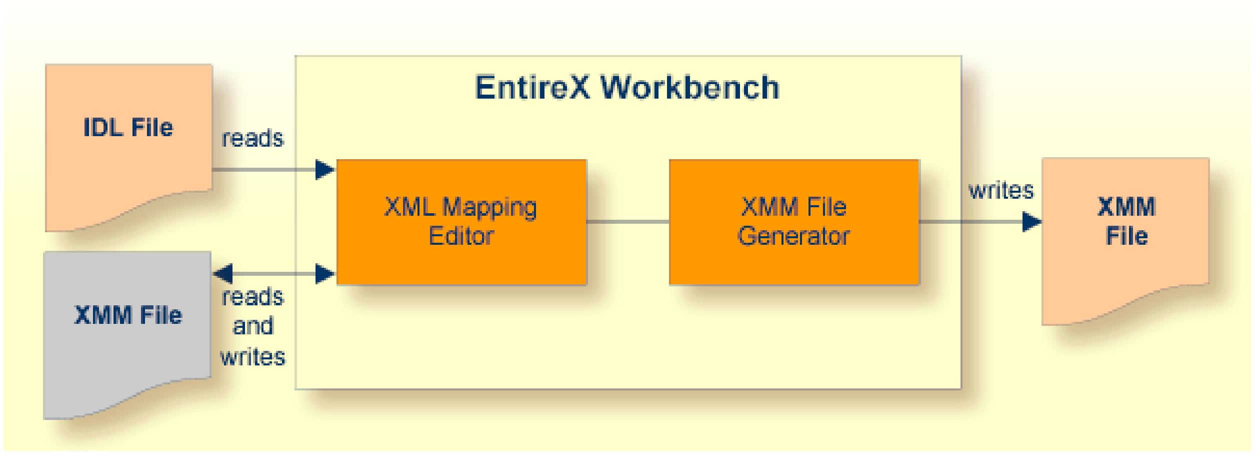
```
<?xml version ="1.0" encoding="utf-8" ?>
<CALC>
  <Operation>+</Operation>
  <Operand_1>333333333</Operand_1>
  <Operand_2>222222222</Operand_2>
</CALC>
```

```
<?xml version ="1.0" encoding="utf-8" ?>
<CALC>
    <Result>555555555</Result>
</CALC>
```

Development Environment

- XML Mapping Editor
- Generation Utilities

During the development process, users map XML document structures to RPC parameters and vice versa, and generate the XMM (XML Mapping) File.



XML Mapping Editor

The input for the XML Mapping Editor is an existing Software AG IDL file. See *XML Mapping Editor*.

The XML Mapping Editor allows users to map the structure of an incoming XML document to the incoming parameters (IN, INOUT) of an Natural RPC Server and the parameters returned by the EntireX/Natural RPC server (OUT, INOUT) to the structure of the outgoing XML document.

To ease the development process, standard mappings can be created automatically. These can be modified and adapted.

Supported standard mappings are:

- element-oriented mapping
- attribute-oriented mapping
- SOAP-conformant mapping

See ***Basic IDL-XML Mapping***.

Generation Utilities

After the mapping process, the following runtime components can be generated:

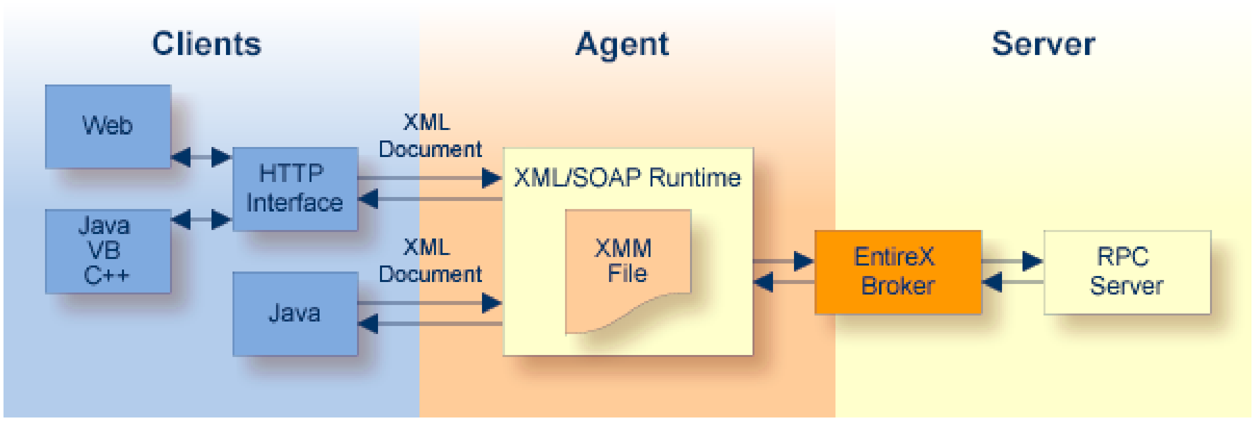
- **XML Mapping Files**
  are used during runtime to map between the XML documents and the EntireX/Natural RPC parameters.

XML/SOAP Runtime Environment

The EntireX XML/SOAP Runtime enables XML-based clients to communicate with an EntireX/Natural RPC Server; and it enables EntireX RPC clients to communicate with applications that have an XML or SOAP interface (via XML/SOAP Runtime).

XML Clients Communicating with EntireX/Natural RPC Servers

The incoming XML document is mapped by the generated XMM file to the EntireX/Natural RPC parameters, using the mapping files created during the development process. From the parameters returned by the EntireX/Natural RPC server, the generated XMM file creates the outgoing XML document, using the corresponding mapping files.



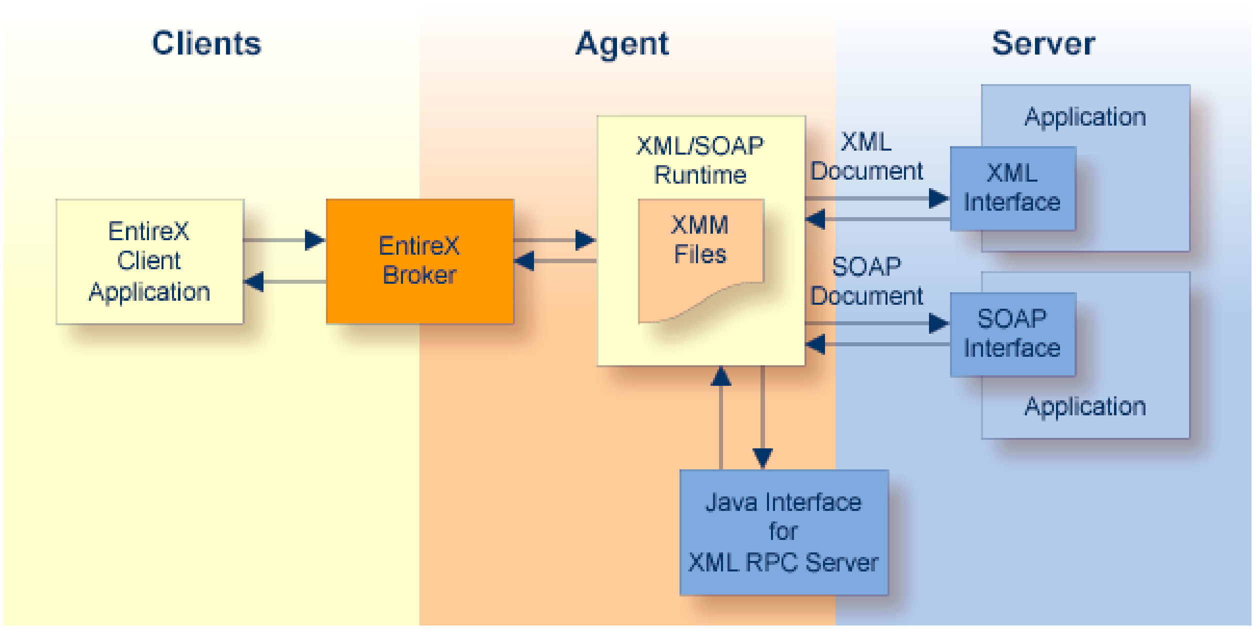
The XML/SOAP Runtime provides two client-side interfaces:

- **A Java interface**
  The Java interface can be used by Java clients. See ***Client Using the Java Interface***.
- **An HTTP interface**
  The HTTP interface supports HTTP Post Request and HTTP Get Request. See ***The HTTP Interface***.

To make it possible to call the XML/SOAP Runtime from within a Web server, see *Administering the EntireX XML/SOAP Listener* in the UNIX and Windows administration sections.

EntireX Clients Communicating with XML or SOAP Interfaces

The XML/SOAP Runtime can connect EntireX clients to applications that have an XML or SOAP interface, for example SAP or PeopleSoft. These applications are then the XML server. Example: An EntireX client requests information of an SAP application. The request is sent to the EntireX Broker and the EntireX XML/SOAP Runtime (translated to XML there) and then to the SAP XML interface, processed by the SAP application. The repsonse is returned the same way.



The XML/SOAP Runtime provides two server-side interfaces:

- **A Java interface**
  The Java interface can be used to modify the document and determine further processing.
- **An HTTP interface**
  The XML/SOAP document is sent to the specified target.

Glossary of Terms

XML-based Client

Client sends and receives data as an XML or SOAP document.

XML-based Server

Server receives and sends data as an XML or SOAP document (for example a Web service).

XML/SOAP RPC Server

With the XML/SOAP RPC Server you can process XML-based server calls from EntireX RPC clients/Natural RPC clients. The EntireX RPC client communicates with the XML-based server, using the XML/SOAP RPC Server.

XML/SOAP Runtime

The EntireX XML/SOAP Runtime enables XML-based clients to communicate with an EntireX/Natural RPC Server; and it enables EntireX RPC clients to communicate with applications that have an XML or SOAP interface (via XML/SOAP Runtime).

XML/SOAP Wrapper

The EntireX XML/SOAP Wrapper enables XML-based communication to EntireX/Natural RPC servers and communication from EntireX/Natural RPC clients to XML-based servers.

XML Mapping File

The XML Mapping File is an XML file (file extension .xmm) generated by the EntireX XML Mapping Editor and used by the EntireX XML/SOAP Runtime, to map between XML documents and EntireX/Natural RPC parameters.

2 Migration Considerations for XML/SOAP Components

Migrating XML/SOAP Components for EntireX Version 8 and above

This section covers the following topics:

Migrating XML/SOAP RPC Server

Update the start script to set the paths to current installation. If the properties file contains a path to the configuration file, you may need to adjust the path. The properties and configuration files can be used as before. See *Configuration File for the XML/SOAP RPC Server* in the UNIX and Windows administration documentation for more information.

Migrating XML/SOAP Wrapper

Update the start script to set the paths to current installation.

Migrating XML/SOAP Listener

Using the Software AG Web Server / Software AG Runtime from Software AG Installation

Save your Web services (AAR files) and modified configuration files (i.e. *axis2.xml*). You do not need to re-generate your Web services (AAR files). If the configuration was modified, perform the same modifications for the new installation and restart the Software AG Web Server. Redeploy the web services (AAR files).

Note: The new installation will use another port by default, which means that the Web clients need to be adjusted.

Using another Web Server

Save your Web services (AAR files) and modified configuration files (i.e. *axis2.xml*). You do not need to re-generate your Web services (AAR files). For deployment of file *wsstack.war* and copying file *entirex.jar* to Web Services Stack, see the separate Web Services Stack documentation, also available under webMethods Product Documentation on the **Software AG Documentation** website.

If configuration was modified, perform the same modifications for new installation and restart the Web server. Redeploy the Web services (AAR files).

Migrating XML/SOAP Components from EntireX Version 7.n

The architecture of the EntireX XML/SOAP Listener (formerly known as XML Servlet) and EntireX XML/SOAP RPC Server was changed with version 8.0. The previous version worked as a Web application running directly in the Web server; the current version works as part of Software AG Common Web Services Stack (WSS), which is a Web application running in a Web server. This change has impact on configuration, packaging and deployment of EntireX Web services.

Note: This section describes migration from a version earlier than 8.

Migrating XML/SOAP RPC Server from Version 7.n

The XML/SOAP RPC Server requires the directory of a Software AG Common Web Services Stack installation. The JAR file *wsstack-client.jar* located in *<WS_STACK_HOME>/lib* must be added to `classpath`. (Note: the JAR file *wsstack-client.jar* must remain in this *lib* directory). File *entirex.jar* from EntireX installation must also be present in `classpath`.

The XML parser required has changed. Therefore, the (optional) properties file (if used) must be modified by redefining the parameter for the XML stream parser in the following way:

```
entirex.sdk.xml.runtime.xmlparserfactory=com.ctc.wstx.stax.WstxInputFactory
```

The configuration file can be used as before. To make use of the latest features, some extensions in the configuration file are required. See *Configuration File for the XML/SOAP RPC Server* in the UNIX and Windows administration documentation for more information.

Migrating XML/SOAP Wrapper from Version 7.n

The used XML parser has changed. XML/SOAP Wrapper uses XMLStreamParser, so the classpath must contain the JAR files of XMLStreamParser. The EntireX installation comes with *stax-api.jar* and *wstx-asl.jar*. The two XMLStreamParser JAR files are located in *<EntireX Home>/classes*.

If you are compiling source using the class `XMLRPCService`, you require an additional JAR file *axiom-api-<version>.jar*. This JAR file is located in *<WebservicesStack>/lib*.

Migrating XML/SOAP Listener from Version 7.n - Introduction

The XML/SOAP Listener requires a Software AG Common Web Services Stack (WSS) installation. The WSS Web application is the container of XML/SOAP Listener. In previous EntireX versions (lower than version 8) the XML/SOAP Listener was referred to as “XML Servlet” and was a Web application itself. The file *entirex.jar* containing the XML/SOAP Runtime is placed in directory *lib* of the WSS Web application. An EntireX Web service is created using the XML Mapping Editor as in previous versions, and using a Packaging Wizard to build an AAR file. The AAR file contains all information for this Web service and must be deployed to the Web application.

The following new terms are applicable:

- **AAR Files**
  EntireX packs the XMM file(s) and some more files in a web service archive (.aar) that is deployed in the Web application WSStack. To put one or more XMM files in one web service archive, select the XMM files and from the context menu choose **Generate Web Service from EntireX Mapping...**. All the files you select must have the same mapping type (i.e. SOAP or non-SOAP mapping); do not mix mapping types.
- **File entirex-forward.war**

  The servlet `entirex-forward` only forwards the message to the new URL. This means you can continue using the old URL and migrate the service to a new infrastructure. The servlet does not process the payload, so this Web application does not contain an *entirex.jar* file and does not contain a mapping file (XMM).

The following scenarios are available for migrating a service offered by XML Servlet of earlier EntireX versions (lower than version 8):

- migrating to one or more services, using new URL(s); see *Migrating from Version 7.n to one or more Listener Services with new URL(s)*
- migrating to one service only, keeping the URL; see *Migrating from Version 7.n to one Listener Service, Keeping the URL used by Applications*

Migrating from Version 7.n to one or more Listener Services with new URL(s)

You can migrate a service offered by XML Servlet of earlier EntireX versions (lower than version 8) to one or more services in the context of XML/SOAP Listener with new URL(s). That means you can split the original service into different services, each with its own URL. However, this means you need to update the address in the web client.

To migrate a service with new URLs

1. Copy all the IDL-XML mapping files (XMMs) listed in file *xml-init.xml* to one Eclipse project.
2. Select multiple XMM files that are to be bundled in one service, and from the context menu choose **Generate Web Service from EntireX Mapping ...**.

3 Modify the service name if required.

4 Uncheck **Use of defaults**.

5 Check **General service parameters (xml-init.xml)** if you have any XMM-independent settings in file *xml-init.xml*.

6 Check **Set connection and security parameters in mapping file**.

7 Follow the wizard.

8 Modify the setting if required. The included XMM files can be selected with combo box at the top.

9 Deploy the service.

10 Change the target URL of calling instances (Web clients).

Migrating from Version 7.n to one Listener Service, Keeping the URL used by Applications

The first step is to generate one service archive with all XMM files listed in file *xml-init.xml*. The second step is the configuration and usage of *entirex-forward.war*. This section covers the following topics:

- Step 1: Generate Service Archive
- Step 2: Use File entirex-forward.war
- Example of Migrating with Servlet entirex-forward

Step 1: Generate Service Archive

To generate a service archive

1. Copy all IDL-XML mapping files (XMMs) listed in *xml-init.xml* to one Eclipse project.
2. Select all XMM files, and from the context menu choose **Generate Web Service from EntireX Mapping ...**.
3. Modify the service name if required. This service name is used for configuring of servlet `entirex-forward` later.
4. Uncheck **Use of defaults**.
5. Check **General service parameters (xml-init.xml)**' if you have any XMM-independent settings in file *xml-init.xml*.
6. Check **Set connection and security parameters in mapping file**.
7. Follow the wizard.
8. Modify the setting if required. The included XMM files can be selected with combo box at the top.
9. Deploy the service.

Step 2: Use File entirex-forward.war

This Web application delivered as *entirex-forward.war* allows you to forward a SOAP/XML request from an endpoint used in an older EntireX version (e.g. *http://hostname:port/entirex/xmlrt*) to an endpoint used in the current EntireX version (e.g *http://hostname:port/wsstack/services/example*).

The steps below explain how to forward, for example, from the endpoint *http://localhost:10010/entirex/xmlrt* to the endpoint *http://localhost:10010/wsstack/services/example*.

Important: The servlet `entirex-forward` forwards the calls to exactly one service and replaces the old EntireX Web application; it does not run in parallel (old EntireX XML Servlet and the new servlet `entirex-forward`).

Note: You *cannot* use the file *entireX-forward.war* to forward a call from one Web application server to another.

To forward a SOAP/XML request

1. Rename *entirex-forward.war* to *entirex.war*.
2. Deploy *entirex.war* into the *webapps* folder of your application server (e.g. Tomcat).
3. Make sure that the Web Services Stack Web application (*wsstack.war*) is still deployed in the same *webapps* folder.
4. Make sure that a service with the name "example" is deployed into the Web Services Stack Web application. See Step 4 under ***Step 1: Generate Service Archive***.
5. Make sure that your application server allows cross-context access. Set the `crossContext` attribute of all `Context` implementations to "true". In Apache Tomcat you can modify the file *$CATALINA_HOME/conf/context.xml*:

   Set

   ```
   <Context>
   ```

   to

   ```
   <Context crossContext="true">
   ```

6. Make sure that the file *web.xml* (located in *webapps/entirex/WEB-INF*) contains the following entries:

   ```
   <servlet>
    <description>
    This is the EntireX ForwardServlet</description>
    <display-name>
    forward</display-name>
    <servlet-name>forward</servlet-name>
    <servlet-class>
    com.softwareag.entirex.xml.forward.ForwardServlet</servlet-class>
    <init-param>
     <description>
     This is the targetURI</description>
     <param-name>targetURI</param-name>
     <!--replace the servicename with service name defined in aar -->
     <param-value>/services/servicename</param-value>
    </init-param>
    <init-param>
     <description>
     This is the target context i.e. /wsstack</description>
     <param-name>targetContext</param-name>
     <param-value>/wsstack</param-value>
   ```

```
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>forward</servlet-name>
  <url-pattern>/xmlrt</url-pattern>
</servlet-mapping>
```

The old endpoint *http://hostname:port/entirex/xmlrt* is built from the name of the Web application ("entirex.war") and the `url-pattern` of the servlet `entirex-forward` ("/xmlrt"). The new endpoint is built from the `targetContext(/wsstack)` and the `targetURI(/"services/servicename")`.

You can modifiy these parameters to adopt your Web service.

Example of Migrating with Servlet entirex-forward

This section describes how a migrated service interacts with servlet `entireX-forward`.

For this example, the Tomcat installation that comes with the EntireX installation (Version 8.2) is used. The assumed endpoint from the previous XML Servlet installation is *http://localhost:10010/entirex/xmlrt*. The endpoint of the migrated service will be *http://localhost:10010/wsstack/services/exampleXML*.

The start point is file *xml-init.xml* (below), with its mapping file *example.xmm*.

Follow the steps above for generating a Web service archive file (*aar) with service name "exampleXML" (see ***Step 1: Generate Service Archive***). The installation contains the generated *exampleXML.aar* in directory *<installation home>/EntireX/examples/RPC/basic/example/XMLClient*. File *exampleXML.aar* is also deployed during installation if the Software AG Web Server was selected.

The last step is to configure and deploy the servlet `entirex-forward`.

To configure and deploy the servlet `entirex-forward`

1. Rename *entirex-forward.war* to *entirex.war*.
2. Deploy *entirex.war* to the *webapps* folder of your application server (e.g. Tomcat).
3. Make sure that the Web Services Stack Web application (*wsstack.war*) is still deployed in the same *webapps* folder.
4. Make sure that a service with the name "exampleXML" is deployed into the Web Services Stack Web application. If the web server is running, start a browser session and enter URL *http://localhost:10010/wsstack/services*. The service "exampleXML" should be listed.
5. Make sure that your application server allows cross-context access. See the `crossContext` attribute of all `Context` implementations to "true". In Apache Tomcat you can modify file *$CATALINA_HOME/conf/context.xml*:

   Set

```
<Context>
```

to

```
<Context="true">
```

6 Make sure that file *web.xml* (located in *webapps/entirex/WEB-INF*) contains the following entries:

```
<servlet>
 <description>
 This is the EntireX ForwardServlet</description>
 <display-name>
 forward</display-name>
 <servlet-name>forward</servlet-name>
 <servlet-class>
 com.softwareag.entirex.xml.forward.ForwardServlet</servlet-class>
 <init-param>
  <description>
  This is the targetURI</description>
  <param-name>targetURI</param-name>
  <param-value>/services/exampleXML</param-value>
 </init-param>
 <init-param>
  <description>
  This is the target context i.e. /wsstack</description>
  <param-name>targetContext</param-name>
  <param-value>/wsstack</param-value>
 </init-param>
</servlet>
<servlet-mapping>
 <servlet-name>forward</servlet-name>
 <url-pattern>/xmlrt</url-pattern>
</servlet-mapping>
```

The old endpoint *http://hostname:port/entirex/xmlrt* is built from the name of the Web application ("entirex.war") and the URL pattern of the servlet `entirex-forward` ("/xmlrt"). The new endpoint is built from the `targetContext(/wsstack)` and the `targetURI(/"services/exampleXML").`

7 Restart Tomcat.

8 Open the Software AG Designer and change to the EntireX perspective.

9 Choose **Windows > Show View > XML Tester** to open the XML Tester.

10 Browse for *<installation home>/EntireX/examples/RPC/basic/example/XMLClient/example.xmm*.

11 Create a sample request document.

12 Start Java RPC server with script `jrpcserver.bat` or `jrpcserver.bsh` (depending on platform) in directory *bin* of the EntireX installation that is used as a back-end.

13 Select **HTTP Test (XML/SOAP listener)**.

14 Set **HTTP URI** to "http://localhost:10010/wsstack/services/exampleXML".

15 Send request to see the service works.

16 Set **HTTP URI** to "http://localhost:10010/entirex/xmlrt".

17 Send request again (the request is sent to "old" address and will be processed in the XML/SOAP Listener embedded in Web Services Stack).

3 Using the XML/SOAP Wrapper

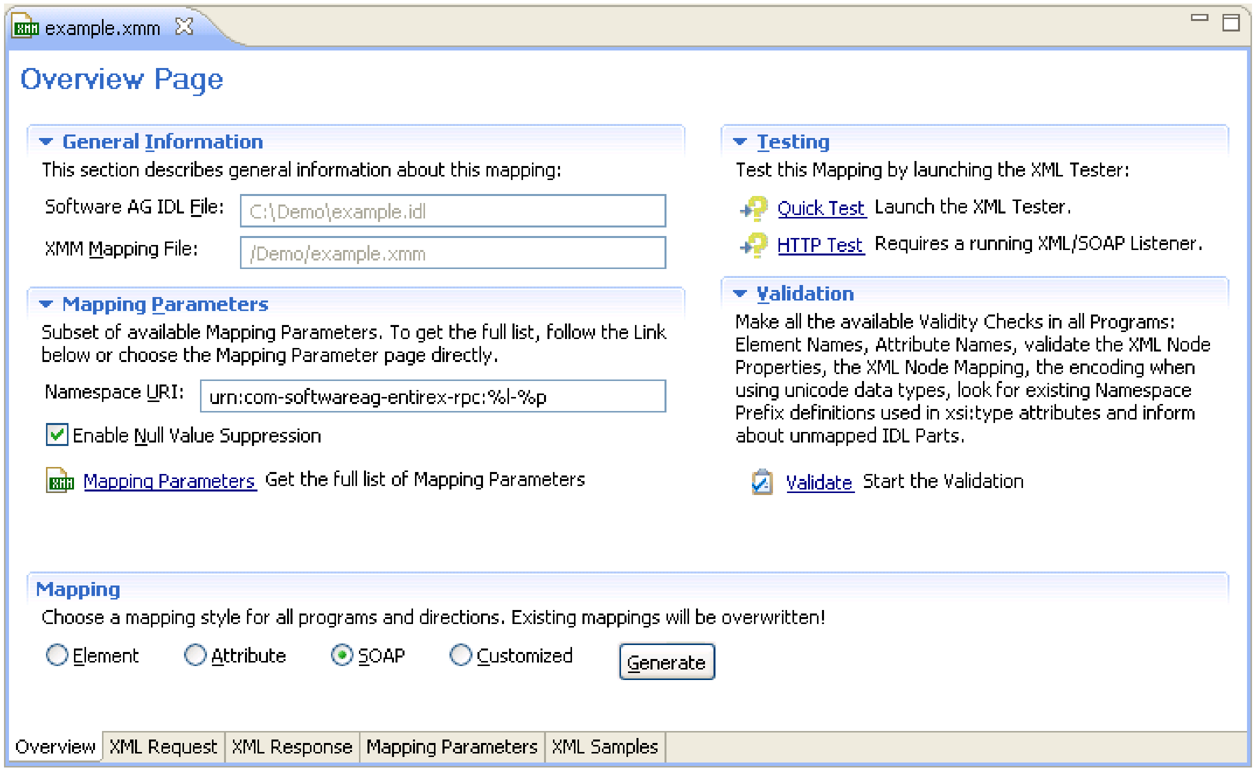
Mapping IDL Parameters to XML Structures

Use the *XML Mapping Editor* to map IDL parameters to XML structures.

The mapping editor offers several mapping types:

- element mapping: all IDL parameters map to elements in an XML document
- attribute mapping: all scalar/simple IDL parameters map to attributes; program name and complex types (groups and structures) map to elements
- SOAP mapping: a standard SOAP document is generated, IDL parameters map to elements in <Body>-part
- user-defined mapping: users create their own XML or SOAP documents and map the IDL parameters to elements or attributes of an XML document.

In addition, the mapping editor shows one of general information (**Overview** page), the request message (**XML Request**), response message within fault information (**XML Response** page), mapping parameters (**Mapping Parameters** page) or sample editor (**XML Samples** page) at one time.



This section describes general information about this mapping:

Software AG IDL File: C:\Demo\example.idl

XMM Mapping File: /Demo/example.xmm

Subset of available Mapping Parameters. To get the full list, follow the Link below or choose the Mapping Parameter page directly.

Namespace URI: urn:com-softwareag-entirex-rpc:%l-%p

[✓] Enable Null Value Suppression

Mapping Parameters Get the full list of Mapping Parameters

Test this Mapping by launching the XML Tester:

Quick Test Launch the XML Tester.

HTTP Test Requires a running XML/SOAP Listener.

Make all the available Validity Checks in all Programs: Element Names, Attribute Names, validate the XML Node Properties, the XML Node Mapping, the encoding when using unicode data types, look for existing Namespace Prefix definitions used in xsi:type attributes and inform about unmapped IDL Parts.

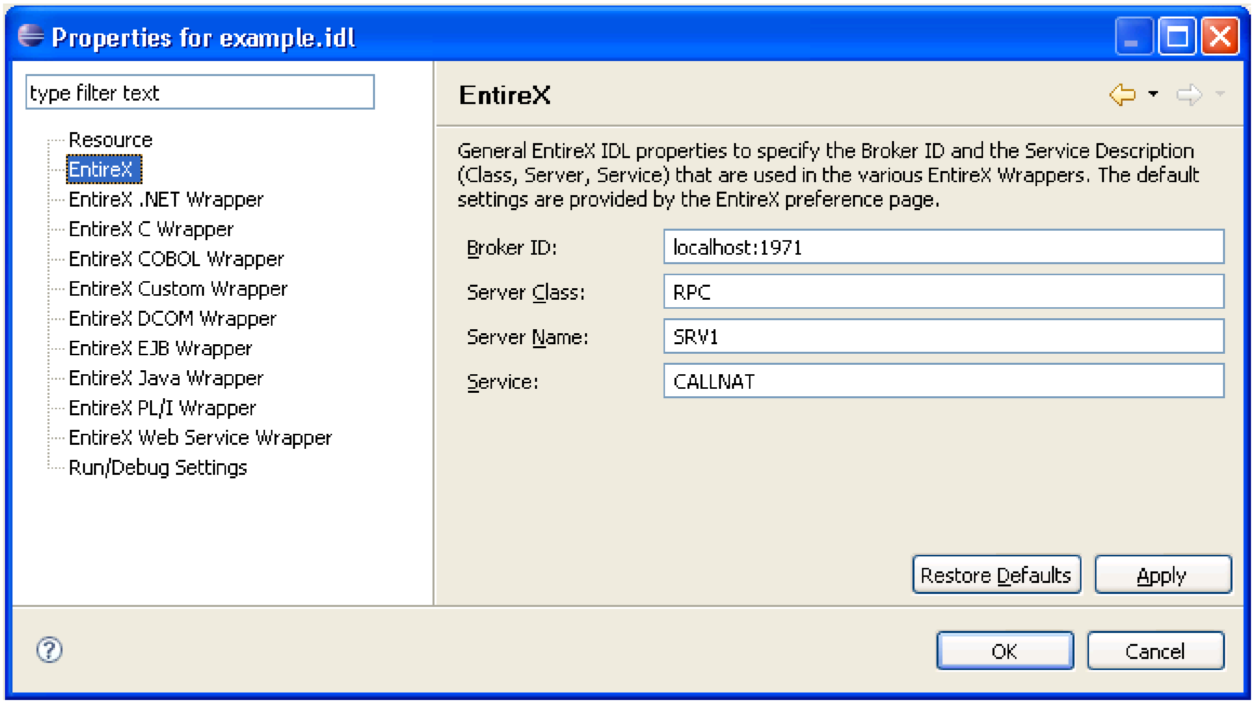
Validate Start the Validation

Choose a mapping style for all programs and directions. Existing mappings will be overwritten!

Element Attribute [✓ SOAP] Customized Generate

Setting Wrapper Properties

Before you start the generation of the XMM file, select the IDL file and open the context menu to set the properties for this IDL file. The settings for an IDL file are inherited from the preferences (they provide the defaults):


Properties for example.idl

type filter text

- Resource
- EntireX
- EntireX .NET Wrapper
- EntireX C Wrapper
- EntireX COBOL Wrapper
- EntireX Custom Wrapper
- EntireX DCOM Wrapper
- EntireX EJB Wrapper
- EntireX Java Wrapper
- EntireX PL/I Wrapper
- EntireX Web Service Wrapper
- Run/Debug Settings

EntireX

General EntireX IDL properties to specify the Broker ID and the Service Description (Class, Server, Service) that are used in the various EntireX Wrappers. The default settings are provided by the EntireX preference page.

Broker ID:	localhost:1971
Server Class:	RPC
Server Name:	SRV1
Service:	CALLNAT

To set the general options, path settings and XML options

1 Select **Preferences...** in the Windows menu of Eclipse.

2 Select **Software AG > EntireX**.

3 Select the tab **XML**.

4 Fill in the screen as follows:

Screen Item	Description
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 XML encoding	Check this box to enable the XML/SOAP Wrapper to use same encoding for the incoming document as for the outgoing document.

5 Confirm the settings with **OK**.

After setting the general properties, set the properties specific to the IDL file. The settings for the XML/SOAP Wrapper classes are on the two tab pages: General and XML.

To set the IDL properties

1 In the file tree browser, select the IDL file to be processed.

2 To display the **Properties** window, use either the context menu or the **File** menu and choose **Properties**.

The subsequent screen will show the defaults you entered with **Tools > Options.**.

3 Use this page to specify the Broker ID, the server address (**Server class**, **Server name**, and **Service**) and other properties. If you do not enter Broker ID, server name, etc., the defaults (as entered in the **Tools** / **Options** menu) will be used.

4 Select the **XML** tab.

5 Complete the screen as follows:

Screen Item	Description
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 XML encoding	Check this box to enable the XML/SOAP Wrapper to use the same encoding for the incoming document as that used for the document sent.

6 Confirm the settings with **OK**.

Generating an XMM File

To generate the XMM file, use the *EntireX Workbench*.

The IDL file, the XML file and the properties are used to generate an XMM file.

To generate the XMM file

1 Specify an IDL-XML mapping.

2 Save the mapping.

If you are using the default settings of the *EntireX Workbench*, an XMM file is saved. The name of the XMM file is `<idl file name>`*.xmm*

Caution: If you modify the IDL file, the IDL-XML mapping file or the properties, you must re-generate the XMM file.

Default Values Used by the XML/SOAP Runtime

The XML/SOAP Runtime uses standard default values for all parameters (for EntireX RPC) and elements/attributes (for XML documents) if no value can be retrieved and no user default value is defined. The table below shows the standard default values for XML and RPC.

Value	Description
A(*n*)	XML: Empty string with length 0 RPC: String with *n* blanks
AV	Empty string with length 0
B(*n*)	String with *n* zeros
BV	Empty field with length 0
K(*n*)	XML: Empty string with length 0 RPC: String with *n* blanks
KV	Empty string with length 0
L	false
I1	0
I2	0
I4	0
F4	0.0
F8	0.0
N	0.0
P	0.0
D	new Date()
T	new Date()
U(*n*)	XML: Empty Unicode string with length 0 RPC: Unicode String with *n* blanks
UV	XML: Empty Unicode string with length 0

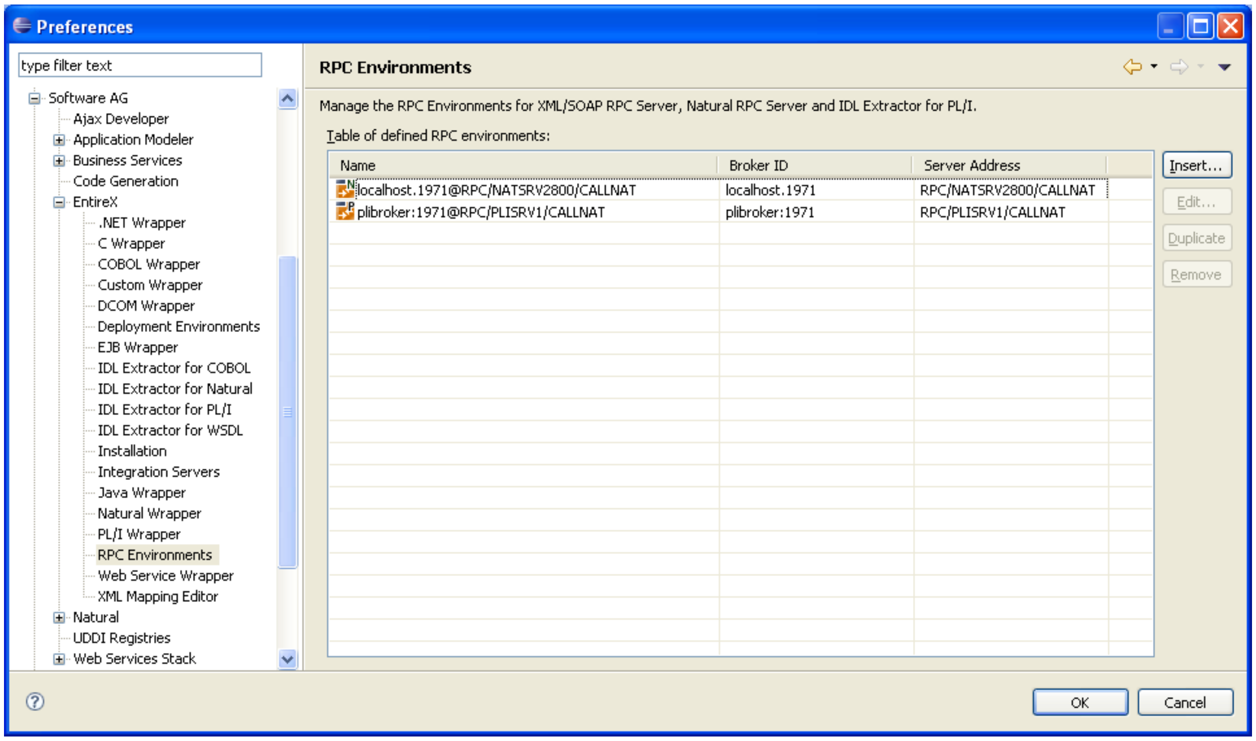
4 RPC Environment Manager

The RPC environment is managed on the RPC environment preference page. The RPC environments can be created, edited and removed. There are several types of RPC environment: Natural, PL/I and XML/SOAP. The RPC environment type will be used to prepare the selection lists of the following wizards:

- Natural RPC Server
- IDL Extractor for PL/I
- XML/SOAP RPC Server

Use the *RPC Environment Monitor* to check the availablity of each RPC environment.

Using these wizards, you can add new RPC environments of the respective type. To manage these RPC environments, open the **Preferences** page.



Manage the RPC Environments for XML/SOAP RPC Server, Natural RPC Server and IDL Extractor for PL/I.

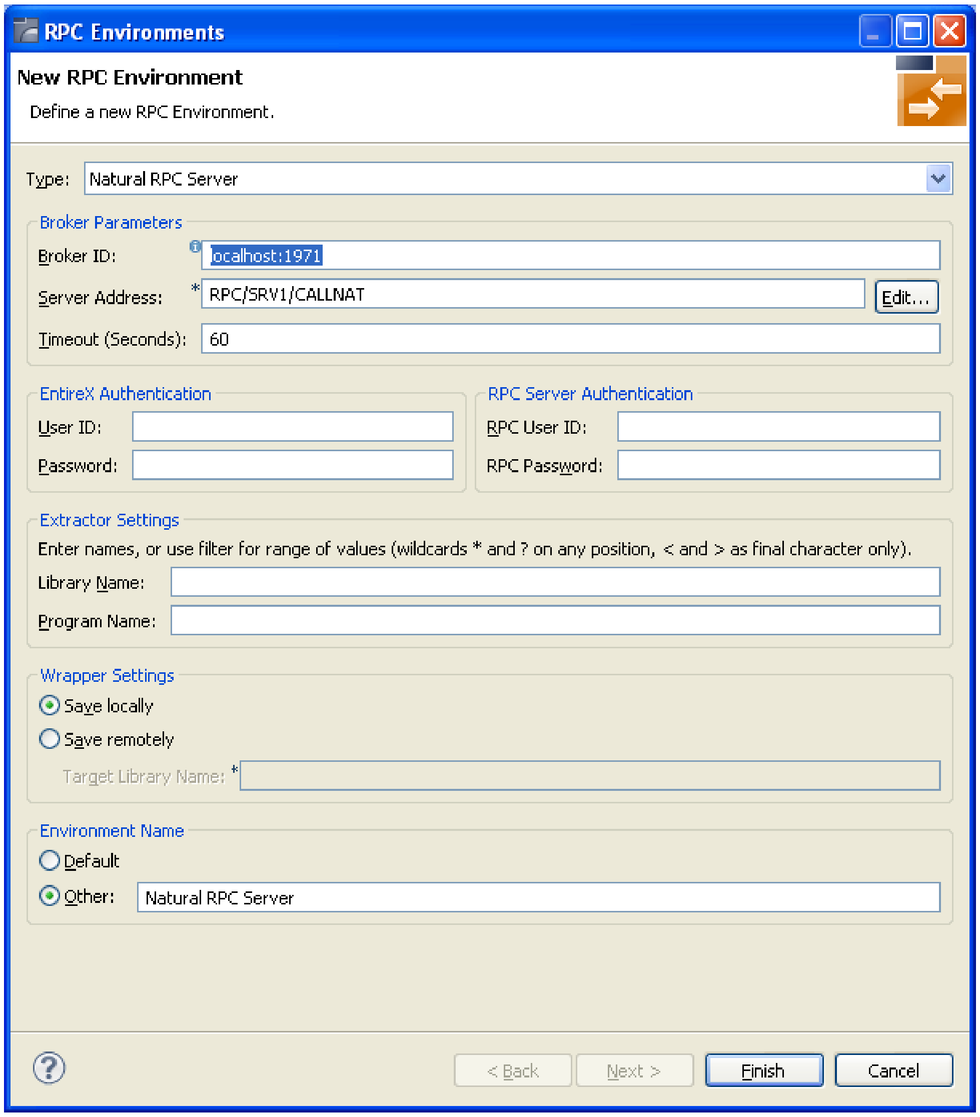
Table of defined RPC environments:

Name	Broker ID	Server Address
localhost.1971@RPC/NATSRV2800/CALLNAT	localhost.1971	RPC/NATSRV2800/CALLNAT
plibroker:1971@RPC/PLISRV1/CALLNAT	plibroker:1971	RPC/PLISRV1/CALLNAT

To edit an existing RPC environment, select the table row and press **Edit...**. If multiple entries are selected, the first entry is used.

To remove an RPC environment, select the table row and press **Remove**. You can select multiple environments.

To create a new RPC environment, choose **Insert...**.



New RPC Environment

Define a new RPC Environment.

Type: Natural RPC Server

Broker Parameters

Broker ID: localhost:1971

Server Address: * RPC/SRV1/CALLNAT

Timeout (Seconds): 60

EntireX Authentication

User ID:

Password:

RPC Server Authentication

RPC User ID:

RPC Password:

Extractor Settings

Enter names, or use filter for range of values (wildcards * and ? on any position, < and > as final character only).

Library Name:

Program Name:

Wrapper Settings

[✓ Save locally] Save remotely

Target Library Name: *

Environment Name

Default [✓ Other:] Natural RPC Server

Choose the **Type** and enter the required fields: **Broker ID**, **Server Address** and a unique **Environment Name**, which will have the default format *brokerID@serverAddress*. The given **Timeout** value must be in the range from 1 to 9999 seconds (default: 60).

EntireX Authentication describes the settings for the broker, and **RPC Server Authentication** describes the settings for the RPC server.

The **Extraction Settings** are used for the IDL Extractors (Natural and PL/I) only. Use them to specify the name of the **Dataset/Library** and the **Member/Program** name.

The **Wrapper Settings** are used for Natural Wrapper only, and you can specify the operation type and target library name (not available for **Save locally**).

5 RPC Environment Monitor

The RPC Environment Monitor is part of the EntireX Workbench. It is an Eclipse view that provides a quick overview of the availability of the defined RPC environments in your workspace.

To open the RPC Environment Monitor from the EntireX perspective

- Choose **Window > Show View > RPC Environment Monitor**.

To open the RPC Environment Monitor from a non-EntireX perspective

- Choose **Window > Show View > Other > Software AG > RPC Environment Monitor**.

The RPC environments are managed on the **Preference** page. See *RPC Environment Manager*.

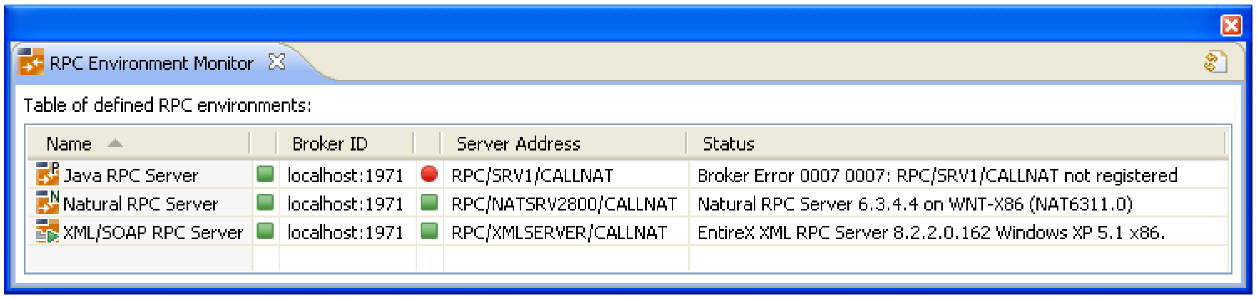


Table of defined RPC environments:

Name		Broker ID		Server Address	Status
Java RPC Server	🟩	localhost:1971	🔴	RPC/SRV1/CALLNAT	Broker Error 0007 0007: RPC/SRV1/CALLNAT not registered
Natural RPC Server	🟩	localhost:1971	🟩	RPC/NATSRV2800/CALLNAT	Natural RPC Server 6.3.4.4 on WNT-X86 (NAT6311.0)
XML/SOAP RPC Server	🟩	localhost:1971	🟩	RPC/XMLSERVER/CALLNAT	EntireX XML RPC Server 8.2.2.0.162 Windows XP 5.1 x86.

The status check starts when the view is opened. To force an additional check, choose **Refresh** from the **Views** toolbar. The status check can be cancelled in the dialog that appears or within the Eclipse progress view. When the check is complete or if it cancelled, the following symbols indicate the status of the corresponding item. The table will be reloaded every time a status check is started to make sure all stored RPC environments are available.

Symbol	Status
🟩	Running.
🔴	Not running.
△	Unknown (at the beginning of the check or if the check was cancelled).

Note: Additional status information (including error messages) is displayed when refreshing the view (by a ping command to all specified RPC servers).

6 EntireX XML Tester

Introduction to the XML Tester

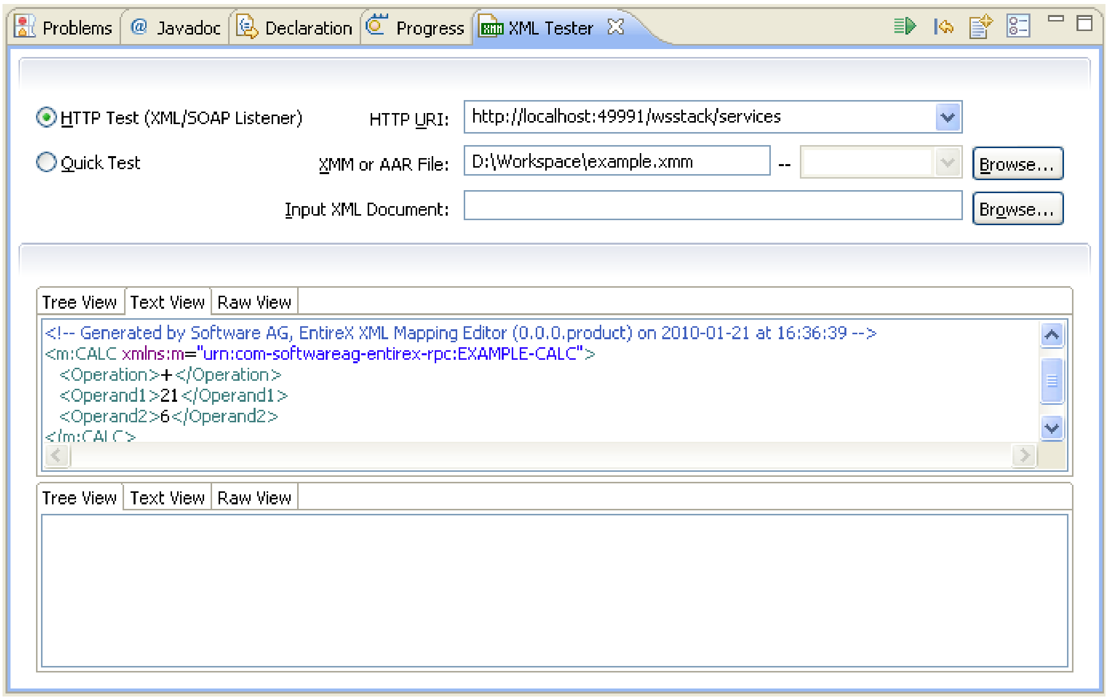
Using the XML Tester you can send an XML document to the EntireX XML/SOAP Listener.

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.



(● HTTP Test (XML/SOAP Listener)) ○ Quick Test

HTTP URI: http://localhost:49991/wsstack/services

XMM or AAR File: D:\Workspace\example.xmm --

Input XML Document:

```
<!-- Generated by Software AG, EntireX XML Mapping Editor (0.0.0.product) on 2010-01-21 at 16:36:39 -->
<m:CALC xmlns:m="urn:com-softwareag-entirex-rpc:EXAMPLE-CALC">
  <Operation>+</Operation>
  <Operand1>21</Operand1>
  <Operand2>6</Operand2>
</m:CALC>
```

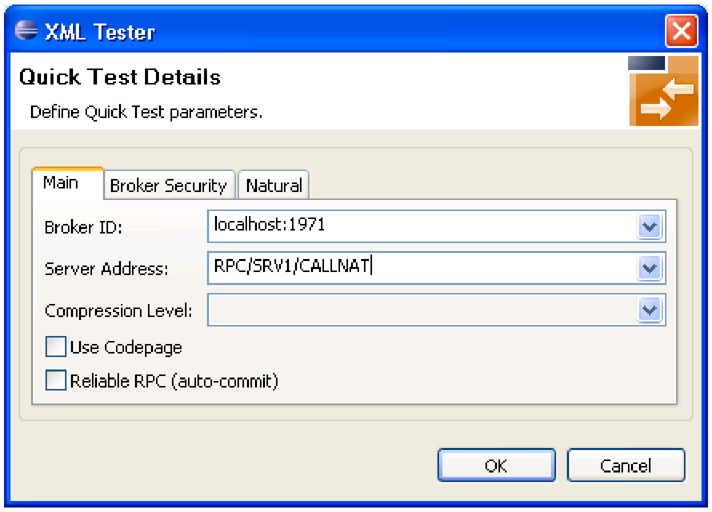
XML Tester Options

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*.



Broker ID: localhost:1971

Server Address: RPC/SRV1/CALLNAT

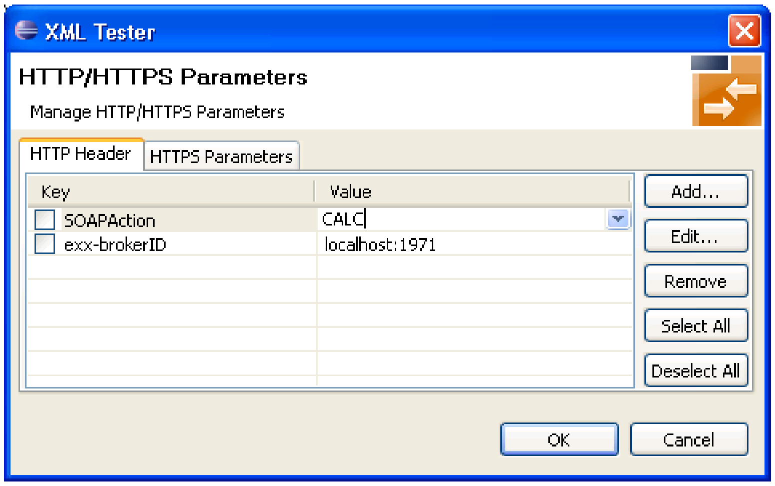
Compression Level:

☐ Use Codepage

☐ Reliable RPC (auto-commit)

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.

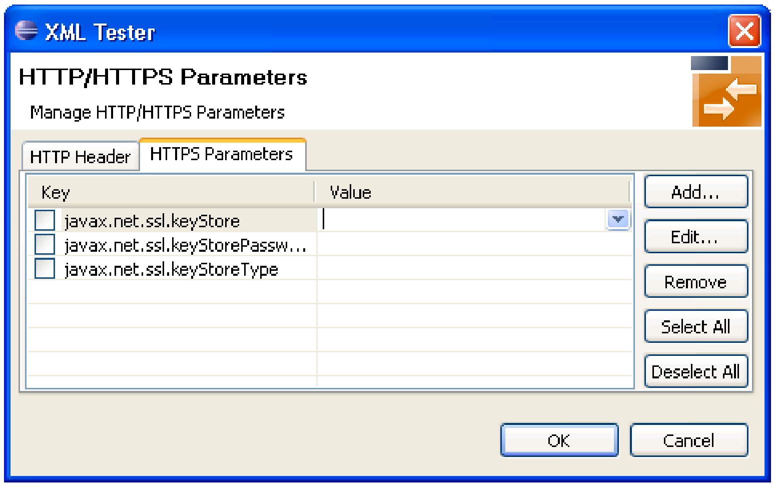

XML Tester

HTTP/HTTPS Parameters

Manage HTTP/HTTPS Parameters

Key	Value
☐ SOAPAction	CALC
☐ exx-brokerID	localhost:1971

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.


XML Tester

HTTP/HTTPS Parameters

Manage HTTP/HTTPS Parameters

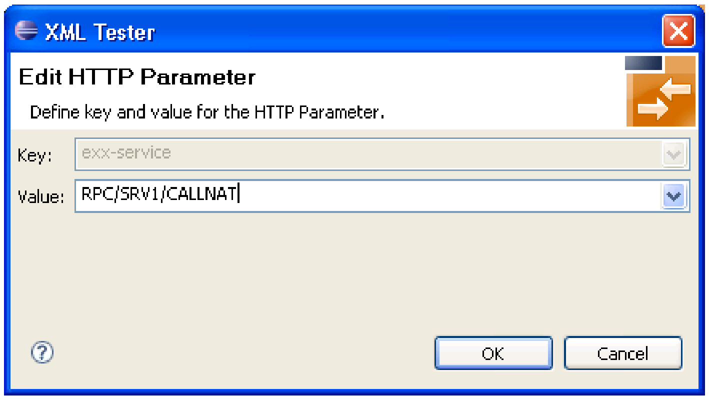
Key	Value
☐ javax.net.ssl.keyStore	
☐ javax.net.ssl.keyStorePassw...	
☐ javax.net.ssl.keyStoreType	

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.



Define key and value for the HTTP Parameter.

Key: exx-service

Value: RPC/SRV1/CALLNAT

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:

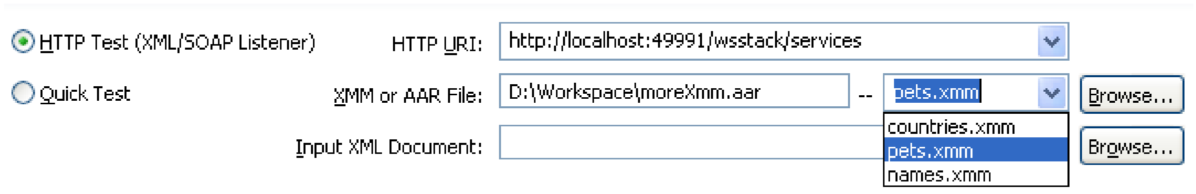
  In the XML Mapping Editor, switch to the **Overview** page, choose **Quick Test** or **HTTP Test** in the testing section.

  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.


(●) HTTP Test (XML/SOAP Listener)
( ) Quick Test

HTTP URI: http://localhost:49991/wsstack/services
XMM or AAR File: D:\Workspace\moreXmm.aar -- pets.xmm
Input XML Document:

countries.xmm
pets.xmm
names.xmm

2 Select the testing interface (Java or HTTP):

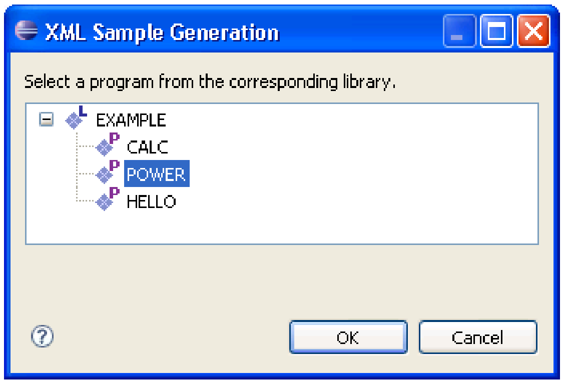
- Java interface: the **XMM or AAR File** field must contain the full path and name of an XMM or AAR archive. Choose the **Quick Test** option to use this interface.
- HTTP interface: enter the name of the URI of the running EntireX XML/SOAP Listener. Choose the **HTTP Test** option to use this interface.

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.

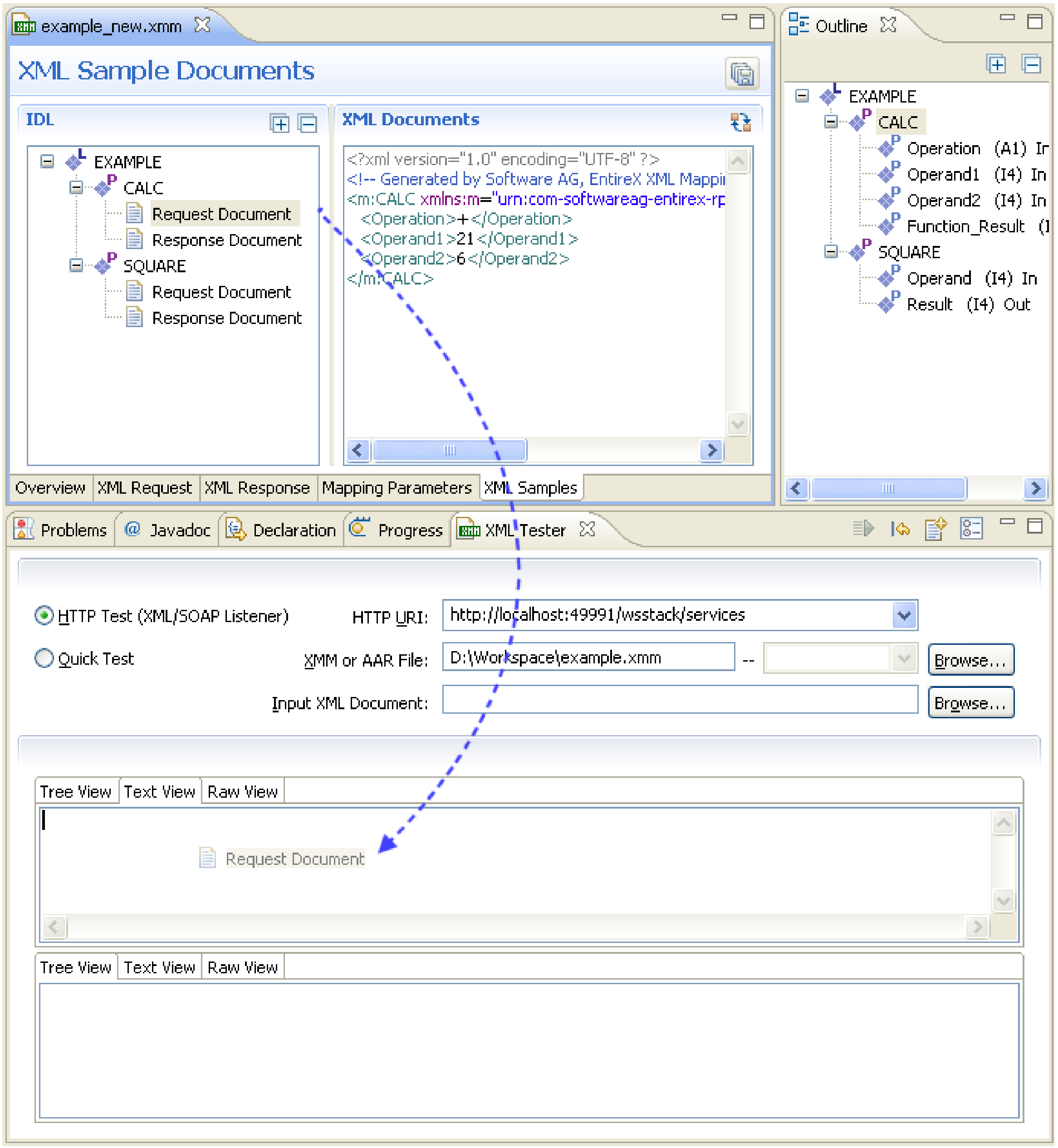
Example dialog:


XML Sample Generation

Select a program from the corresponding library.

EXAMPLE
CALC
POWER
HELLO

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



```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Software AG, EntireX XML Mappi
<m:CALC xmlns:m="urn:com-softwareag-entirex-rp
  <Operation>+</Operation>
  <Operand1>21</Operand1>
  <Operand2>6</Operand2>
</m:CALC>
```

HTTP URI: http://localhost:49991/wsstack/services

XMM or AAR File: D:\Workspace\example.xmm

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:

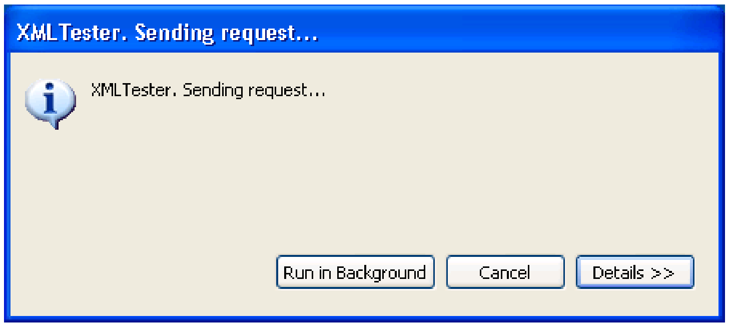
- HTTP Header
- HTTPS Parameters

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.



XMLTester. Sending request...

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

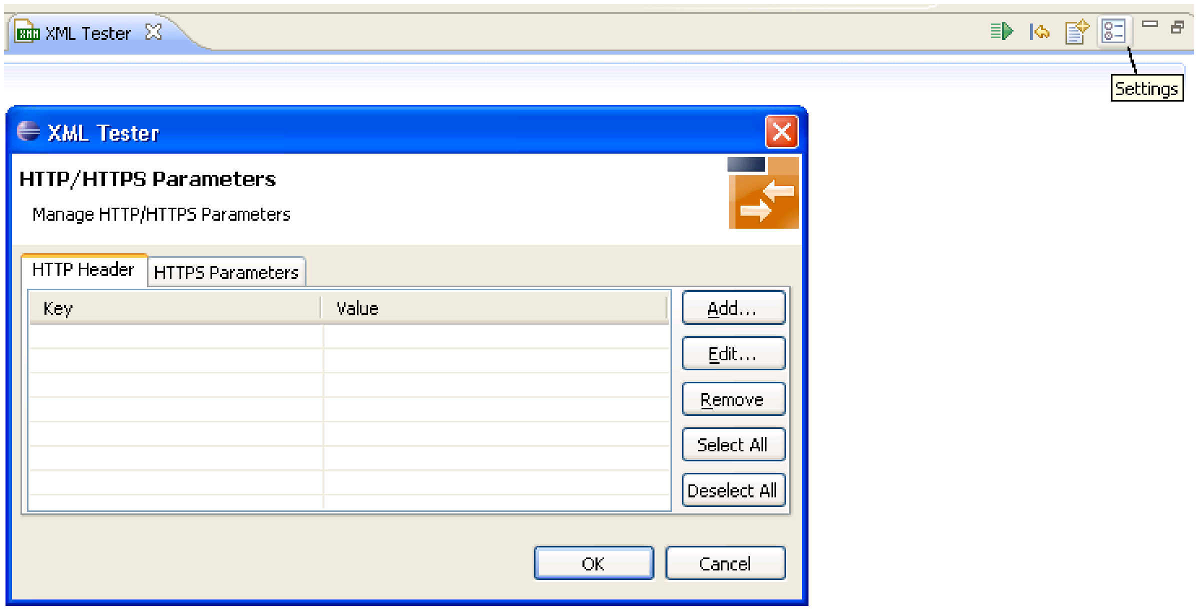
7 When a response is received, it is displayed in the output area. Errors are also displayed in the output area.

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.

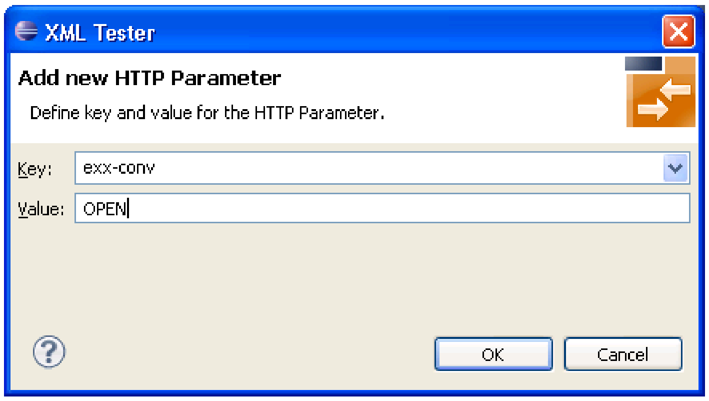
XML Tester for Conversational RPC

To use the XML Tester with conversational RPC

1 Open **Settings**.

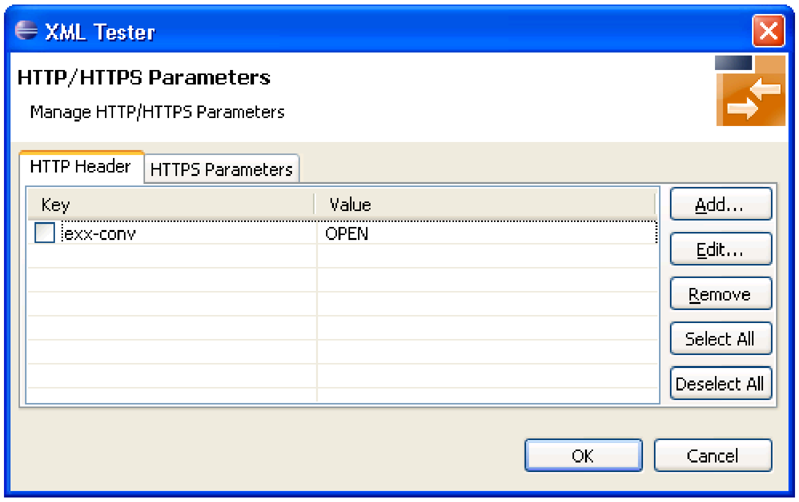


2 Add key `exx-conv` and set value to "OPEN".



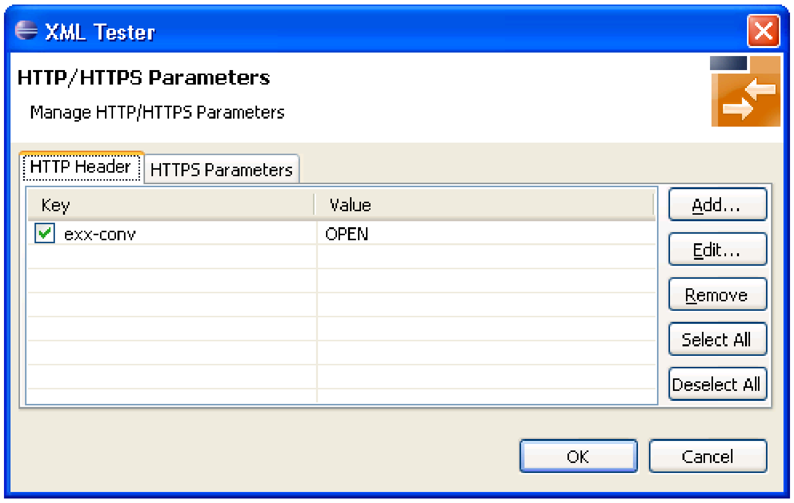
Key: exx-conv
Value: OPEN

3 Press **OK**.



Key	Value
☐ exx-conv	OPEN

4 Select `exx-conv`.



Key	Value
✓ exx-conv	OPEN

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.

7 Using the XML/SOAP Wrapper 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/SOAP Wrapper.

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 name of the IDL file includes the project name. In the example, the project *Demo* is used. If the IDL file name describes a file inside the Eclipse workspace, the name is case-sensitive.

If the first part of the IDL file name is not a project name in the current workspace, the IDL file name is used as a relative (based on the IDL file) or absolute file name in the file system. Thus, the IDL files do not need to be part of an Eclipse project.

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

Further Examples

Windows Example 1

```
<workbench> -map:soap C:\Temp\example.idl
```

Uses the IDL file *C:\Temp\example.idl* and generates the XML mapping file *example.xmm* in parallel to the IDL. Slashes and backslashes are permitted in the file name. Output to standard output:

```
Using workspace file: C:\myWorkspace\.
Processing IDL file: C:\Temp\example.idl
Store XML mapping file: C:\Temp\example.xmm
Exit value: 0
```

Windows Example 2

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

Generates XML mapping files for all IDL files in project /*Demo*.

Linux Example 1

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

If the project *Demo* exists in the workspace and *example.idl* exists in this project, this file is used. Otherwise, */Demo/example.idl* is used from file system.

Linux Example 2

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

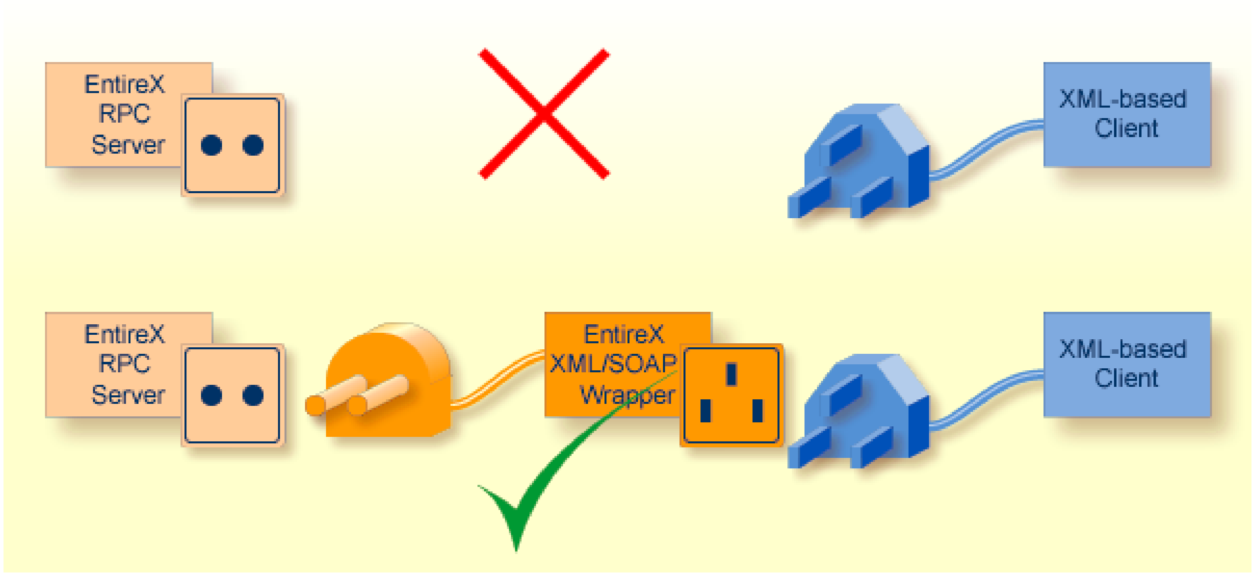
Generates XML mapping files for all IDL files in project *Demo* (or in folder /*Demo* if the project does not exist).

8 Tracing the XML/SOAP Runtime

See *Tracing the XML/SOAP Runtime* in the UNIX and Windows administration documentation.

9 Introduction to Writing Applications with the XML/SOAP Wrapper

Connecting between XML-based Clients and an EntireX RPC Server



Publish an Existing EntireX RPC Server for XML-based Clients

You have an existing EntireX RPC server and want to extend its availability to XML-based clients (e.g. offer the functionality of your server as a Web service). The simplest approach is to use the EntireX XML/SOAP Wrapper to convert your XML documents to EntireX RPCs and vice versa. Then the XML-based client will appear to communicate with an XML-based server.

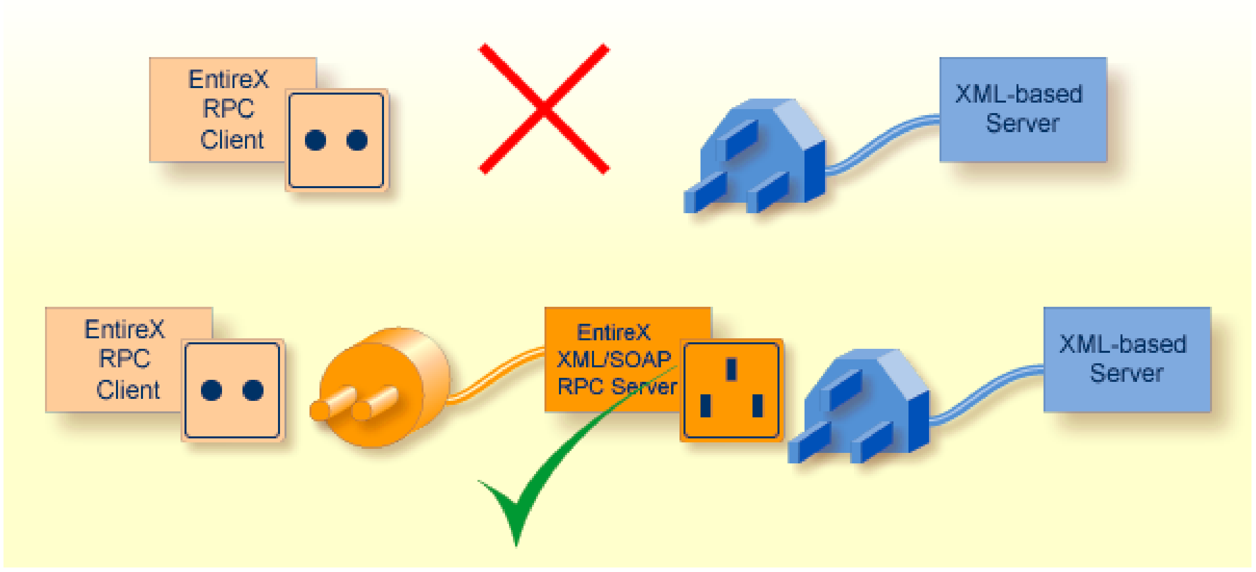
See *Connect an Existing EntireX RPC Server with an XML-based Client*.

Use your XML-based Client and Connect to an EntireX RPC Server

You have an XML-based client that is to communicate with an EntireX RPC server or access a component accessible via an EntireX RPC server. The simplest approach is to use the EntireX XML/SOAP Wrapper to convert your XML documents to EntireX RPCs and vice versa. Then an XML-based client will appear to communicate with an XML-based server (your EntireX RPC server).

See *Build an EntireX RPC Server and Use an Existing XML-based Client with It*.

Connecting EntireX Clients and XML-based Server



Involve an XML-based Server in your EntireX Application

You know or have written XML-based servers (e.g. Web services) and want to involve them in your EntireX application. The simplest approach is to use the EntireX XML/SOAP RPC Server to convert your EntireX RPCs to XML documents and vice versa; the EntireX XML/SOAP RPC Server handles the HTTP communication.

See *Build an EntireX RPC Client and Use an Existing XML-based Server*.

Connect your RPC Client and the XML-based Server

You have an EntireX RPC client and want to implement your server application as an XML-based server and you want the EntireX RPC client to use this XML-based server. The simplest approach is to use the EntireX XML/SOAP RPC Server to convert your EntireX RPCs to XML documents and vice versa; the EntireX XML/SOAP RPC Server will handle the HTTP communication.

See *Connect an Existing EntireX RPC Client to an XML-based Server*.

10 Writing Advanced Applications with the XML/SOAP Wrapper

XML/SOAP Listener

With the XML/SOAP Listener you can define parameters inside the payload of a message. We recommend this approach rather than HTTP parameters. Define the setting in the SOAP header and under the first tag of XML document as follows:

SOAP Documents

```
...
<soap-env:SOAPHeader>
  <exx:EntireX xmlns:exx="urn:com.softwareag.entirex.xml.rt">
    <!-tags with parameter setting e.g: -->
    <exx-natural-library>libraryname</exx-natural-library>
    <exx-natural-security>true</exx-natural-security>
  </exx:EntireX>
...
</soap-env:SOAPHeader>
...
```

XML Documents

```
<root-tag>
  <exx:EntireX xmlns:exx="urn:com.softwareag.entirex.xml.rt">
    <!-tags with parameter setting e.g: -->
    <exx-natural-library>libraryname</exx-natural-library>
    <exx-natural-security>true</exx-natural-security>
  </exx:EntireX>
...
</root-tag>
```

Natural Logon or Changing the Library Name

The library name sent with the RPC request to the EntireX RPC or the Natural RPC Server is specified in the IDL file. See `library-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation. When the RPC is executed, this library name can be overwritten.

XML/SOAP Wrapper (Java API)

To overwrite the library

- An EntireX XML/SOAP Wrapper client (Java API) must call the `setLibraryName` method.

To force the library to be considered by Natural RPC Servers

- Call the `setNaturalLogon` method.

XML/SOAP Listener

To overwrite the library

- Use the parameter `exx-natural-library`.

To force the library to be considered by Natural RPC Servers

- Set the parameter `exx-natural-security` to "true".

Caution: Natural RPC Servers and EntireX RPC Servers behave differently regarding the library name.

See also *Natural Logon or Changing the Library Name* under *Common Features of Wrappers and RPC-based Components*.

Using RPC Compression

EntireX and Natural RPC support a feature called RPC compression to reduce network traffic. The default for compression is on. See also *RPC Compression* under *Common Features of Wrappers and RPC-based Components*.

XML/SOAP Wrapper (Java API)

▶ To switch compression on and off

- Use the `setCompression` method of the class `XMLRPCService` inherited from class `RPCService` in the Javadoc documentation of the Java ACI.

▶ To check the current compression setting

- Use the `getCompression` method of the class `XMLRPCService` inherited from class `RPCService` in the Javadoc documentation of the Java ACI.

XML/SOAP Listener

▶ To switch compression on and off

- Use the parameter `exx-compression`. Possible values: True, False.

Using Conversational RPC

It is assumed that you are familiar with the concepts of conversational and non-conversational RPC. See also *Conversational RPC* under *Common Features of Wrappers and RPC-based Components*.

For conversational RPC you need an instantiated conversation object. See `Conversation` in the Javadoc documentation of the Java ACI. Conversational RPC is enabled by passing a reference to this object to the method `setConversation`. See `setConversation` in the Javadoc documentation of the Java ACI. Different stubs can participate in the same conversation if they use the same instance of a Conversation object. An RPC conversation is terminated by calling either the `closeConversation` method or the `closeConversationCommit` method for one stub.

XML/SOAP Wrapper (Java API)

To enable conversational RPC

- Create a Conversation object and set this with `setConversation` on the wrapper object.

  Different wrapper objects can participate in the same conversation if they use the same instance of a conversation object.

To abort a conversational RPC communication

- Call the `closeConversation` method.

To close and commit a conversational RPC communication

- Call the `closeConversationCommit` method.

XML/SOAP Listener

Conversations can only be used in connection with sessions. If the session is interrupted, the conversation will be deleted.

To use conversational RPC

- Use the parameter `exx-conv` with the value OPEN.

To continue conversational RPC

- Pick up the parameter `exx-sessionID` in response and set the parameter as HTTP parameter or in the same way as in the response document inside the request document.

To abort a conversational RPC communication

- Use the parameter `exx-conv` with the value BACKOUT.

To close and commit a conversational RPC communication

- Use the parameter `exx-conv` with the value COMMIT.

See also *XML Tester for Conversational RPC*.

Caution: Natural RPC Servers and EntireX RPC Servers behave differently when ending an RPC conversation.

See also *Conversational RPC* under *Common Features of Wrappers and RPC-based Components*.

Using Natural Security

A Natural RPC Server may run under Natural Security to protect RPC requests. See also *Natural Security* under *Common Features of Wrappers and RPC-based Components*.

XML/SOAP Wrapper (Java API)

To authenticate an EntireX XML/SOAP Wrapper client (Java API) against Natural Security

- Specify a user ID and password in the logon method of class `Broker`.

  If different user IDs and/or passwords are used for EntireX Security and Natural Security, use the methods `setRPCUserId` or `setRPCPassword` to set the user IDs and/or passwords for Natural Security.

To force an EntireX XML/SOAP Wrapper client (Java API) to log on to a specific Natural library

1. Call the `setLibraryName` method.
2. Call the `setNaturalLogon` method.

See also ***Natural Logon or Changing the Library Name***.

XML/SOAP Listener

To authenticate against Natural Security

- Specify the parameters `exx-userID` and `exx-password`.

  If a different user ID or password is used for EntireX Security and Natural Security, use the parameters `exx-rpc-userID` and `exx-rpc-password` to set the user ID or password for Natural Security.

To force a logon to a specific Natural library

1. Use the parameter `exx-natural-library`.
2. Set the parameter `exx-natural-security` to True.

See also ***Natural Logon or Changing the Library Name***.

Using Compression

Java-based EntireX applications (including applications using classes generated by the Java Wrapper) may compress the messages sent to and received from the broker. There is a general way to enable compression using broker ID, and another way that depends on whether you use the XML/SOAP Wrapper or the XML/SOAP Listener.

Using Broker ID

You may append the keyword `compresslevel` with one of the values below to the Broker ID.

Examples

- localhost:1971?compresslevel=BEST_COMPRESSION
- localhost?poolsize=4&compresslevel=9

Both examples set the compression level to 9.

XML/SOAP Wrapper (Java API)

Using setCompressionLevel()

Set the compression level with the method `setCompressionLevel()` as an integer or a string argument.

You can use the constants defined in class `java.util.zip.Deflater`.

If the string

- starts with Y, compression is switched on with level 6,
- starts with N, compression is switched off (level 0).

Permitted values are the integers 0 - 9 and the corresponding strings:

Compression	Level
BEST_COMPRESSION	9
BEST_SPEED	1
DEFAULT_COMPRESSION	6
DEFLATED	8
NO_COMPRESSION	0

XML/SOAP Listener

To set the compression level

- Use the parameter `exx-compressLevel`. The values are described in the section above (*XML/SOAP Wrapper (Java API)*).

Using EntireX Security

Java-based EntireX applications that require security can use the security services offered by EntireX Security. See also *Overview of EntireX Security* in the EntireX Security documentation | `EntireXSecurity` in the Javadoc documentation of the Java ACI.

Use the methods for security, which are included in class `Broker`. See `Broker` in the Javadoc documentation of the Java ACI. The two alternatives using security are:

- using EntireX Security
- using your own security implementation

XML/SOAP Wrapper (Java API)

To use EntireX Security, call `Broker.useEntireXSecurity()` for a Broker object. You can set the encryption level with this call and you can enable the auto mode. The encryption level allows the values `ENCRYPTION_LEVEL_NONE`, where the message is not encrypted, `ENCRYPTION_LEVEL_BROKER`, where the message is encrypted on the way to the EntireX Broker, and `ENCRYPTION_LEVEL_TARGET`, where the message is encrypted the whole way to the target. The auto mode specifies that the Broker object uses the EntireX Security as needed by the EntireX Broker. If the EntireX Broker uses security, the EntireX Security object is used by the Broker object. The method `useEntireXSecurity()` must be called before the first call of `logon()`, which has to use a password. The security object cannot change during a session with the EntireX Broker.

To use your own security implementation, implement the interface `BrokerSecurity`. This implementation must have an accompaning security exit for the EntireX Broker. See *Using Sample Security*

Exits for Broker Security. Call the methods `setSecurity()` with the security object and set encryption level or auto mode in the same way as the `useEntireXSecurity()` methods.

XML/SOAP Listener

The parameter `exx-use-security` (true, false) is responsible for EntireX Security. Set the encryption level with the required parameter `exx-encryption-level` (0,1,2).

HTTP Proxy Settings

If the target server of your Web service has to be reached through a firewall, set and adjust to your needs the following properties:

- `-Dhttp.proxySet=true`
- `-Dhttps.proxySet=true`
- `-Dhttp.proxyHost=httpprox.mydomain.org`
- `-Dhttps.proxyHost=sslprox.mydomain.org`
- `-Dhttp.proxyPort=8080`
- `-Dhttps.proxyPort=443`
- `-Dhttp.nonProxyHosts=*mydomain.org|localhost`
- `-Dhttps.nonProxyHosts=*mydomain.org|localhost`
- `-Dhttp.proxyUser`
- `-Dhttps.proxyUser`
- `-Dhttp.proxyPassword`
- `-Dhttps.proxyPassword`

Add the proxy settings to the start script.

XML/SOAP RPC Server with HTTP Basic Authentication

The XML/SOAP RPC Server uses basic authentication for a Web service if the configuration contains the attribute `basicAuthentication` block in <TargetServer>. Basic authentication is used for all calls associated with defined XMM files for the <TargetServer>.

Basic authentication can be used with fixed credentials or credentials set from the client application:

- If <TargetServer> contains attributes `user` and `password`, these settings are used for basic authentication.

- Otherwise the client application must provide the credentials: Enable Natural logon and set RPC user ID and RPC password.

See *Configuration File for the XML/SOAP RPC Server* in the UNIX and Windows administration documentation.

XML/SOAP Listener with HTTP Basic Authentication and UsernameToken Authentication for EntireX Authentication

The XML/SOAP Listener allows you to use the user credentials from the incoming request by means of Basic Authentication or `UsernameToken`. The same credentials are used for EntireX Broker authentication and (Natural) RPC Server authentication. This means you need to make some settings for the EntireX Web service in Web Service Wizard and Configuration Editor.

Note: `UsernameToken` is part of WS-Security. See **WS-Security `UsernameToken` Specification**. See also *Example: Setting up an EntireX Client to Consume a Secured Web Service* in the IDL Extractor for WSDL documentation.

The priority of credentials settings is as follows:

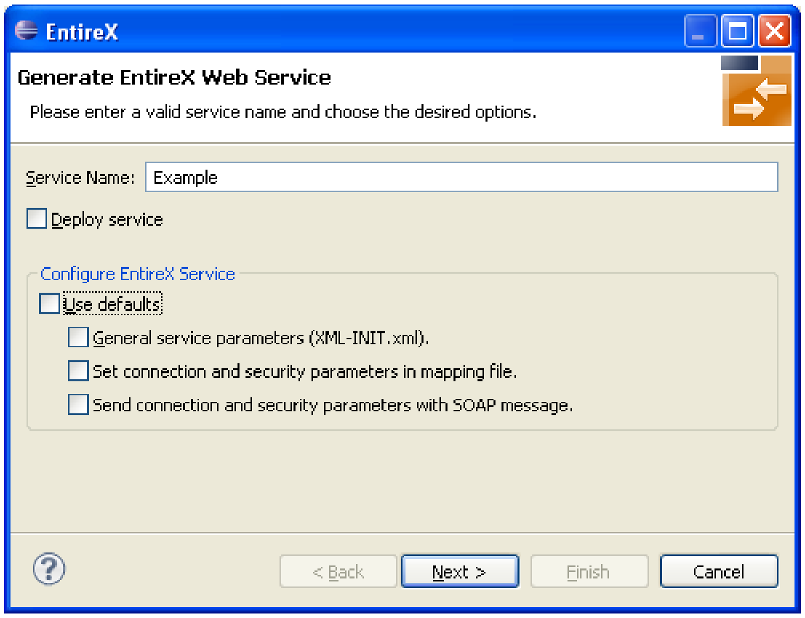
1. `exx-userID`, `exx-password`, `exx-rpc-userID`, `exx-rpc-password` (highest priority)
2. `UsernameToken`
3. Basic Authentication (lowest priority)

To use the XML/SOAP Listener with Basic Authentication and `UsernameToken` Authentication

1 Select an IDL file or XMM file.

2 Choose **Generate Web Service from Software AG...**.

3 Disable check box **Use Defaults**.



Generate EntireX Web Service

Please enter a valid service name and choose the desired options.

Service Name: Example

[ ] Deploy service

Configure EntireX Service

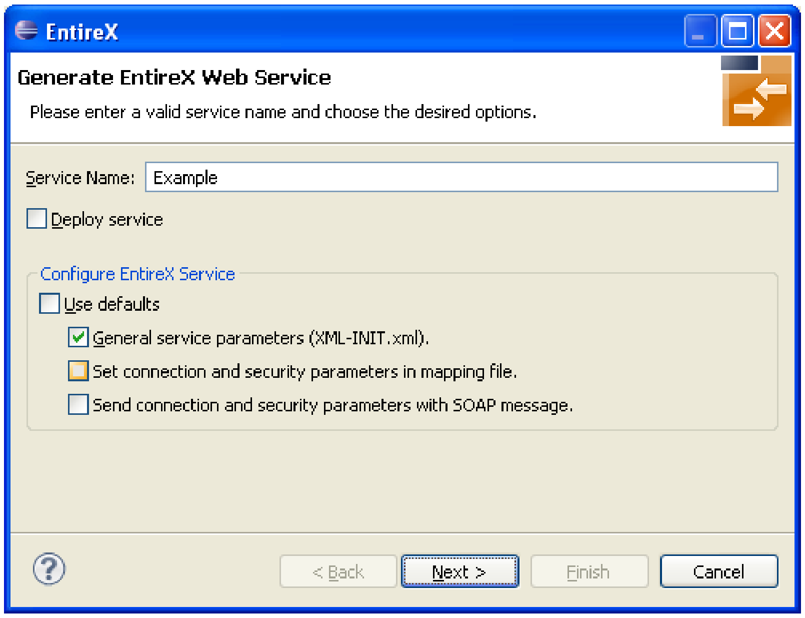
[ ] Use defaults

[ ] General service parameters (XML-INIT.xml).

[ ] Set connection and security parameters in mapping file.

[ ] Send connection and security parameters with SOAP message.

4 Enable at least **General service parameters...**.



Generate EntireX Web Service

Please enter a valid service name and choose the desired options.

Service Name: Example

[ ] Deploy service

Configure EntireX Service

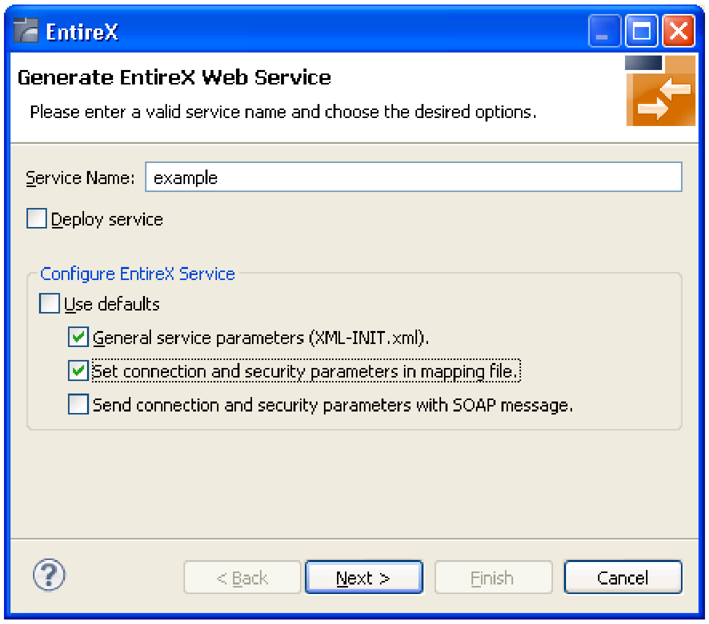
[ ] Use defaults

[✓] General service parameters (XML-INIT.xml).

[ ] Set connection and security parameters in mapping file.

[ ] Send connection and security parameters with SOAP message.

5 If using EntireX Security or Natural Security, enable **Set connection and security...** too.


EntireX

Generate EntireX Web Service

Please enter a valid service name and choose the desired options.

Service Name: example

☐ Deploy service

Configure EntireX Service

☐ Use defaults

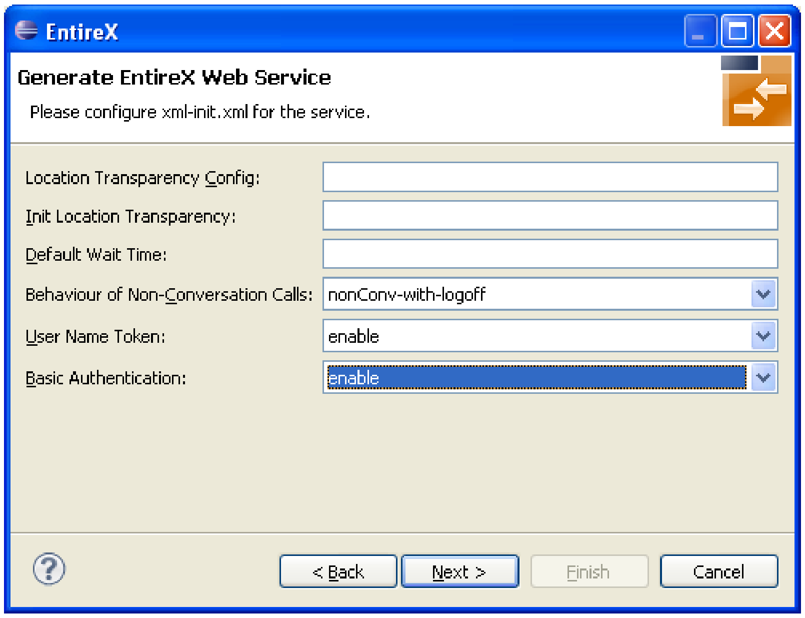
☑ General service parameters (XML-INIT.xml).

☑ Set connection and security parameters in mapping file.

☐ Send connection and security parameters with SOAP message.

6 Press **Next**.

7 Enable the required authentication. In this example, both possibilities of web service authentication are enabled.


EntireX

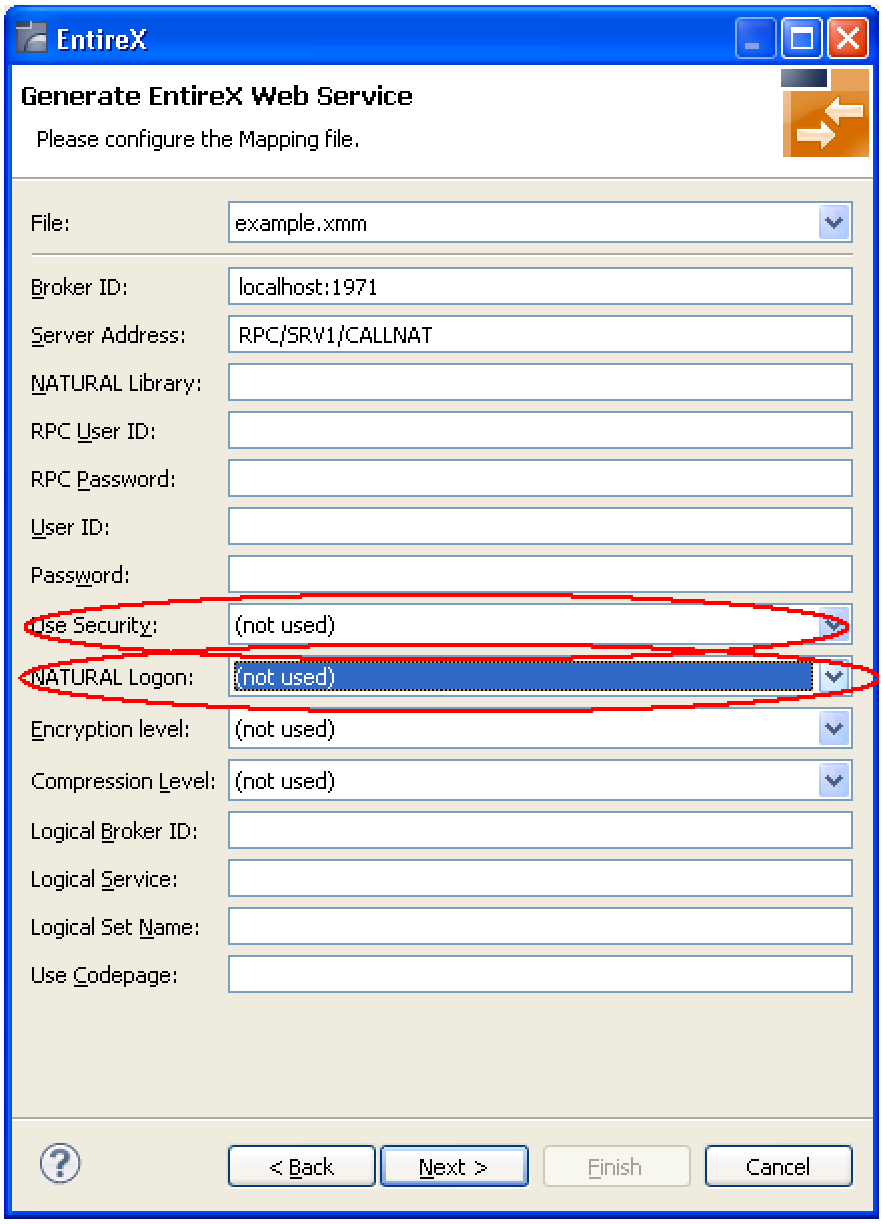
Generate EntireX Web Service

Please configure xml-init.xml for the service.

Location Transparency Config:	
Init Location Transparency:	
Default Wait Time:	
Behaviour of Non-Conversation Calls:	nonConv-with-logoff
User Name Token:	enable
Basic Authentication:	enable

8 Press **Next**.

9 The page with XMM settings appears if it was selected before (step 5). Enable the required security (EntireX Security and/or Natural Logon).



Generate EntireX Web Service

Please configure the Mapping file.

File:	example.xmm
Broker ID:	localhost:1971
Server Address:	RPC/SRV1/CALLNAT
NATURAL Library:	
RPC User ID:	
RPC Password:	
User ID:	
Password:	
Use Security:	(not used)
NATURAL Logon:	(not used)
Encryption level:	(not used)
Compression Level:	(not used)
Logical Broker ID:	
Logical Service:	
Logical Set Name:	
Use Codepage:	

10 Press **Next** and follow the wizard.

11 After generating the web service archive (extension "aar"), open the generated AAR file with the Configuration Editor (e.g. with double click).

For more information on the Configuration Editor see *Configuring Web Services*.

Using SSL or TLS with the XML/SOAP RPC Server

Using HTTPS with XML/SOAP RPC Server requires setting Java properties and changing the protocol from http to https in the configuration file. This section covers the following topics:

See also *Configuration File for the XML/SOAP RPC Server* in the UNIX and Windows administration documentation.

SSL or TLS Settings

▶ To configure SSL communication for the JRE

- Set the following properties:
  - **-Djavax.net.ssl.keyStore=<filename-without-blanks>**
    Here we keep the certificate and the private signing key of our client application, which is the EntireX XML/SOAP RPC Server.
  - **-Djavax.net.ssl.keyStorePassword=<you-should-know-it>**
    The password that protects the keystore.
  - **-Djavax.net.ssl.keyStoreType=pkcs12**
    If not jks (default).
  - **-Djavax.net.ssl.trustStore=<filename-without-blanks>**
    Here we keep the trusted certificate of the Web service host or the certificate of its signing (issuing) certificate authority.
  - **-Djavax.net.ssl.trustStorePassword=<you-should-know-it>**
    The password that protects the truststore.
  - **-Djavax.net.ssl.trustStoreType=**
    If not jks (default).

For more information about Java and SSL, see your Java documentation (JSSE documentation).

Sample Start Script

```
set CLASSPATH=.;.\classes\entirex.jar;..\WS-Stack\lib\wsstack-client.jar

set PROXYSETTINGS=-Dhttps.proxySet=true
-Dhttps.proxyHost=sslproxy.mydomain
-Dhttps.proxyPort=443
-Dhttps.nonProxyHosts="localhost"

set SSL=-Djavax.net.ssl.keyStore=C:\myKeystore.p12
-Djavax.net.ssl.keyStorePassword=myKeystorePassword
-Djavax.net.ssl.keyStoreType=pkcs12
-Djavax.net.ssl.trustStore=C:\myTrustStore.jks
-Djavax.net.ssl.trustStorePassword=myTruststorePassword

java -classpath %CLASSPATH% %SSL% %PROXYSETTING% ↩
com.softwareag.entirex.xml.rt.XMLRPCServer
```

For the changes that are required to the start script, see your Java documentation (JSSE documentation).

Configuration File Settings

Specify the fully qualified host name as TargetServer. The host name has to match the CN (Common Name) item of the host certificate.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<EntireX
xmlns="http://namespaces.softwareag.com/entirex/xml/runtime/configuration" ↩
version="8.0"
>
    <XmlRuntime Version="1">
       <BrokerInfo>
       <BrokerId>localhost:1971</BrokerId>
       <ServerAddress>RPC/XMLSRV1/JAVA</ServerAddress>
       </BrokerInfo>
       <TargetServer name="https://targethost:8080/entirex/xmlrt">
            <xmms>
                <exx-xmm name="yourFile1.xmm" />
                <exx-xmm name="yourFile2.xmm" />
            </xmms>
        </TargetServer>
    </XmlRuntime>
</EntireX>
```

Using Internationalization with EntireX XML Components

XML components support Conversion and Translation for Internationalization. If you choose Conversion (which is recommended), the correct codepage must be send as locale string to the EntireX Broker matching the encoding of the data sent. This codepage must also be a codepage supported by the broker, see *Locale String Mapping* in the internationalization documentation for information on how the broker derives the codepage from the locale string. For Translation and more details on Conversion, see *Internationalization with EntireX*.

To enable EntireX XML components to send a codepage as locale string to the EntireX Broker, they must be prepared as described below:

- XML/SOAP Wrapper (Java API)
- XML/SOAP Listener
- XML/SOAP RPC Server

XML/SOAP Wrapper (Java API)

To enable use of the encoding of an incoming XML document for Broker communication

1. Call `useCodePage(true)` on the `XMLRPCService` object. The XML/SOAP Wrapper will then use the codepage retrieved from the XML document to send data to EntireX Broker.
2. Use a stream-oriented method of `XMLRPCService` to transfer the data to XML/SOAP Wrapper.

XML/SOAP Listener

To enable use of the encoding of an incoming XML document for broker communication

- Activate the parameter `exx-use-codepage` (true/false). The XML/SOAP Wrapper will then use the codepage retrieved from the incoming XML document to send data to the EntireX Broker.

XML/SOAP RPC Server

The encoding for broker communication is defined by the parameter `codepage`.

- The locale string for broker communication can be overridden for a broker version 7.2.x and above. Instead of using the default encoding of the JVM, the given encoding is used.
- It can be forced for a broker version 7.1.x and below.
- It does not change the default encoding of your Java virtual machine.

- We recommend using UTF-8 as the file encoding for your JVM and the value LOCAL to send the default encoding of the JVM to the broker, i.e start the XML/SOAP RPC Server with `-Dcodepage=LOCAL` and `-Dfile.encoding=utf-8`. See also *Using the Abstract Codepage Name LOCAL* under *Locale String Mapping* in the internationalization documentation for more information.

The encoding for the outgoing XML document is determined by the IDL to XML Mapping. See *Defining the XML Encoding*.

Null Value Suppression

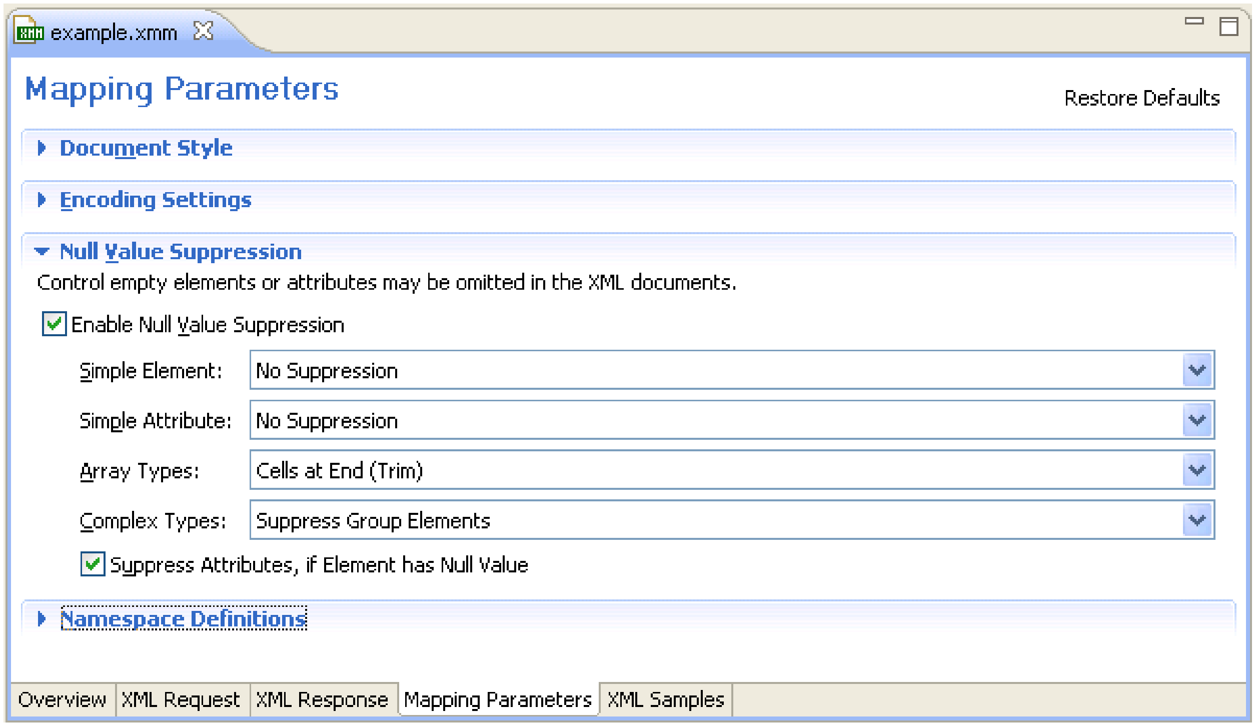
Introduction

The EntireX XML/SOAP Runtime introduced a feature called *null value suppression* to reduce to amount of data transferred between Web client and Web service. Null value suppression (NVS) allows you to hide tags or attributes with a specified value (so-called null value).

The different types of NVS are explained below. The EntireX XML Mapping Editor provides two ways of setting/modifying the NVS type:

- using a general setting on tab **Mapping Parameter**
- on each element or attribute in the definition of request/response on tab **XML Request/ XML Response**

For several data types, a null value is defined by default. See ***Default Definition of Null Value***. To change one of these null values, open the **XML Request/XML Response** tab, select the element/attribute and modify the property of the null value manually. For this modification use the **Properties** view or the **Properties** dialog (open with the context menu).



Mapping Parameters

Null Value Suppression

Control empty elements or attributes may be omitted in the XML documents.

☑ Enable Null Value Suppression

Simple Element: No Suppression

Simple Attribute: No Suppression

Array Types: Cells at End (Trim)

Complex Types: Suppress Group Elements

☑ Suppress Attributes, if Element has Null Value

Namespace Definitions

Default Setting for Null Value Suppression

Data Type	Suppression Mode
Simple Elements	No Suppression
Simple Attributes	No Suppression
Array Types	Cells at end (trim)
Complex Types	Suppress group elements

Tip: The default setting for elements and attributes changed with version 7.3 from "Suppress Element/Attribute" to "No Suppression". The null value suppression for elements and attributes can be set independently. The null value suppression for Complex Types was introduced with version 7.3.

Definition and Examples of Null Value Suppression Mode

If there is a significant difference between pure XML and SOAP for a null value suppression mode, two examples are introduced.

Suppression Mode	Valid for
No Suppression	Elements, attributes
Elements	Elements
Attributes	Attributes
Cells at End (Trim)	Elements inside an array definition
All Empty Cells	Elements inside an array definition
Suppress Group Elements	Elements inside a group definition
Depends On Element	Attributes

No Suppression

The element or attribute is always present in document. The minimum and maximum occurrence of element/attribute must be set to one.

Example 1

XML document	Displayed XML Document

`<prog>`
`   <integer>0</integer>`
`</prog>`
	`<prog>`
	`   <integer>0</integer>`
	`</prog>`

Elements

An element is hidden if

- the element value is equal to null value
- all attributes of the element can be suppressed
- the element has only subelements that can be suppressed

The minimum occurrence of element must be zero, and the maximum occurrence of element must be one.

Example 2

XML document	Displayed XML Document
`<prog> <gr> <integer>0</integer> </gr> </prog>`	`<prog />`

Attributes

An attribute with null value is hidden.

The minimum occurrence of attribute must be zero, and the maximum occurrence of attribute must be one.

Example 3

XML document	Displayed XML Document
`<prog integer="0" name="Henry"/>`	`<prog name="Henry"/>`

Cells at End (Trim)

All elements of array that fulfills the assertion of "Suppress Element/Attribute" are suppressed if its index is higher than the highest index of the non-suppressed element.

The minimum occurrence of elements must be lower than the maximum occurrence, and the maximum occurrence of elements must be the maximum number of elements or unlimited.

XML document	Displayed XML Document
`<prog> <array> <integer>0</integer> <integer>1</integer> <integer>0</integer> <integer>2</integer> <integer>0</integer> <integer>0</integer> </array> </prog>`	`<prog> <array> <integer>0</integer> <integer>1</integer> <integer>0</integer> <integer>2</integer> </array> </prog>`

All Empty Cells

All elements of array that fulfills the assertion of NVS_FIELD are suppressed.

The minimum occurrence of elements must be lower than the maximum of occurrence and maximum occurrence of elements must be maximum number of elements or unlimited.

Example 5a

XML document	Displayed XML Document
`<prog>`	`<prog>`
`   <array>`	`   <array>`
`      <integer>0</integer>`	`      <integer>1</integer>`
`      <integer>1</integer>`	`      <integer>2</integer>`
`      <integer>0</integer>`	`   </array>`
`      <integer>2</integer>`	`</prog>`
`      <integer>0</integer>`	
`      <integer>0</integer>`	
`  </array>`	
`</prog>`	

Example 5b (for SOAP documents the XML/SOAP Runtime creates position attributes)

SOAP document	Displayed SOAP Document
`<prog>`	`<prog>`
`   <array>`	`  <array>`
`      <integer>0</integer>`	`    <integer SOAP-ENC:position="[1]">1</integer>`
`      <integer>1</integer>`	`    <integer SOAP-ENC:position="[3]">2</integer>`
`      <integer>0</integer>`	`  </array>`
`      <integer>2</integer>`	`</prog>`
`      <integer>0</integer>`	
`      <integer>0</integer>`	
`  </array>`	
`</prog>`	

Suppress Group Elements

The suppression mode allows you to suppress group information if - and only if - all elements of the group can be suppressed.

The minimum occurrence of elements must be zero.

Example 6

XML document	Displayed XML Document

`<prog>`
`  <person>`
`    <firstname>Henry</ firstname >`
`    <lastname>Miller</ lastname >`
`    <someInformation>2</ someInformation >`
`  </person>`
`  <person>`
`    <firstname></ firstname >`
`    <lastname></ lastname >`
`    <someInformation>0</ someInformation >`
`  </person>`
`  <person>`
`    <firstname>John</ firstname >`
`    <lastname>Miles</ lastname >`
`    <someInformation>0</ someInformation >`
`  </person>`
`</prog>`
	`<prog>`
	`  <person>`
	`    <firstname>Henry</ firstname >`
	`    <lastname>Miller</ lastname >`
	`    <someInformation>2</ ↩`
	`someInformation >`
	`  </person>`
	`  <person/>`
	`  <person>`
	`    <firstname>John</ firstname >`
	`    <lastname>Miles</ lastname >`
	`    <someInformation>0</ ↩`
	`someInformation >`
	`  </person>`
	`</prog>`

Depends On Element

Attribute of the element is visible if the element does not have the null value.

The minimum occurrence of attribute and associated element must be zero, and the maximum occurrence must be one.

Example 7

XML document	Displayed XML Document

`<prog type="integer">0`
`  <parm type="integer">0</parm>`
`</prog>`
	`<prog />`

Default Definition of Null Value

Data Types	Null Value
String	Empty String
Integer	0
Floating Point	0.0
Numeric	0.0
Time	No default definition
Date	No default definition

Data Types	Null Value
Binary	No default definition
Boolean	False

User-specified Settings

For further settings, use the method `setUserProperty` in `XLRPCService` (EntireX XML/SOAP Runtime) in the Javadoc documentation of the Java ACI.

Map Fault to IDL Parameter

- Introduction
- Example
- Testing the Fault Mapping

Introduction

The XML/SOAP RPC Server maps the values of IDL parameters to XML/SOAP documents and vice versa. If the Web service responds with a fault document, this information is mapped to an error and returned to RPC client normally. With the optional feature **Map Fault to IDL Parameter** you can map values from a normal response and also from a fault document response. This means that no RPC error is returned to the RPC client; instead the fault information is contained in the IDL file. An RPC error is returned to the client only if internal error processing problems occurred within the XML/SOAP RPC Server. This feature is available for SOAP and XML documents.

Example

Note: This example illustrates the feature **Map Fault to IDL Parameter**. Other features mentioned here, such as renaming parameters or assigning a prefix/namespace to a parameter, are described elsewhere.

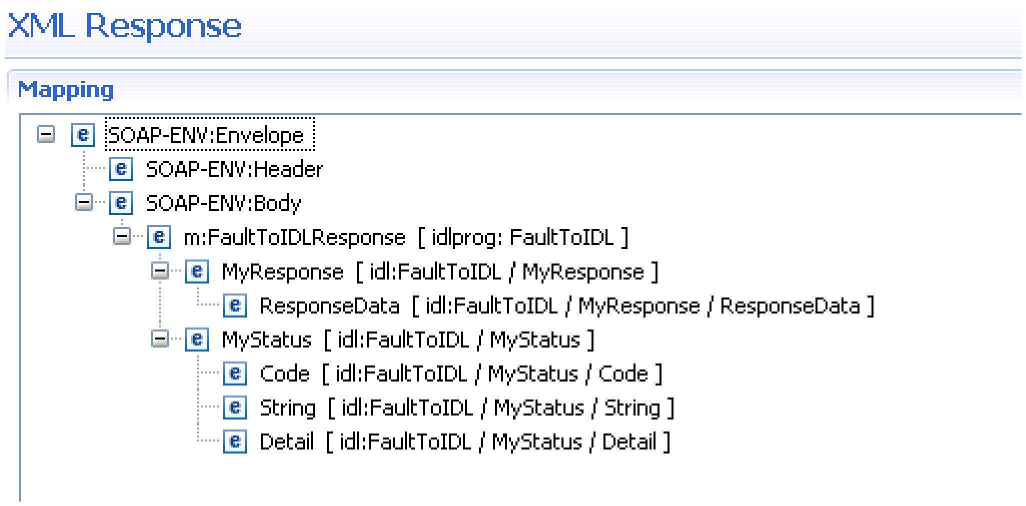
Sample IDL File

```
/* Sample IDL file
library 'Demo' is
  program 'FaultToIDL' is
    define data parameter
      1 MyRequest  In
      2  RequestData     (AV)
      1 MyResponse Out
      2  ResponseData    (AV)
      1 MyStatus   Out   ** parameters for fault information
      2   Code  (AV)
      2   String    (AV)
      2   Detail    (AV)
    end-define
```

IDL-XML Mapping

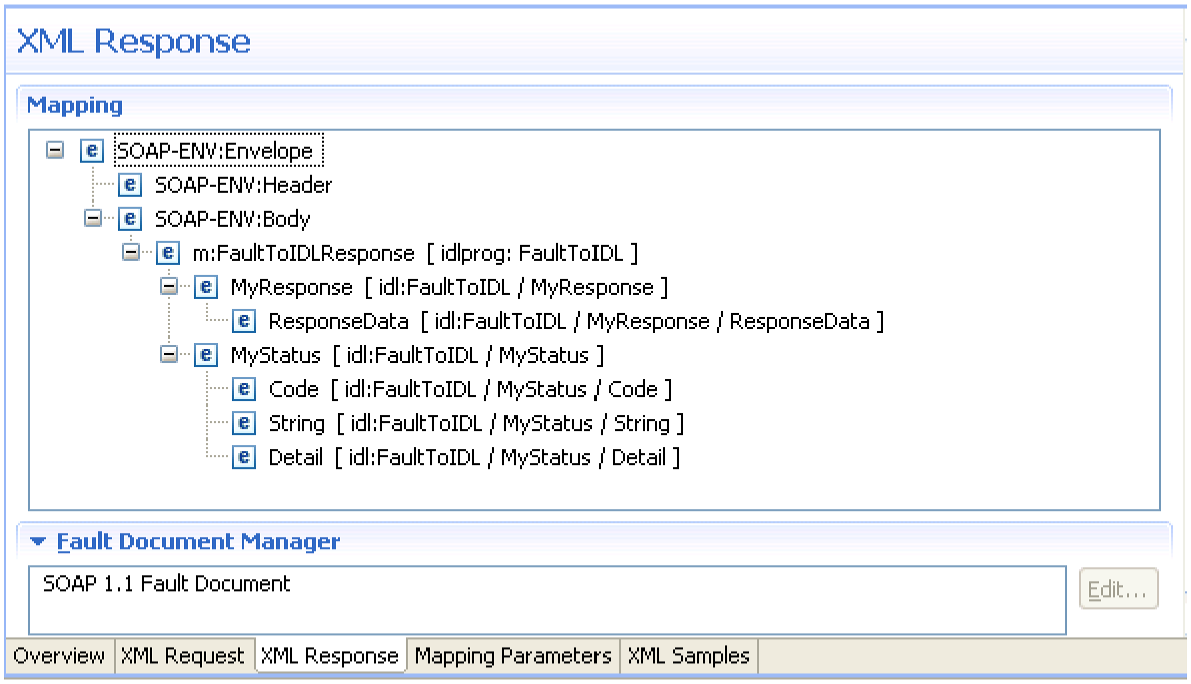
To map fault items to IDL

1 In the XML Mapping Editor, generate a (SOAP) mapping and select the response tab.



```
SOAP-ENV:Envelope
  SOAP-ENV:Header
  SOAP-ENV:Body
    m:FaultToIDLResponse [ idlprog: FaultToIDL ]
      MyResponse [ idl:FaultToIDL / MyResponse ]
        ResponseData [ idl:FaultToIDL / MyResponse / ResponseData ]
      MyStatus [ idl:FaultToIDL / MyStatus ]
        Code [ idl:FaultToIDL / MyStatus / Code ]
        String [ idl:FaultToIDL / MyStatus / String ]
        Detail [ idl:FaultToIDL / MyStatus / Detail ]
```

2 Remove the parameter `MyStatus`, including its children, because the regular response will not contain these parameters and the corresponding IDL parameters will be used for fault information later.



XML Response

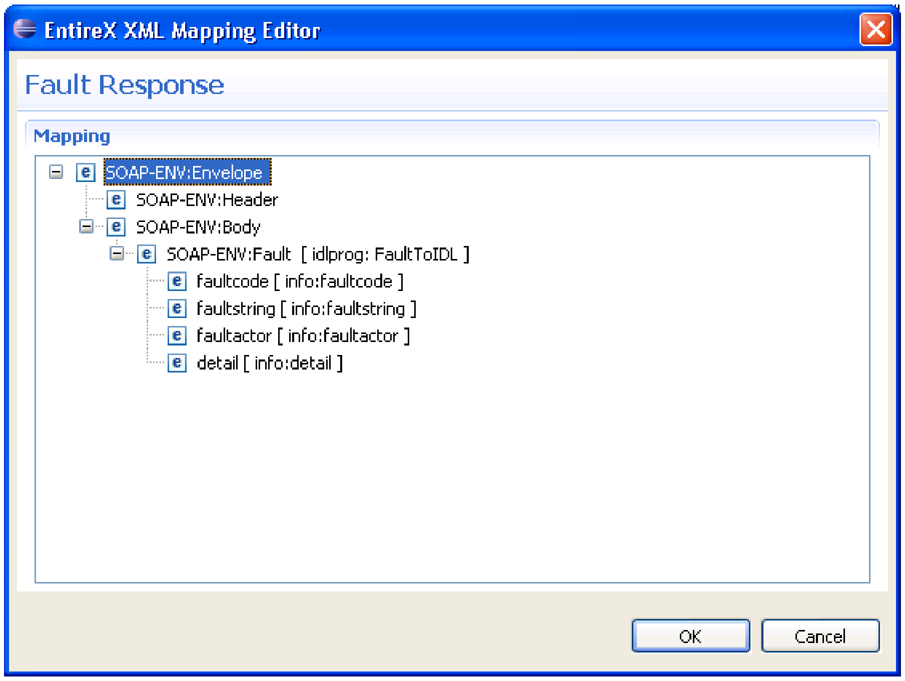
Mapping

```
SOAP-ENV:Envelope
    SOAP-ENV:Header
    SOAP-ENV:Body
        m:FaultToIDLResponse  [ idlprog: FaultToIDL ]
            MyResponse  [ idl:FaultToIDL / MyResponse ]
                ResponseData  [ idl:FaultToIDL / MyResponse / ResponseData ]
            MyStatus  [ idl:FaultToIDL / MyStatus ]
                Code  [ idl:FaultToIDL / MyStatus / Code ]
                String  [ idl:FaultToIDL / MyStatus / String ]
                Detail  [ idl:FaultToIDL / MyStatus / Detail ]
```

Fault Document Manager

SOAP 1.1 Fault Document

3 Open the Fault Document Manager (see bottom of opened tab **Response**) and select the document to open the following wizard:



EntireX XML Mapping Editor

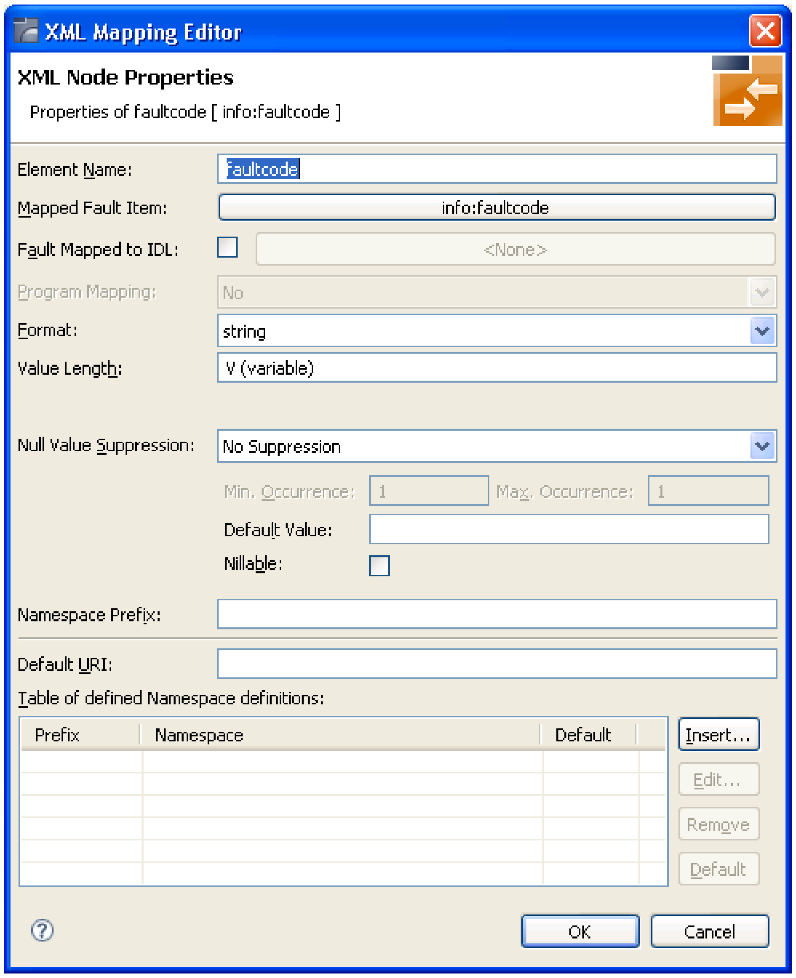
Fault Response

Mapping

```
SOAP-ENV:Envelope
    SOAP-ENV:Header
    SOAP-ENV:Body
        SOAP-ENV:Fault  [ idlprog: FaultToIDL ]
            faultcode [ info:faultcode ]
            faultstring [ info:faultstring ]
            faultactor [ info:faultactor ]
            detail [ info:detail ]
```

4 In the following steps, map "faultcode", "faultstring" and "detail" to IDL parameters. Fault item "faultactor" is not used in this example.

5 Select "faultcode" and open the properties shown on the following screen:


XML Mapping Editor

XML Node Properties

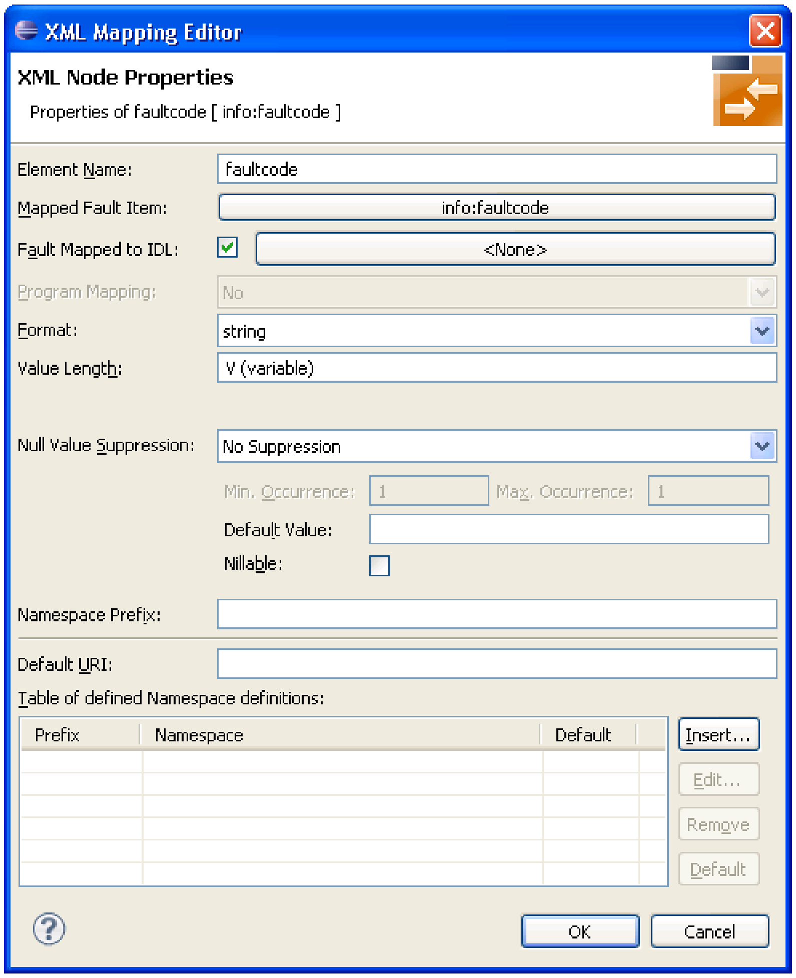
Properties of faultcode [info:faultcode]

Element Name:	faultcode
Mapped Fault Item:	info:faultcode
Fault Mapped to IDL:	☐ <None>
Program Mapping:	No
Format:	string
Value Length:	V (variable)
Null Value Suppression:	No Suppression
Min. Occurrence:	1
Max. Occurrence:	1
Default Value:	
Nillable:	☐
Namespace Prefix:	
Default URI:	

Table of defined Namespace definitions:

Prefix	Namespace	Default

6 Check **Fault Mapped to IDL** to enable the mapping path button. In this example, a mapping path has not yet been entered and the button is labelled "<none>".



XML Node Properties

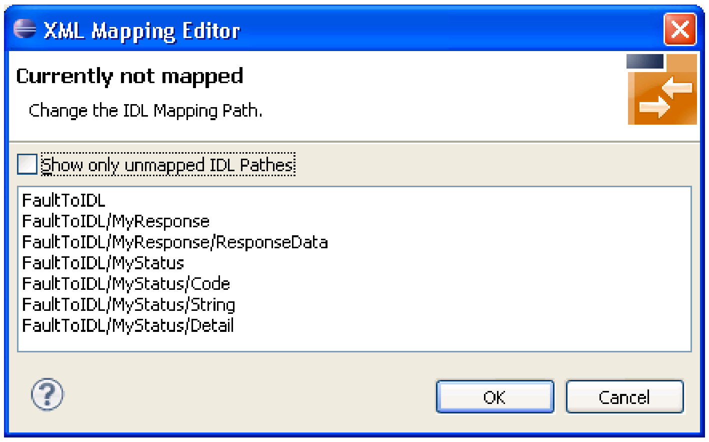
Properties of faultcode [info:faultcode]

Element Name:	faultcode
Mapped Fault Item:	info:faultcode
Fault Mapped to IDL:	✓ <None>
Program Mapping:	No
Format:	string
Value Length:	V (variable)
Null Value Suppression:	No Suppression
Min. Occurrence:	1
Max. Occurrence:	1
Default Value:	
Nillable:	☐
Namespace Prefix:	
Default URI:	

Table of defined Namespace definitions:

Prefix	Namespace	Default

7 Press the mapping path button.



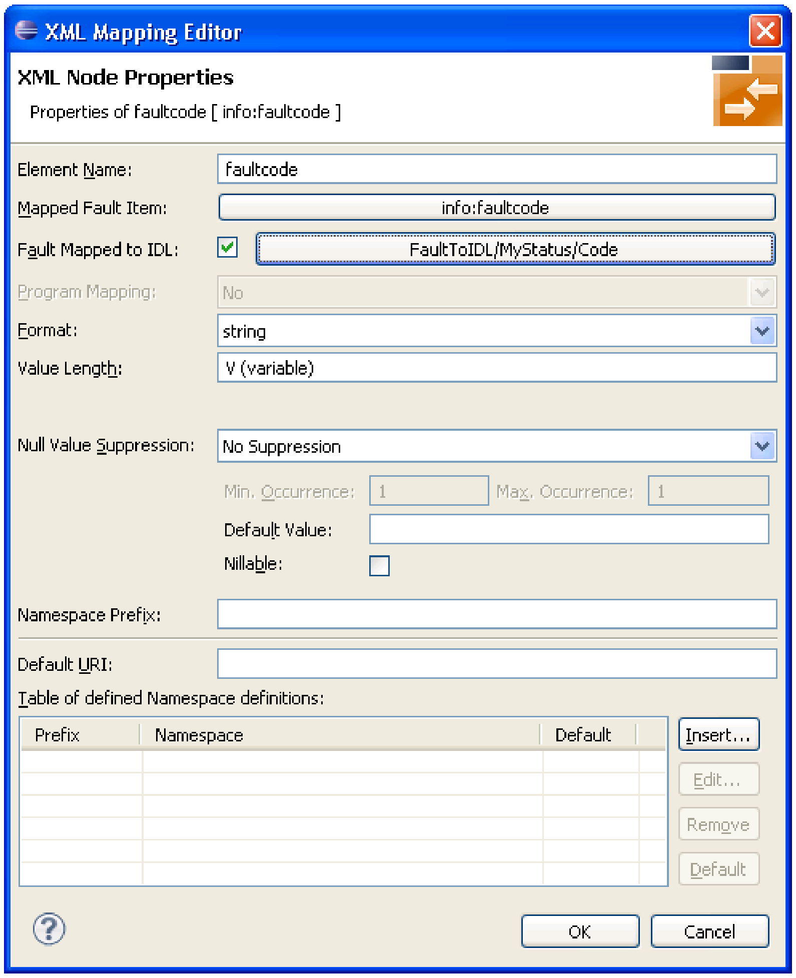
Currently not mapped

Change the IDL Mapping Path.

☐ Show only unmapped IDL Pathes

FaultToIDL
FaultToIDL/MyResponse
FaultToIDL/MyResponse/ResponseData
FaultToIDL/MyStatus
FaultToIDL/MyStatus/Code
FaultToIDL/MyStatus/String
FaultToIDL/MyStatus/Detail

8 Select the path to IDL parameter, for example "FaultToIDL/MyStatus/Code" and choose **OK** to display the following screen:



XML Node Properties

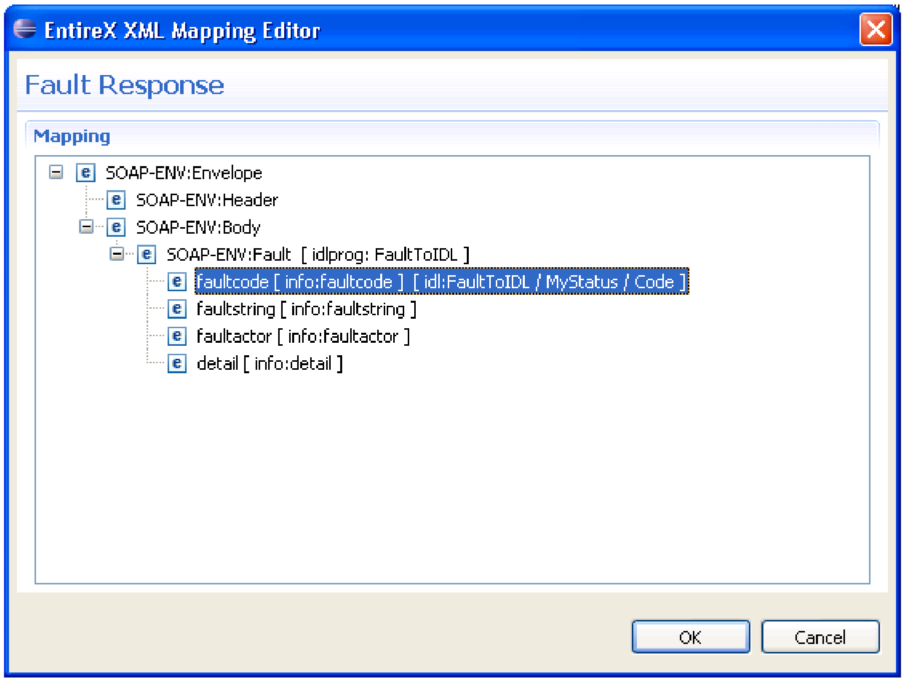
Properties of faultcode [info:faultcode]

Element Name:	faultcode
Mapped Fault Item:	info:faultcode
Fault Mapped to IDL:	✓ FaultToIDL/MyStatus/Code
Program Mapping:	No
Format:	string
Value Length:	V (variable)
Null Value Suppression:	No Suppression
Min. Occurrence:	1
Max. Occurrence:	1
Default Value:	
Nillable:	☐
Namespace Prefix:	
Default URI:	

Table of defined Namespace definitions:

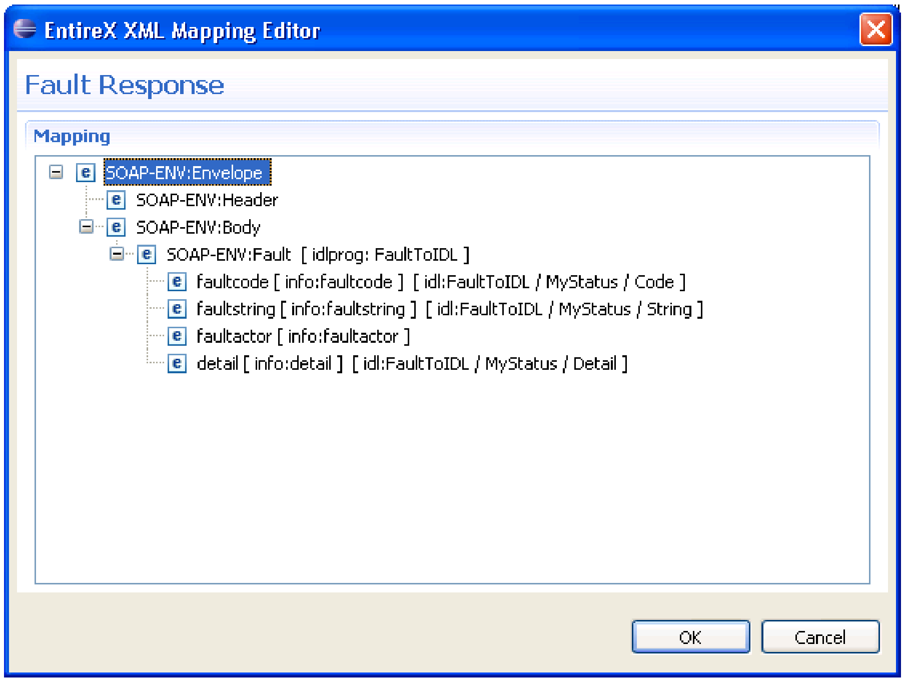
Prefix	Namespace	Default

9 Choose **OK**.



```
SOAP-ENV:Envelope
  SOAP-ENV:Header
  SOAP-ENV:Body
    SOAP-ENV:Fault  [ idlprog: FaultToIDL ]
      faultcode [ info:faultcode ]  [ idl:FaultToIDL / MyStatus / Code ]
      faultstring [ info:faultstring ]
      faultactor [ info:faultactor ]
      detail [ info:detail ]
```

10 Repeat the steps above to select the fault items "faultstring" (path to IDL parameter e.g. "FaultToIDL /MyStatus/String") and "detail" (path to IDL parameter e.g. "FaultToIDL /MyStatus/Detail").



```
SOAP-ENV:Envelope
  SOAP-ENV:Header
  SOAP-ENV:Body
    SOAP-ENV:Fault  [ idlprog: FaultToIDL ]
      faultcode [ info:faultcode ]  [ idl:FaultToIDL / MyStatus / Code ]
      faultstring [ info:faultstring ]  [ idl:FaultToIDL / MyStatus / String ]
      faultactor [ info:faultactor ]
      detail [ info:detail ]  [ idl:FaultToIDL / MyStatus / Detail ]
```

Note: The fault item "info:detail" contains the complete document fragment enclosed by an associated tag (in this example tag <detail>).

11 Choose **OK** to save the IDL-XML mapping.

In subsequent steps you need to

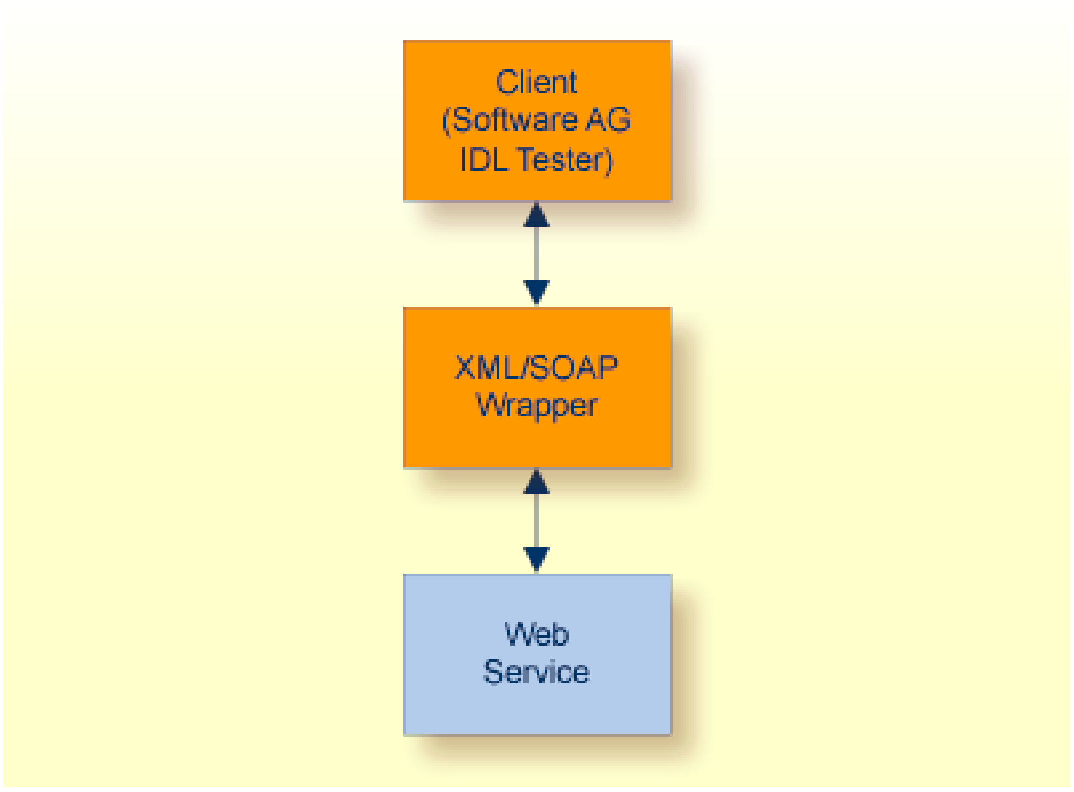
- Set up the XML/SOAP RPC Server with this XMM.
- Set up a new Web service or use an existing one.

Testing the Fault Mapping

As a quick test, implement a Web service that behaves as follows:

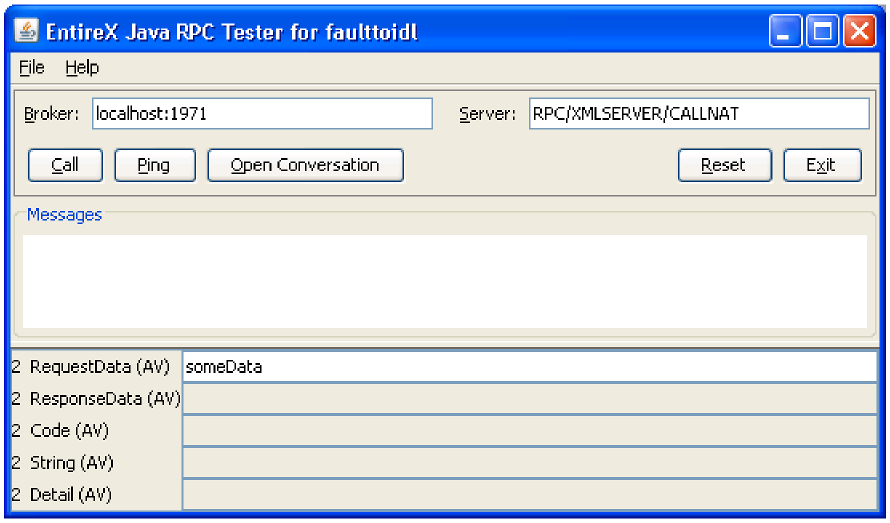
- If the "requestData" field contains any data except "throwException", the field "responseData" in the response document is set to a concatenation of the string "Receiving:" and the value of field "requestData". See Request 2).
- If the "requestData" field contains "throwException", the web service responds with a SOAP fault.

Test Scenario



Request 1 (Expecting Normal Response)

The following screen illustrates a request that expects a normal response:

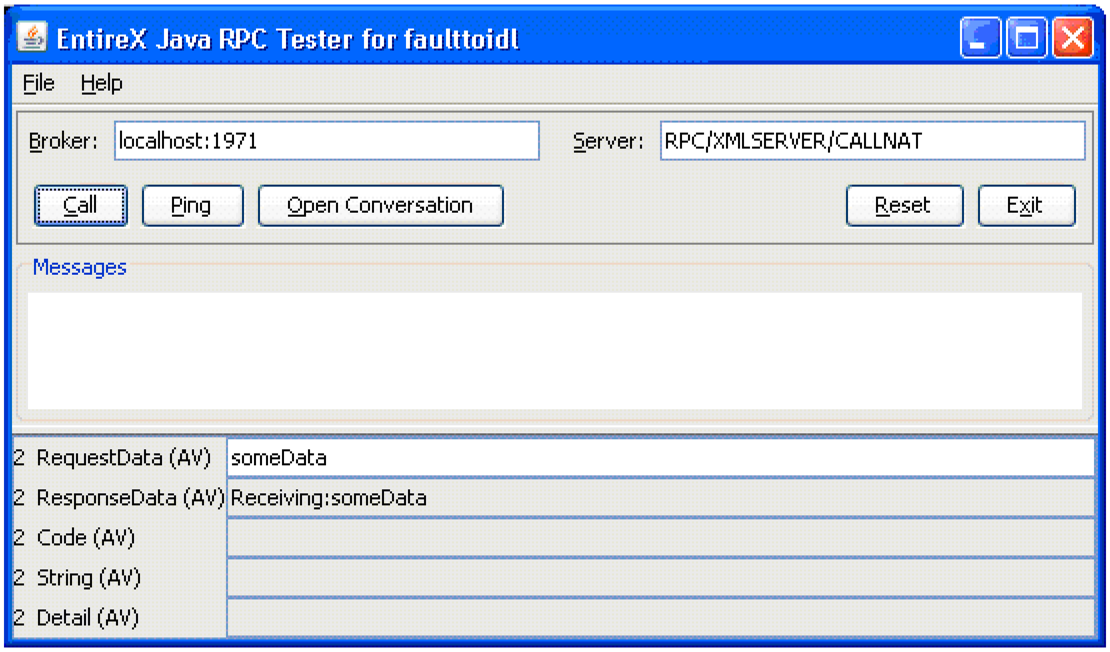


Broker: localhost:1971

Server: RPC/XMLSERVER/CALLNAT

2 RequestData (AV)	someData
2 ResponseData (AV)	
2 Code (AV)	
2 String (AV)	
2 Detail (AV)	

The following response is received (field responseData is filled):

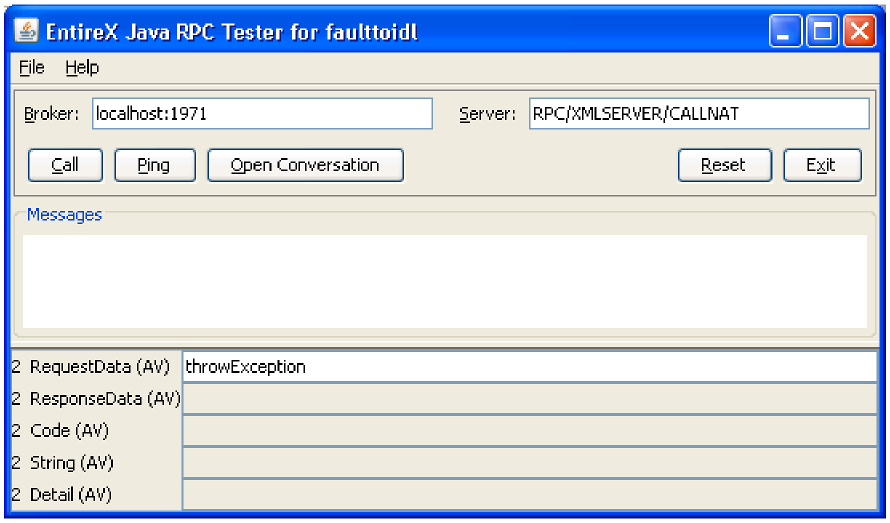


Broker: localhost:1971
Server: RPC/XMLSERVER/CALLNAT

2 RequestData (AV)	someData
2 ResponseData (AV)	Receiving:someData
2 Code (AV)	
2 String (AV)	
2 Detail (AV)	

Request 2 (Expecting Fault Document)

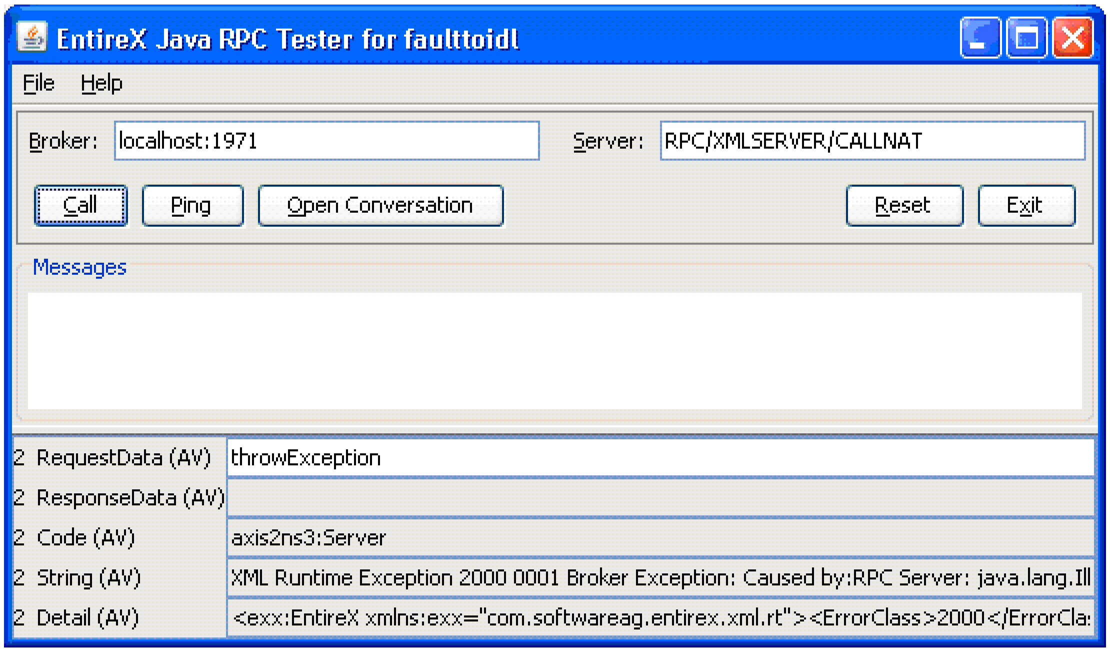
The following screen illustrates a request where a fault document from the Web service is expected:



Broker: localhost:1971
Server: RPC/XMLSERVER/CALLNAT

2 RequestData (AV)	throwException
2 ResponseData (AV)	
2 Code (AV)	
2 String (AV)	
2 Detail (AV)	

The following fault information is provided:



Broker: localhost:1971

Server: RPC/XMLSERVER/CALLNAT

2 RequestData (AV)	throwException
2 ResponseData (AV)	
2 Code (AV)	axis2ns3:Server
2 String (AV)	XML Runtime Exception 2000 0001 Broker Exception: Caused by:RPC Server: java.lang.Ill
2 Detail (AV)	<exx:EntireX xmlns:exx="com.softwareag.entirex.xml.rt"><ErrorClass>2000</ErrorCla:

Whitespace Handling

The XML/SOAP Runtime trims whitespace in values by default. The whitespace handling is also determined by defining attribute `xml:space` (see XML specification) on element(s). The attribute `xml:space` has the higher priority and is inherited from children of the element recursively.

This section covers the following topics:

- Attribute xml:space
- Changing the Default for Whitespace Handling

Attribute xml:space

The attribute `xml:space` can be added in the XML Mapping Editor. Select an element and add new child, select the checkbox for xml:space and choose **OK**. Depending on the application, perform these steps for the request and/or response document definition. The attribute is added with value "preserve". If another value is required, open the properties on the attribute and change the default value.

Note: The XML/SOAP Runtime only supports the value "preserve" for attribute `xml:space`, all other values disable the preserving of whitespace (that is, whitespace is trimmed).

Changing the Default for Whitespace Handling

The steps required to keep whitespace as default depend on the EntireX component:

- **XML/SOAP Listener**
  Add the following line to file axis2.xml in WS-Stack Web application:

  ```
  <parameter name="exx-xml-space">preserve</parameter>
  ```

- **Java API (`XMLRPCService`)**
  Set the user property `entirex.sdk.xml.runtime.xmlspace`:

  ```
  XMLRPCService rpcService = new XMLRPCService(...);
  ...
  rpcService.setUserProperty("entirex.sdk.xml.runtime.xmlspace","preserve");
  ```

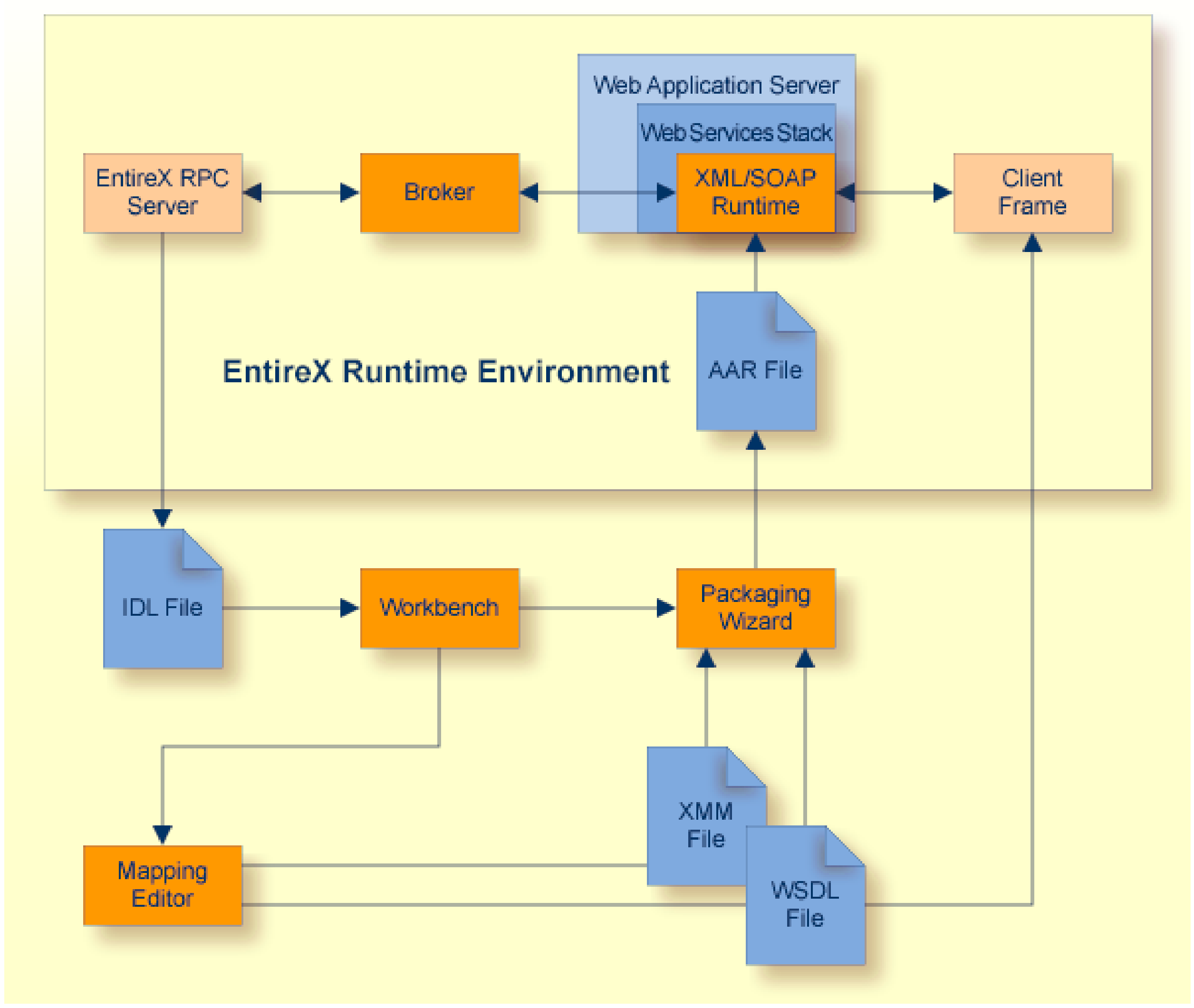
- **XML/SOAP RPC Server**
  Add the following line to properties file of the XML/SOAP RPC Server:

  ```
  entirex.sdk.xml.runtime.xmlspace=preserve
  ```

11 Connect an Existing EntireX RPC Server with an XML-based Client

This chapter describes how to connect an existing RPC server with an XML-based client, using a Web application that contains the EntireX XML/SOAP Runtime as part of Software AG Common Web Services Stack, or using the Java API of EntireX XML/SOAP Runtime.

Connect RPC Server with XML-based Client, using a Web Application



To generate the application, using a default SOAP mapping for the Web Service

- Select the IDL file that was used for creating the EntireX RPC server. From the context menu, choose and **Generate Web Service from Software AG IDL** and follow the instructions given by the wizard.

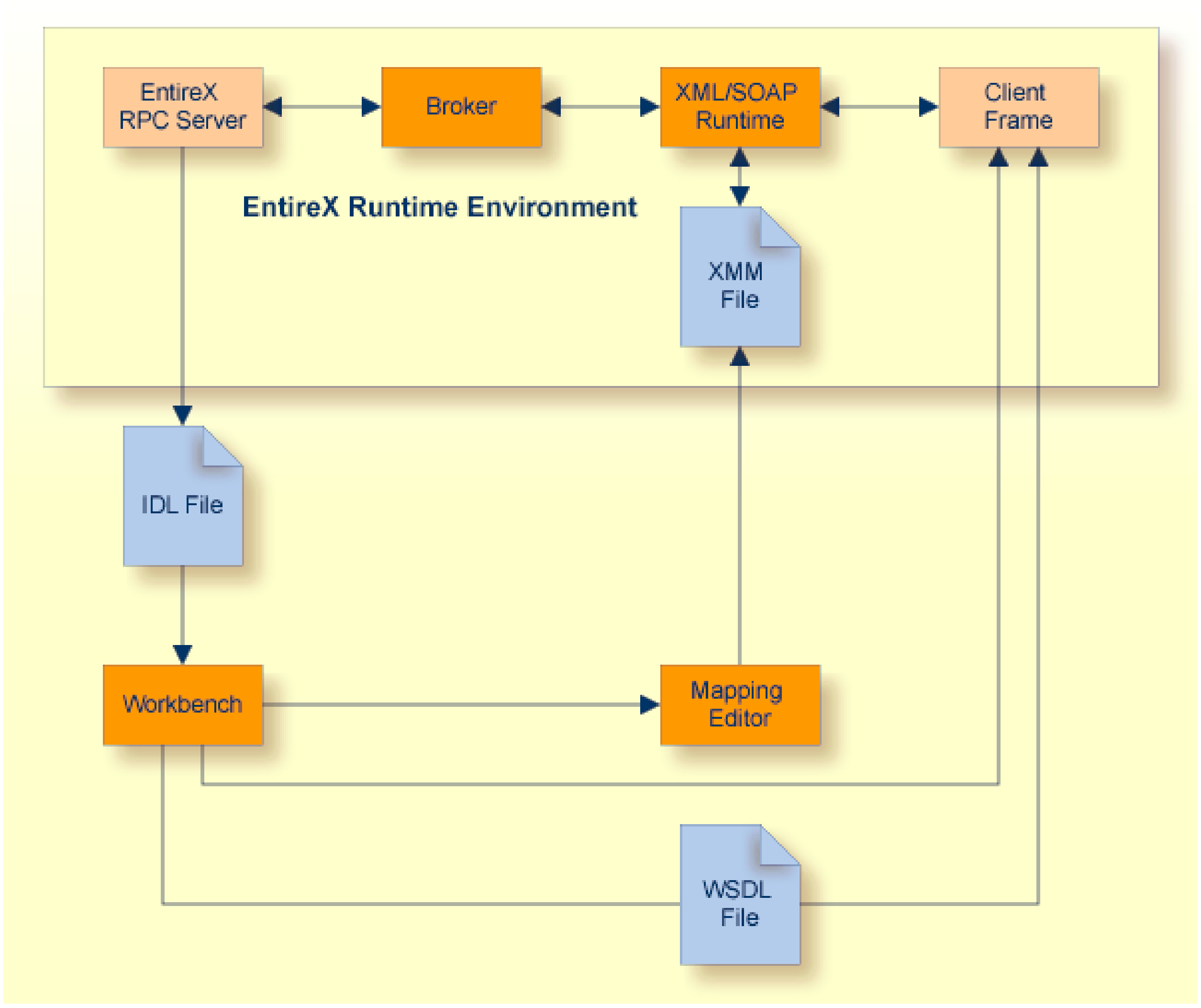
To generate the application, not using an XML Mapping or a non-Default SOAP Mapping

1. Select the IDL file that was used for creating the EntireX RPC server and open it with the EntireX XML Mapping Editor.
2. Select the mapping and press **Generate**. Save the mapping file.
3. Depending on the tools used for the generation of the XML-based client, create an XML Schema or WSDL for the generation of the XML-based client.

   Select the IDL file, and from the context menu choose **Generate Web Service from Software AG IDL**.

   For the generation of the XML Schema file: Select the XMM file and open the context menu. Choose **Generate XML Schema (XSD)...**.

Connect RPC Server with XML-based Client, using the Java API of EntireX XML/SOAP Runtime



To generate the application

1. Select the IDL file that was used for creating the EntireX RPC server and open it with the EntireX XML Mapping Editor.
2. Select the mapping **SOAP** and press **Generate**. Save the mapping file.
3. Depending on the tools used for the generation of the XML-based client, create an XML Schema or WSDL for the generation of the XML-based client.

Select the IDL file, and from the context menu choose **Generate Web Service from Software AG IDL**.

For the generation of the XML Schema file: Select the XMM file and from the context menu choose **Generate XML Schema (XSD)...**.

Note: If this is not possible, write an XML-based client.

4 Create an XML-based client with a suitable tool. Follow the instructions on the screen, or *Generate Web Services from Software AG IDL File* in the Web Services Wrapper documentation.

Running the Application

To run the application

1 Start the EntireX Broker (if required).

2 Configure and start the EntireX RPC server (if not started already).

3 Run the client application.

12 Build an EntireX RPC Server and Use an Existing XML-based Client with It

Generation Process

To generate the application

1. If a WSDL file or an XML Schema file for the XML-based client exists, use the IDL Extractor for XML Schema or the IDL Extractor for WSDL, generating the IDL-XML mapping automatically.

   Otherwise, write a suitable IDL file.
2. Select this generated or written IDL file and open the context menu to generate the desired server skeleton, or, if you are using Natural, write a Natural server.

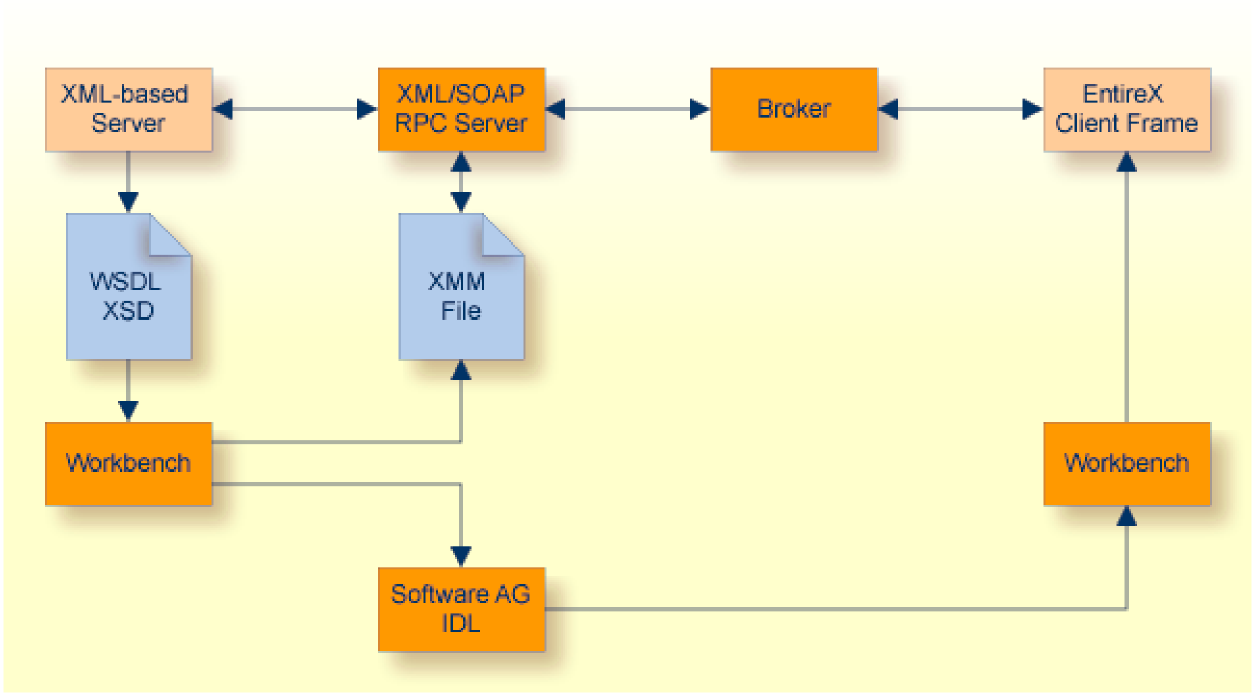
Running the Application

To run the application

1. Start the EntireX Broker (if required).
2. Configure and start the EntireX RPC server or Natural RPC Server.
3. Run the client application.

13 Build an EntireX RPC Client and Use an Existing XML-based Server

Generation Process



To generate the application

1. If a WSDL file or an XML Schema file for the XML-based server exists, use the IDL Extractor for XML Schema or the IDL Extractor for WSDL, generating the IDL-XML-Mapping automatically.

   Otherwise, write a suitable IDL file.
2. Select this generated or written IDL and open the context menu to generate the desired client frame or, if you are using Natural, write a Natural client.

Running the Application

To run the application

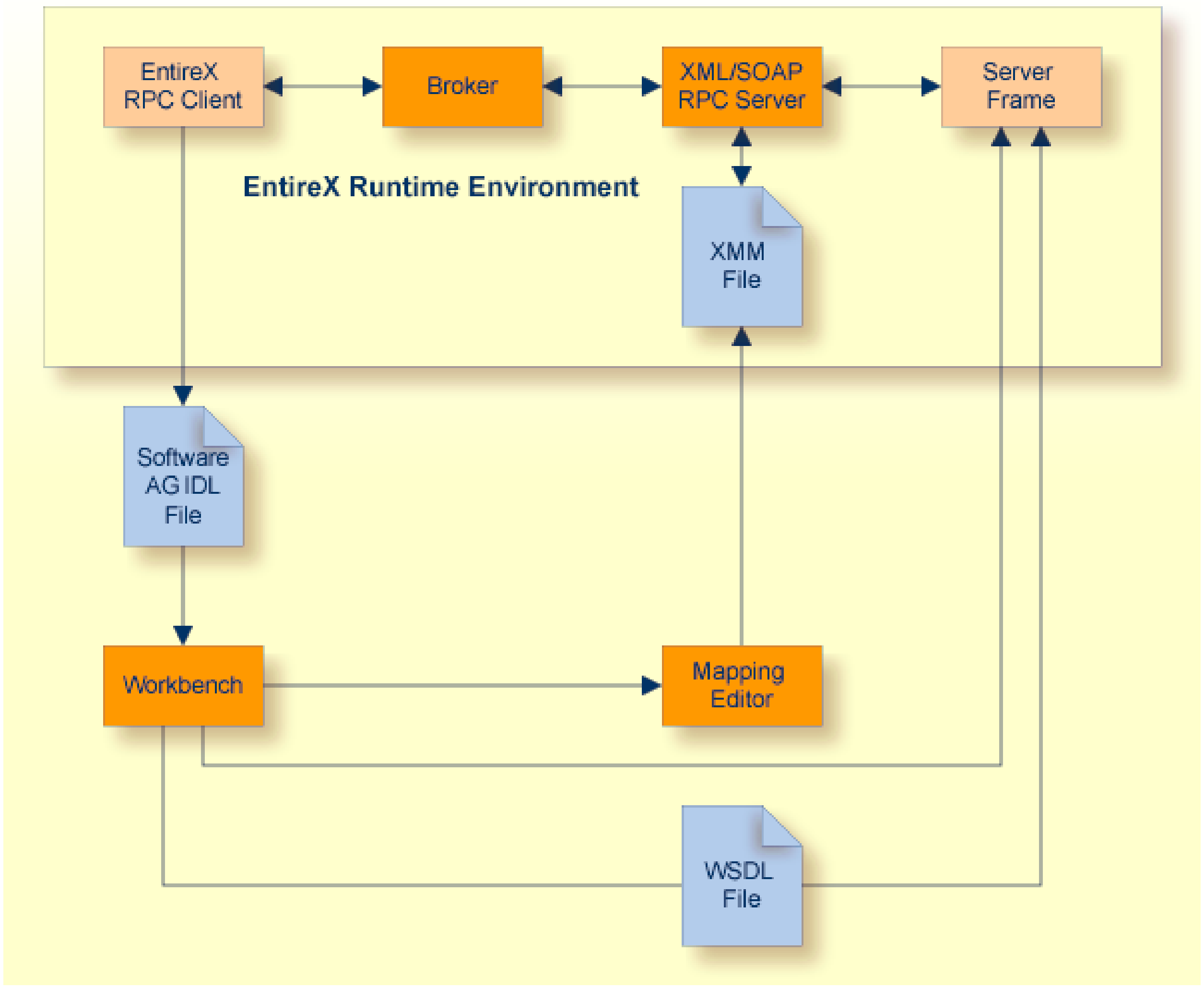
1. Configure the XML/SOAP RPC Server. See *Administering the EntireX XML/SOAP RPC Server* in the UNIX and Windows administration documentation.
2. Start the EntireX Broker.

3 Start the XML/SOAP RPC Server.

4 Run the client application.

14 Connect an Existing EntireX RPC Client to an XML-based Server

Generation Process



To generate the application

1. Select the IDL file and open the context menu to create the EntireX RPC Client.
2. Select the IDL file and open with **EntireX XML Mapping Editor**. Select the mapping "SOAP" and choose **Generate**. Save the mapping.

   Note: If an XML-based server does not exist, proceed as follows:

3. Depending on tools for generation of XML-based server create an XML Schema or WSDL for generation of XML-based server.

Select the IDL file and open the context menu. Select **Generate Web Service from Software AG IDL**.

For generation of XML Schema file: Choose **File > Save XML Schema as**.

Create an XML-based client with a suitable tool and the WSDL / XML Schema file.

For generation of XML Schema file: Select the XMM file and in the context menu choose **Generate XML Schema (XSD)...**.

Note: If this is not possible, write an XML-based server.

Running the Application

To run the application

1. Configure the XML/SOAP RPC Server. See *Administering the EntireX XML/SOAP RPC Server* in the UNIX and Windows administration documentation.
2. Start the EntireX Broker (if required).
3. Start the XML/SOAP RPC Server.
4. Run the client application.

15 Configuring Client and Server Applications

Configuring a Client to Call the EntireX XML/SOAP Runtime (Java API)

Configuration Information

The configuration information (which broker to use) is retrieved from the XMM file by default, or the client sets this information explicitly.

For information on available methods, see `XMLRPCService` (EntireX XML/SOAP Runtime); for available inherited methods, see `RPCService` (EntireX Java ACI).

For configuration of the behavior of XML/SOAP Runtime with `setUserProperty` see `XMLRPCService`.

Environment Settings

EntireX XML/SOAP Runtime requires correct setting for XML Stream Parser allowing access to JAXP methods. The classpath must contain the required JAR files.

The Java properties are:

```
javax.xml.stream.XMLInputFactory
    (default: com.ctc.wstx.stax.WstxInputFactory)
javax.xml.stream.XMLOutputFactory
    (default: com.ctc.wstx.stax.WstxOutputFactory)
```

These must be set for the client application if they differ from the default settings.

Configuring a Client to Call the EntireX XML/SOAP Runtime (XML/SOAP Listener)

Configuration Information

The configuration information (for the EntireX Broker) is retrieved from one of the following sources:

- XMM file (lowest priority) by default
- configuration xml-init.xml inside the Web service description (AAR file) generated by Packaging Wizard
- configuration of EntireX Web service in an external configuration file
- HTTP/payload parameters (highest priority)

See also *Administering the EntireX XML/SOAP Listener* in the UNIX and Windows administration sections.

Environment Settings

EntireX XML/SOAP Runtime requires correct setting for the XML Stream Parser to allow access to JAXP methods. The classpath must contain the required JAR files. The Java properties are:

```
javax.xml.stream.XMLInputFactory
    (default: com.ctc.wstx.stax.WstxInputFactory)
javax.xml.stream.XMLOutputFactory
    (default: com.ctc.wstx.stax.WstxOutputFactory)
```

If the settings differ from the default settings, set them in the start script of the Web server.

Configuring an XML/SOAP RPC Server

The XML/SOAP RPC Server reads configuration data from configuration file (lowest priority), property file and command line (highest priority). You can define location and name of property file and configuration file as command-line parameters. The property file may define the JAXP parameters and the location and name of the configuration file. Default for the properties file is *entirex.xmlrpcserver.properties* and for configuration file *entirex.xmlrpcserver.configuration.xml* located in the working directory. The configuration file (in XML format) contains information about EntireX Broker and a list of target servers, including the mapping file associated with them. If a configuration file or a properties file contains non-encrypted or base64-encoded passwords, the passwords are replaced by the encrypted ones.

See *Administering the EntireX XML/SOAP RPC Server* in the UNIX and Windows administration documentation.

16 Deployment to XML/SOAP RPC Server and Dynamic Configuration of XML/SOAP RPC Server

Introduction

The EntireX Workbench supports two methods of modifying the configuration of XML/SOAP RPC Server:

- deploy an XMM to a specified XML/SOAP RPC Server directly
- interact with XML/SOAP RPC Server to perform configuration changes

The changes are activated immediately without restarting the XML/SOAP RPC Server.

Notes:

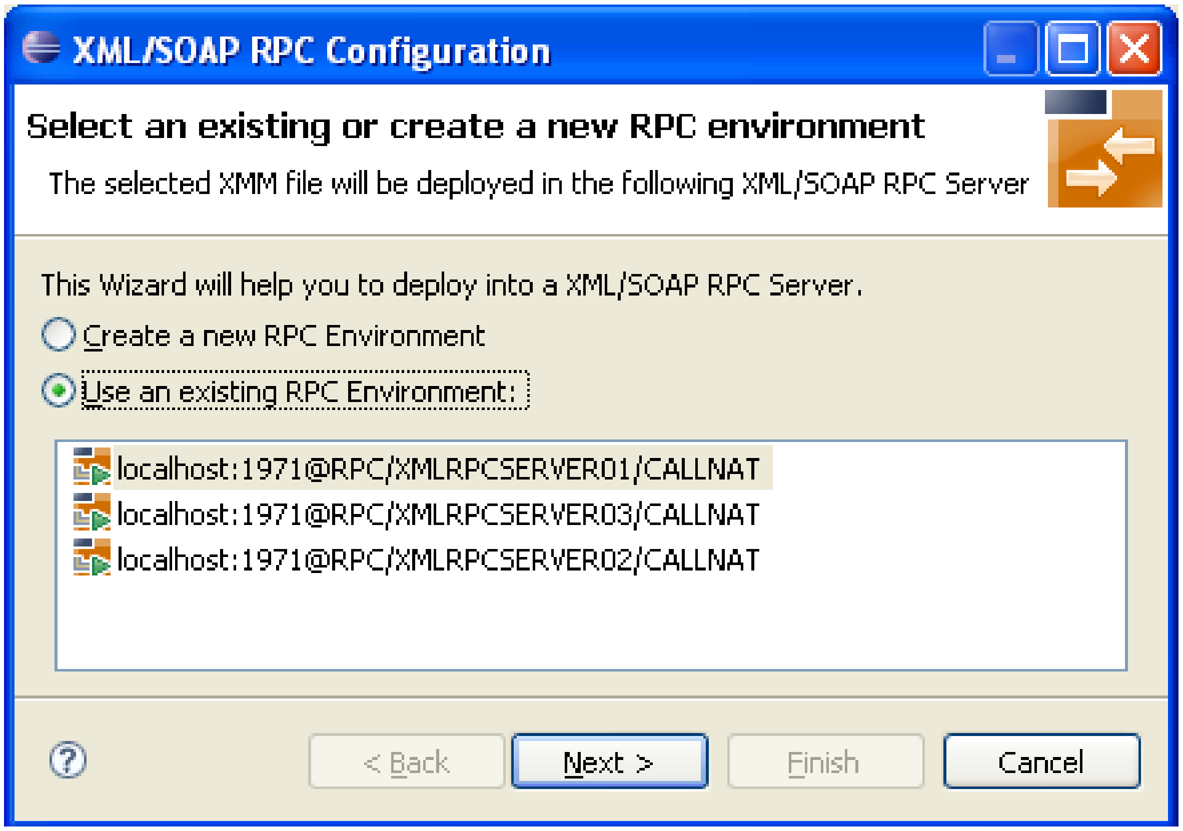
1. Setting the property `entirex.server.allowdevelopment` in the XML/SOAP RPC Server's properties file (default name: *entirex.xmlrpcserver.properties*) to "true" (default) *enables* deployment and dynamic configuration; setting this property to "false" *disables* deployment and dynamic configuration.
2. A deploy call with XMM and WSDL may transport a large amout of data. We therefore recommend you increase the value of broker attribute `MAX-MESSAGE-LENGTH` in the attribute file.

Deploying an XMM File to XML/SOAP RPC Server

To deploy an XMM file to XML/SOAP RPC Server

1 Select an XMM file.

2 From the context menu, choose **Deploy to EntireX XML/SOAP RPC Server ...** to display the following screen:



XML/SOAP RPC Configuration

Select an existing or create a new RPC environment

The selected XMM file will be deployed in the following XML/SOAP RPC Server

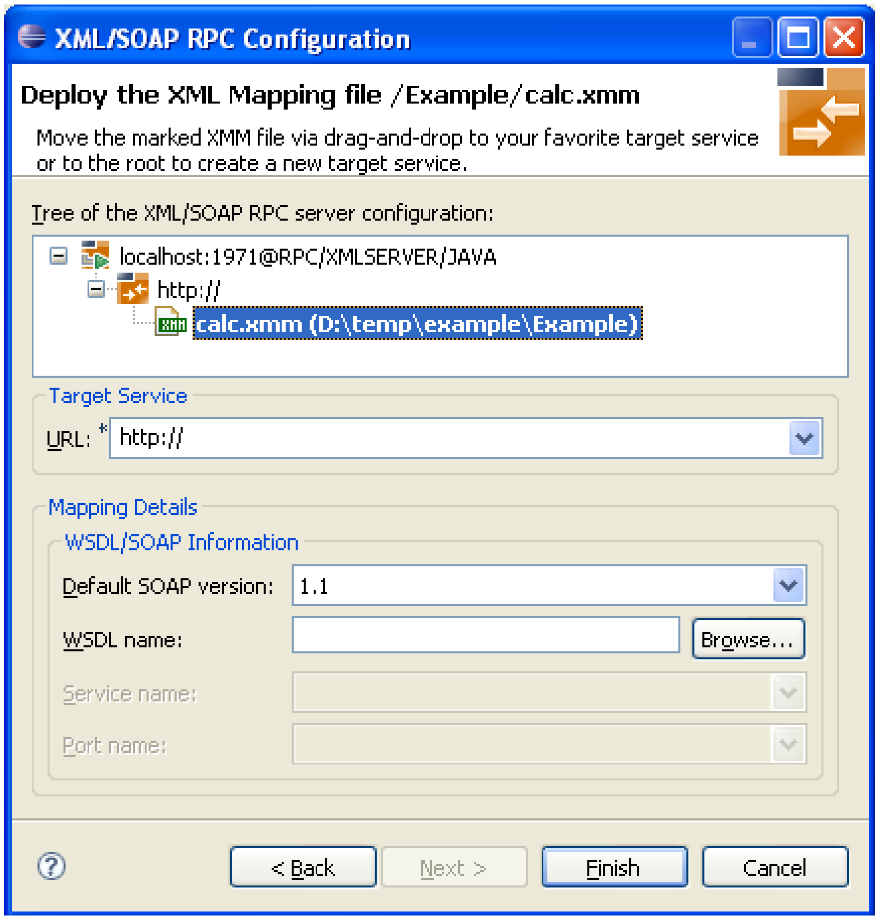
This Wizard will help you to deploy into a XML/SOAP RPC Server.

( ) Create a new RPC Environment

(●) Use an existing RPC Environment:

- localhost:1971@RPC/XMLRPCSERVER01/CALLNAT
- localhost:1971@RPC/XMLRPCSERVER03/CALLNAT
- localhost:1971@RPC/XMLRPCSERVER02/CALLNAT

3 Create a new RPC environment for Broker ID and server address or select an XML/SOAP RPC Server from the list.


XML/SOAP RPC Configuration

Deploy the XML Mapping file /Example/calc.xmm

Move the marked XMM file via drag-and-drop to your favorite target service or to the root to create a new target service.

Tree of the XML/SOAP RPC server configuration:

localhost:1971@RPC/XMLSERVER/JAVA
http://
calc.xmm (D:\temp\example\Example)

Target Service

URL: * http://

Mapping Details

WSDL/SOAP Information

Default SOAP version: 1.1

WSDL name:

Service name:

Port name:

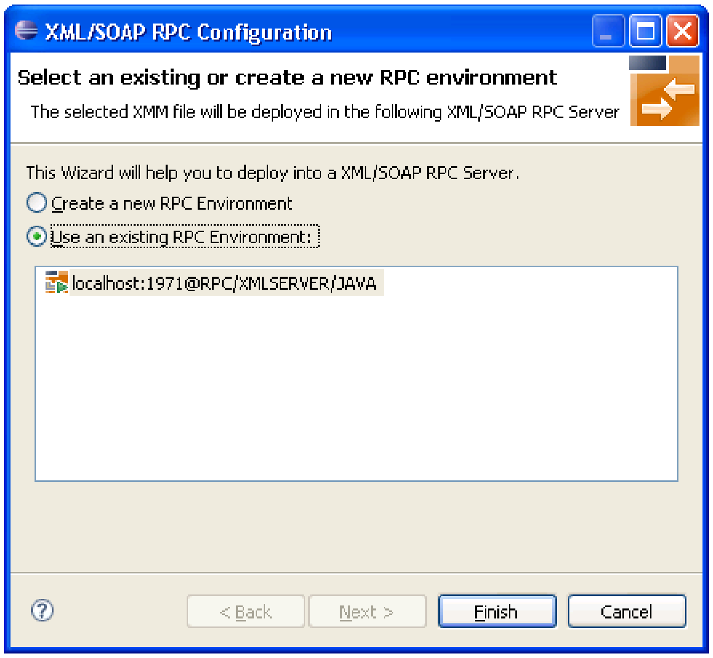
4 Complete the configuration by adding URL and, optionally, the WSDL file. If a WSDL file with same name exists in directory, the field is filled by default. Choose **Finish**.

Undeploying an XMM File to XML/SOAP RPC Server

To undeploy an XMM file to XML/SOAP RPC Server

1 Select an XMM file.

2 From the context menu, choose **Undeploy from EntireX XML/SOAP RPC Server ...** to display the following screen:



XML/SOAP RPC Configuration

Select an existing or create a new RPC environment

The selected XMM file will be deployed in the following XML/SOAP RPC Server

This Wizard will help you to deploy into a XML/SOAP RPC Server.

( ) Create a new RPC Environment

(●) Use an existing RPC Environment:

localhost:1971@RPC/XMLSERVER/JAVA

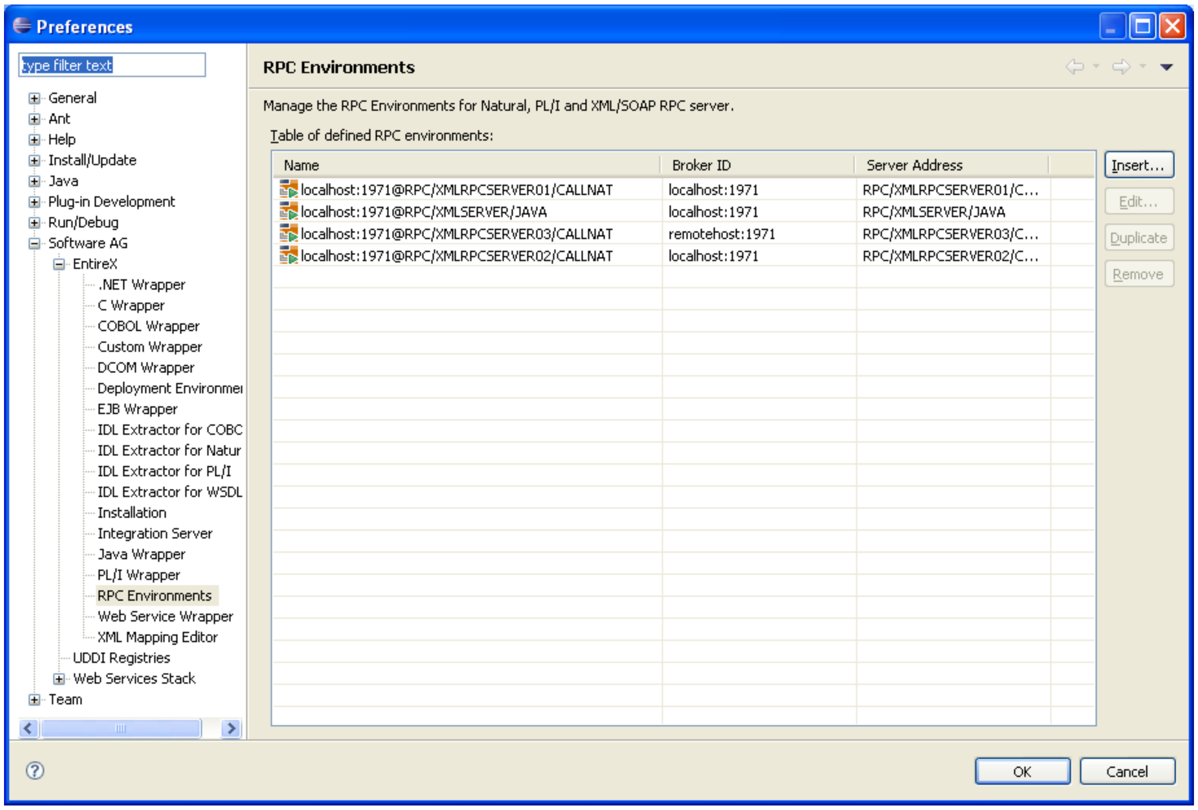
3 Create a new RPC environment for broker ID and server address, or select an RPC server from the list.

4 Choose **Finish**.

Configuring XML/SOAP RPC Server Dynamically

To configure an XML/SOAP RPC Server dynamically

1 From the **Window** menu, choose **Preferences** to disply a screen similar to the one below:



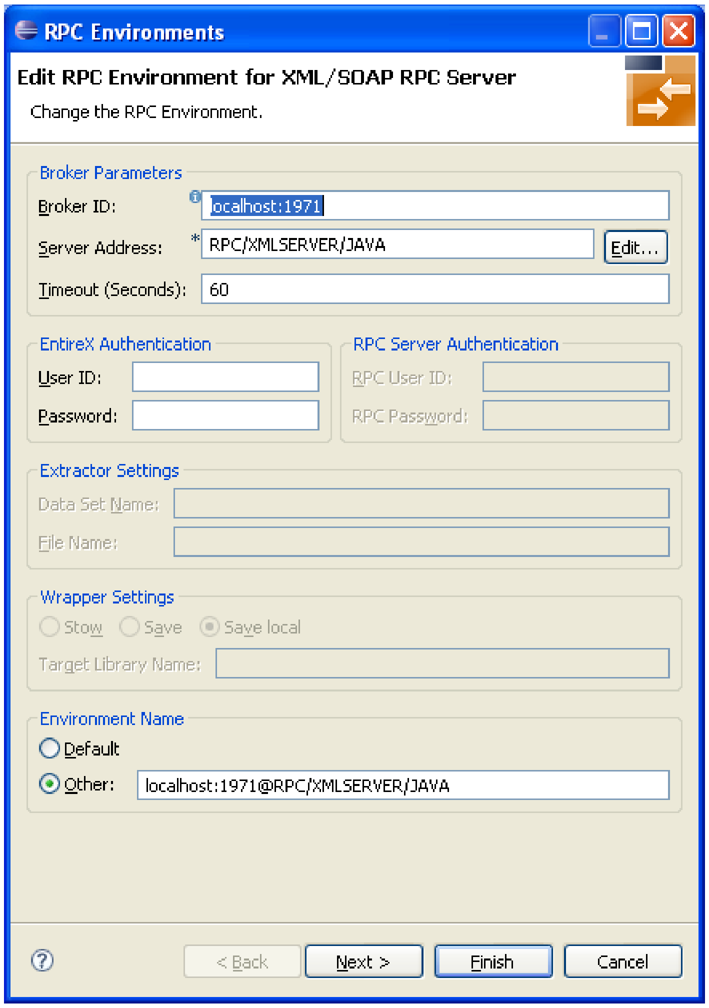
RPC Environments

Manage the RPC Environments for Natural, PL/I and XML/SOAP RPC server.

Table of defined RPC environments:

Name	Broker ID	Server Address
localhost:1971@RPC/XMLRPCSERVER01/CALLNAT	localhost:1971	RPC/XMLRPCSERVER01/C...
localhost:1971@RPC/XMLSERVER/JAVA	localhost:1971	RPC/XMLSERVER/JAVA
localhost:1971@RPC/XMLRPCSERVER03/CALLNAT	remotehost:1971	RPC/XMLRPCSERVER03/C...
localhost:1971@RPC/XMLRPCSERVER02/CALLNAT	localhost:1971	RPC/XMLRPCSERVER02/C...

2 Select an XML/SOAP RPC Server environment or create a new one.



Edit RPC Environment for XML/SOAP RPC Server

Change the RPC Environment.

Broker Parameters

Broker ID:	localhost:1971
Server Address: *	RPC/XMLSERVER/JAVA
Timeout (Seconds):	60

EntireX Authentication

User ID:	
Password:	

RPC Server Authentication

RPC User ID:	
RPC Password:	

Extractor Settings

Data Set Name:	
File Name:	

Wrapper Settings

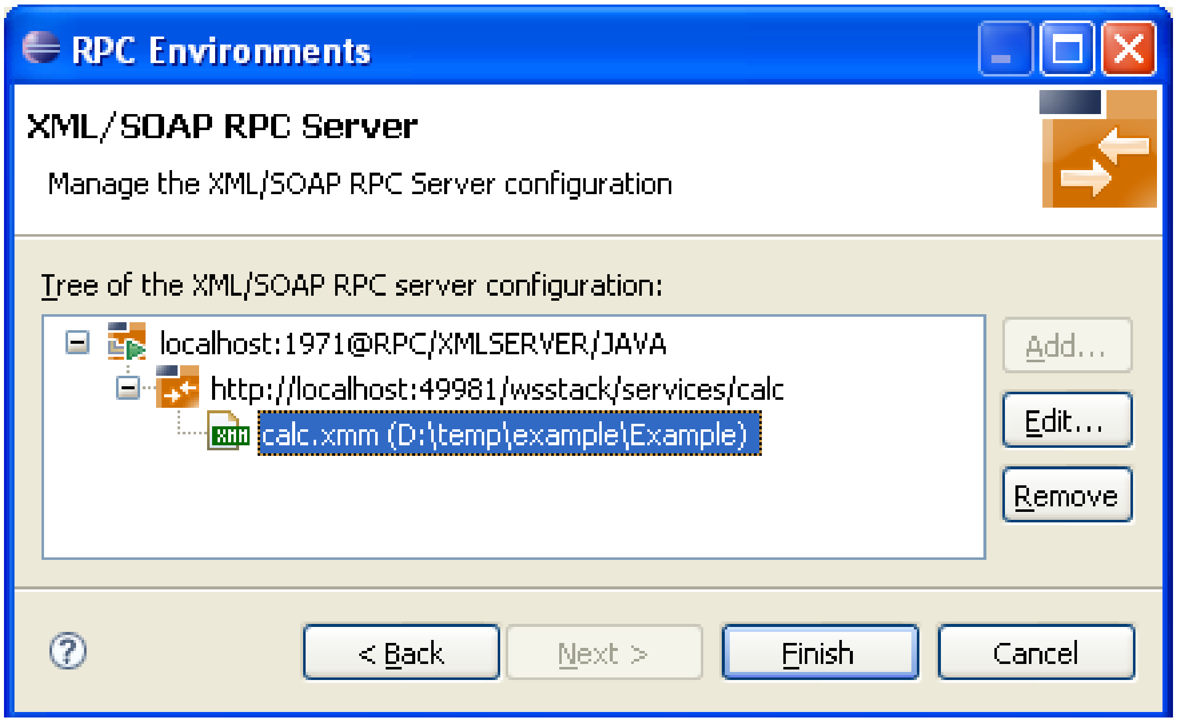
Stow Save [✓ Save local]

Target Library Name:

Environment Name

Default [✓ Other:] localhost:1971@RPC/XMLSERVER/JAVA

3 Choose **Next** to view or manage the configuration of Web services.


RPC Environments
XML/SOAP RPC Server
Manage the XML/SOAP RPC Server configuration
Tree of the XML/SOAP RPC server configuration:
localhost:1971@RPC/XMLSERVER/JAVA
http://localhost:49981/wsstack/services/calc
calc.xmm (D:\temp\example\Example)
Add...
Edit...
Remove
< Back
Next >
Finish
Cancel

4 The second wizard page allows you to add, modify and remove service addresses and their configuration as well as add, modify and remove XMM file in the selected configuration. The graphical user interface supports drag-and-drop operations.

17 Examples

Example 1: Existing Natural Client that Connects to a Web Service

Natural Program

```
* CALC     : CLIENT                                                  *
* ---------------------------------------------------------------- *
DEFINE DATA
LOCAL
  01 #OPERATOR                (A1)
  01 #OPERAND1                (I4)
  01 #OPERAND2                (I4)
  01 #RESULT                  (I4)
*
  01 #MSG                     (A60)
END-DEFINE
* ---------------------------------------------------------------- *
ASSIGN #OPERAND1 = 12345
ASSIGN #OPERATOR = '+'
ASSIGN #OPERAND2 = 67890
ASSIGN #RESULT   = 0
*
CALLNAT 'CALC' #OPERATOR  (AD=O)
               #OPERAND1  (AD=O)
               #OPERAND2  (AD=O)
               #RESULT    (AD=A)
*
COMPRESS #OPERAND1 #OPERATOR #OPERAND2 '=' #RESULT
            INTO #MSG LEAVING NO SPACE
*
PRINT #MSG
* ---------------------------------------------------------------- *
END
```

- Use the IDL Extractor for Natural to get the IDL file. In the EntireX perspective, use **New > IDL Extractor for Natural** . In other perspectives, use **New > Other... > Software AG > IDL Extractor for Natural**.

Software AG IDL

```
Library 'Example' Is
 Program 'Calc' Is
  Define Data Parameter
   1 Operator        (A1) In
   1 Operand_1       (I4) In
   1 Operand_2       (I4) In
   1 Function_Result (I4) Out
  End-Define
```

- Select the IDL file and open the context menu.
- Choose **Generate Web Service from Software AG IDL**, and the XMM and WSDL file will be generated. See *EntireX Web Services Wrapper*.

Example.wsdl

```
<?xml version="1.0" encoding="utf-8"?>
<definitions name="example" ↩
targetNamespace="http://namespace.softwareag.com/entirex/xml/mapping"
   xmlns="http://schemas.xmlsoap.org/wsdl/" ↩
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" ↩
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" ↩
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:tns="http://namespace.softwareag.com/entirex/xml/mapping" ↩
xmlns:sn0="urn:com-softwareag-entirex-rpc:EXAMPLE">
   <types>
      <xsd:schema targetNamespace="urn:com-softwareag-entirex-rpc:EXAMPLE">
         <xsd:element name="CALC">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="Operator" type="xsd:string"/>
                  <xsd:element name="Operand_1" type="xsd:int"/>
                  <xsd:element name="Operand_2" type="xsd:int"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="CALCResponse">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="Function_Result" type="xsd:int"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:schema>
   </types>
   <message name="CALCSoapIn">
      <part name="parameters" element="sn0:CALC"/>
   </message>
   <message name="CALCSoapOut">
      <part name="parameters" element="sn0:CALCResponse"/>
   </message>
   <portType name="EXAMPLEPort">
      <operation name="CALC">
         <input message="tns:CALCSoapIn"/>
         <output message="tns:CALCSoapOut"/>
      </operation>
   </portType>
   <binding name="EXAMPLESOAP11Binding" type="tns:EXAMPLEPort">
      <soap:binding style="document" ↩
transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="CALC">
         <soap:operation soapAction="CALC"/>
         <input>
            <soap:body use="literal"/>
         </input>
         <output>
            <soap:body use="literal"/>
         </output>
      </operation>
   </binding>
   <binding name="EXAMPLESOAP12Binding" type="tns:EXAMPLEPort">
      <soap12:binding style="document" ↩
transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="CALC">
         <soap12:operation soapAction="CALC"/>
         <input>
            <soap12:body use="literal"/>
         </input>
         <output>
            <soap12:body use="literal"/>
         </output>
      </operation>
   </binding>
   <service name="example">
      <port name="EXAMPLESOAP11Port" binding="tns:EXAMPLESOAP11Binding">
         <soap:address location="http://localhost:10010/wsstack/example"/>
      </port>
      <port name="EXAMPLESOAP12Port" binding="tns:EXAMPLESOAP12Binding">
         <soap12:address location="http://localhost:10010/wsstack/example"/>
      </port>
   </service>
</definitions>
```

- Create a service skeleton with Apache Axis.

```
java -classpath org.apache.axis.wsdl.WSDL2Java --server-side --skeletonDeploy true ↩
Example.wsdl
```

- And write an implementation of this service.

```
/**
 * Service.java
 *
 * Implementation of ExamplePort
 * generated by the Apache Axis WSDL2Java emitter.
 */

package com.softwareag.namespace;

public class Service implements ExamplePort
{
    public int calc(java.lang.String operator_, int operand_1, int operand_2)
    throws java.rmi.RemoteException
    {
    int result = 0;
        if (operator_.equals("+"))
    {
      result = operand_1 + operand_2;
    }
        else if (operator_.equals("-"))
    {
      result = operand_1 - operand_2;
    }
        else if (operator_.equals("*"))
    {
      result = operand_1 * operand_2;
    }
        else if (operator_.equals("/"))
    {
      result = operand_1 / operand_2;
    }
    return result;
    }
}
```

- Build and deploy the service (see Apache Axis documentation).
- Configure the XML/SOAP RPC Server

entirex.xmlrpcserver.properties

```
# jaxp parameters
# if jaxp properties are not set in system properties
#
# xmlruntime configuration file
entirex.sdk.xml.runtime.configurationfile=.entirex.xmlrpcserver.configuration.xml
```

entirex.xmlrpcserver.configuration.xml

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<EntireX
    xmlns="http://namespaces.softwareag.com/entirex/xml/runtime/configuration" ↩
version="7.1.1">
    <XmlRuntime Version="1">
         <BrokerInfo>
            <BrokerId>localhost:1971</BrokerId>
            <ServerAddress>RPC/SRV1/CALLNAT</ServerAddress>
        </BrokerInfo>
     <TargetServer name="http://localhost:8080/axis/services/ExamplePort">
            <XmmList>
                <!-the name of XMM file-->
                <Xmm name="./Calc.xmm" />
            </XmmList>
        </TargetServer>
    </XmlRuntime>
</EntireX>
```

- start the XML/SOAP RPC Server:

  ```
  java com.softwareag.entirex.xml.rt.XMLRPCServer
  ```

Example 2: Publish an EntireX RPC Server for Web Clients

Software AG IDL File

```
library 'UserList' is
    program 'Add' is
        define data parameter
        1 Name (AV) In
        end-define

   program 'Retrieve' is
        define data parameter
        1 Name (AV/V) Out
        end-define
```

To publish the EntireX RPC/SOAP server

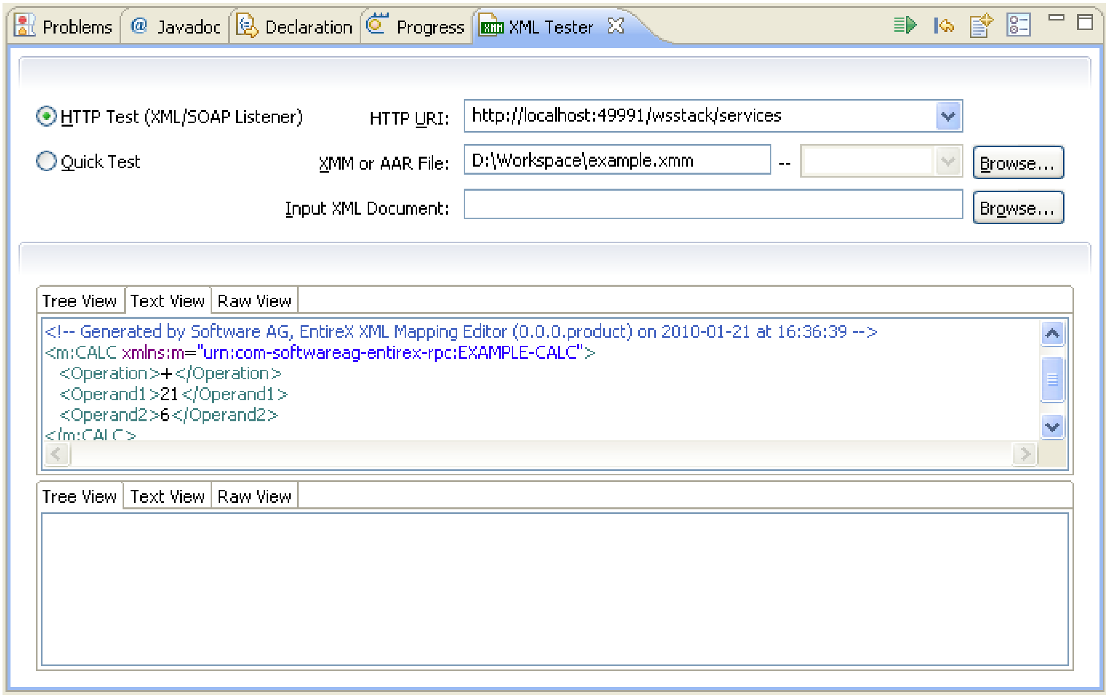
1 Create a new IDL file *User List* (Using: **New > Software AG IDL file** or **New > Others ... > Software AG > Software AG IDL file**).

2 Select the IDL file and generate the RPC server from the context menu.

3 Select the IDL file, and from the context menu choose **Generate Web Service from Software AG IDL**. Generate and deploy the AAR file with the Packaging Wizard.

4 Select the tab **XML Samples** in the EntireX XML Mapping Editor and save one or all sample documents.

5 Select the request document in the tree and open the XML Tester with the context menu.



HTTP Test (XML/SOAP Listener) [✓] HTTP URI: http://localhost:49991/wsstack/services

Quick Test XMM or AAR File: D:\Workspace\example.xmm --

Input XML Document:

```
<!-- Generated by Software AG, EntireX XML Mapping Editor (0.0.0.product) on 2010-01-21 at 16:36:39 -->
<m:CALC xmlns:m="urn:com-softwareag-entirex-rpc:EXAMPLE-CALC">
  <Operation>+</Operation>
  <Operand1>21</Operand1>
  <Operand2>6</Operand2>
</m:CALC>
```

6 Change the URL to the required address, for example: *http://localhost:10010/wsstack/services/example*. Choose **Send**. The response document will be displayed in the lower portion of the screen.

18 Frequently Asked Questions (FAQ) and Troubleshooting

XML/SOAP Listener

- If you receive message `2000 0007: Incoming XML document is invalid`, there could be a difference between the mapping and the document, for example an element-preferred mapping (XMM file) does not match the SOAP document.
- Deploying/calling an EntireX Web service fails with
  `com.softwareag.wsstack.ui.deployment.DeploymentException:`
  `com.softwareag.entirex.xml.rt.WSSServiceLifeCycle`.

  For the XML/SOAP Listener you need to set up a Web server and install Software AG Common Web Services Stack to this Web server. See *Deploying Web Services Stack Runtime* under *Writing Web Services Applications* in the Web Services Wrapper documentation.

  After successful installation: If *entirex.jar* was not added to *wsstack.war*, copy or upload `inst_dir`*/EntireX/classes/entirex.jar* to the *WEB-INF/lib* folder in the Web Services Stack application folder.

XML/SOAP RPC Server

- Server does not start
  - Check classpath
  - If a file is not found:
    - Check location of property file, configuration file or XMM(s)
- XML/SOAP RPC Server starts but receives an HTTP error for several reasons. Check the received information fault document. If there is no information fault document, start XML/SOAP RPC Server again with trace (level = ADVANCED) and analyze the trace (look for the fault document).
- XML/SOAP RPC Server gets the following error message: "`Caused by: java.lang.ClassCastException: org.apache.xerces.jaxp.SAXParserFactoryImpl cannot be cast to javax.xml.stream.XMLInputFactory`". To correct the error, modify the properties file by redefining the parameter for the XML stream parser as follows:

  ```
  entirex.sdk.xml.runtime.xmlparserfactory=com.ctc.wstx.stax.WstxInputFactory
  ```

In any case, check the following:

- Configuration file: TargetServer attribute “name” is not fully qualified
  - Name may require a domain specification

  - Name only contains target without service specification or/and with wrong port (non-default port)
- Mapping
  - Mapping does not match service requirements
  - SOAPAction value is wrong or undefined
- Connection cannot be established or connection failed
  - Wrong HTTP setting

    Change / set Java properties `http.proxyHost`, `http.proxyPort`, `httpsproxyHost`, `https.proxyPort`
  - Application error (service side)

XML/SOAP RPC Server in the Software AG Runtime

Use file *<installation home>/profiles/CTP/configuration/logging/log_config.xml* to configure the logging settings. The log file *sag-osgi.log* or *wrapper.log* is written to directory *<installation home>/profiles/CTP/logs*.

To analyse the log, search for lines containing "EXX" and check the status/error:

EXX: Configuration Error: Reading configuration file fails.

Explanation	The entirex.servers.properties cannot be loaded.
Action	The file must be located in directory *<installation home>\EntireX\etc\exx\workspace* and be accessible.

EXX: Reading configuration fails: File does not exist.

Explanation	The file entirex.servers.properties cannot be found.
Action	Validate the installation.

EXX: XML/SOAP RPC Server does not start. Reason: {0}".

Explanation	The server cannot start for reason {0}.
Action	1. Validate the properties and configuration file settings (see also log). 2. Validate if the properties and configuration are correct. 3. Validate if the paths of XMM files in configuration file are correct and the files are accessible.

19 Reference - HTTP and Java Interface

Client Using the Java Interface

See also the delivered example `XMMInvoker` and `XLRPCService` (EntireX XML/SOAP Runtime) in the Javadoc documentation of the Java ACI.

Step 1: Writing the Client Program

To write the client program

1 Initialize the `Broker` Object

```
Broker broker = new Broker(brokerID, userID);
// if a logon should be issued
 broker.logon(password);
```

2 Initialize the `XMLRPCService` Object

```
XMLRPCService service;
service = new XMLRPCService(xmmfilename);//constructor with XMM
```

or

```
XMLRPCService service;
service = new XMLRPCService(broker, serverName, xmmfilename);
```

3 Initialize the `Conversation` Object

For one `XMLRPCService`, only one conversation may be used at a time.

```
  private Conversation conv;
  conv = new Conversation(service);
```

4 Assign the incoming XML document:

```
String xmlDocument = "<?xml version="1.0" encoding="iso8859-1"?>" +
"<CALC Operation=\"-\" Operand_1=\"1000000611\"
Operand_2=\"1000000288\"></CALC>";
```

5 Invoke the Service

Non-conversational

```
try
{
  String result = service.invokeXML(xmlDocument);
}
catch (XMLException e)
{
   // Error handling ...
}
catch (BrokerException e)
{
  // Error handling ...
}
catch (Exception e)
{
  // Error handling ...
}
```

Conversational

```
try
{
  service.setConversation(conv);
  String result = service.invokeXML(xmlFromFile(filename));
  service.closeConversationCommit();
}
catch (XMLException e)
{
   // Error handling ...
}
catch (BrokerException e)
{
  // Error handling ...
}
catch (Exception e)
{
  // Error handling ...
}
```

The string result contains the returned document. It should be:

```
<CALC Function_result="899" />
```

Step 2: Running the Client Program

The definition of the Java classpath must include

- the generated XMM file, see *Generating an XMM File*
- the *entirex.jar* file (delivered in the directory *<EntireX Home>/classes*)
- the files for an XMLStreamParser, for example the delivered *wstx-asl.jar* and *stax-api.jar* (in the directory *<EntireX Home>/classes*)

The Java Interface

Class XMLRPCService

`XMLRPCService` extends `com.softwareag.entirex.aci.RPCService`. See `XLRPCService` (EntireX XML/SOAP Runtime) in the Javadoc documentation of the Java ACI.

The HTTP Interface

The HTTP interface supports HTTP `POST` Request and HTTP `GET` Request. The following HTTP headers and parameters are available:

Header/Parameter	Direction	Description
`exx-brokerID`	IN	If this attribute is set, it overwrites both the properties of the XMM file and the settings of the servlet initialization.
`exx-service`	IN	The service name is the triple set of server class/server name/service. If this attribute is set, it overwrites both the properties of the XMM file and the settings of the servlet initialization.
`exx-userID`	IN	The userID specified here is used for calling the broker.
`exx-password`	IN	The password specified here is used for calling the broker.
`exx-use-security`	IN	Possible values: `true | false`. This optional parameter determines whether EntireX Security is used. If the parameter is *not* defined for the XML/SOAP Listener / XML Tester (Quick test), the runtime determines if EntireX Security is required or not. If this parameter is defined, the behavior is fixed depending on value. See *EntireX Security for EntireX Broker* in the EntireX Security documentation.
`exx-encryption-level`	IN	Possible values: `0 | 1| 2`. See `ENCRYPTION-LEVEL`, class `Broker` and method `BrokerSecurity` in the Javadoc documentation of the Java ACI.

Header/Parameter	Direction	Description
`exx-rpc-userID`	IN	The RPC user ID specified here is used for Natural Security. If no RPC user ID and no RPC password is defined, the values of `exx-userID` and `exx-password` are used for these values.
`exx-rpc-password`	IN	The RPC password specified here is used for Natural Security. If no RPC user ID and no RPC password is defined, the values of `exx-userID` and `exx-password` are used for these values.
`exx-natural-security`	IN	Possible values: `true` | `false`. Determines whether Natural Security is used. See *Using Natural Security* in the Java Wrapper documentation.
`exx-natural-library`	IN	The Natural library. Applicable only if `exx-natural-security` is "`true`". See *Using Natural Security* in the Java Wrapper documentation.
`exx-use-codepage`	IN	Determines the translation processing of the EntireX Broker. Valid values: `true` | `false` | <`character encoding`>. If a character encoding is set, this character encoding is used for RPC message. See method `useCodePage` and `setCharacterEncoding` in the documentation on class `BrokerService` in the Javadoc documentation of the Java ACI.
`exx-compresslevel`	IN	Sets the compression level. See *Using Compression* under *Writing Advanced Applications - EntireX Java ACI*.
`exx-compression`	IN	Possible values: `true` |`false`. See ***Using Compression***.
`exx-xml-sessionID`	IN OUT	This HTTP header is returned from the servlet to the client application. If the client returns it to the servlet with the preceding call, the same session will be used. For conversations, the `exx-conv` parameter is also required.
`exx-xml-info`	OUT	In this HTTP header additional information is returned.
`exx-xml-error`	OUT	In this HTTP header error information is returned.
`exx-conv`	IN	Possible values: `OPEN` | `COMMIT` | `BACKOUT`. Conversations can only be used in connection with sessions. If the session is interrupted, the conversation is deleted.
`exx-reliable`	IN	Possible values: `AUTO-COMMIT` | `OFF`. See ***Reliable RPC for XML/SOAP Wrapper***.

20 XML Structures and IDL-XML Mapping

To understand the functionality and usage of the XML Mapping Editor, it is necessary to look at the possible XML structures and how they are mapped to Software AG IDL.

XML Structure Description

An XML structure is the type of an XML document, that is, the blueprint to build or parse the XML document. Every program of every library within a Software AG IDL file corresponds to at least two XML structures: one for the incoming and one for the outgoing XML document. The Error or Fault directions are also described as XML structures. There are InErr and OutErr XML structures that are returned by the broker or server in case of broker or data errors or servicing problems. The XML structures all start with one root node (corresponding to the library/program combination of the Software AG IDL file), and all XML elements and attributes are linked under that root node and may be further cascaded.

The XML structure may be represented as a tree of XML structure nodes (so-called XML parts). For the EntireX XML/SOAP Wrapper, no cyclic XML structures (that is, non-tree XML structures) are allowed because mapping to Software AG IDL parts would be illegal. However, in general this is not a severe restriction because Software AG IDL parts may not be cyclic either.

The XML parts have various properties. Important properties are the node name (tag name) and the node type (element or attribute). Other properties are the data type and length, minimum and maximum occurrence, the default values, the encoding, or the used or defined namespace. XML parts may have links to IDL nodes (their IDL mapping links, see below).

In the XML Mapping Editor, the XML structures are displayed as node trees. For XML structures, sample XML documents can be generated to visualize the expected or generated XML documents.

Basic IDL-XML Mapping

Mapping between IDL and XML describes the location of the data in the XML documents that correspond to the RPC parameters. Getting data for the RPC request is called incoming direction, putting RPC response data into an XML document is called outgoing direction. RPC parameters can correspond to elements or attributes.

There are various basic strategies for generating elements or attributes from a given Software AG IDL. The most important strategies are:

- Element-preferred mapping: Model the library/program and every IDL parameter as an element. The elements are cascaded in the same way as their IDL counterparts. The tree of IDL parts and the tree of XML elements are very similar. Arrays or repeated groups may get envelope elements.

- Attribute-preferred mapping: Model the library/program as root element. Model Arrays and repeated groups as elements, possibly with envelope elements around them. Model all other IDL parameters as attributes.
- SOAP: Construct a SOAP message format according to the SOAP specification. Every Software AG IDL part relates to an element in the SOAP:Body portion. Further fine-tuning according to the specification may be done.

Some users prefer element modelling, others like to add attributes to existing elements wherever possible. To minimize the work of building XML structures in the XML Mapping Editor, default mappings are available. The XML Mapping Editor allows you to choose an element-preferred, an attribute-preferred or a SOAP-conformant strategy (plus various detail level switches). The element-preferred mode generates only element nodes, whereas the attribute-preferred mode generates attributes wherever possible.

Example

IDL File

```
Library 'EXAMPLE' Is
 Program 'CALC' Is
  Define Data Parameter
   1 Operation (A1) In
   1 Operand_1 (I4) In
   1 Operand_2 (I4) In
   1 Function_result (I4) Out
  End-Define
```

Element-preferred Mode

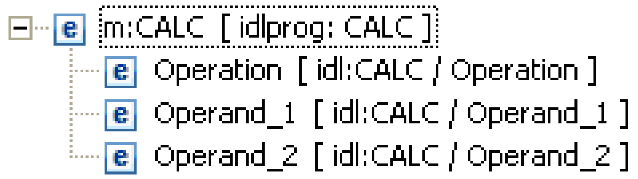
In the element-preferred mode, the IDL is mapped to the incoming XML document:

```
<CALC>
    <Operation> ... </Operation>
    <Operand_1> ... </Operand_1>
    <Operand_2> ... </Operand_2>
</CALC>
```

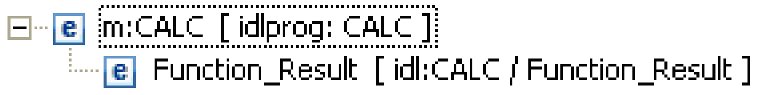
and the outgoing XML document:

```
<CALC>
    <Function_result> ... </Function_result>
</CALC>
```

The corresponding XML structure trees are:


m:CALC [idlprog: CALC]
Operation [idl:CALC / Operation]
Operand_1 [idl:CALC / Operand_1]
Operand_2 [idl:CALC / Operand_2]

and


m:CALC [idlprog: CALC]
Function_Result [idl:CALC / Function_Result]

Attribute-preferred Mode

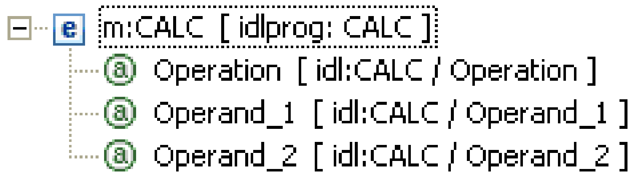
In the attribute-preferred mode, the Software AG IDL above is mapped to the incoming XML document:

```
<CALC Operation="..." Operand_1="..." Operand_2="..."/>
```

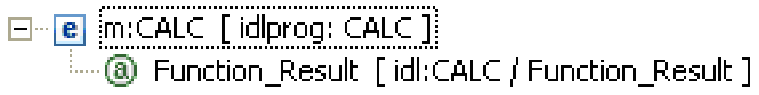
and the outgoing XML document.

```
<CALC Function_result="..."/>
```

The corresponding XML structure trees are:


m:CALC [idlprog: CALC]
Operation [idl:CALC / Operation]
Operand_1 [idl:CALC / Operand_1]
Operand_2 [idl:CALC / Operand_2]

and


m:CALC [idlprog: CALC]
Function_Result [idl:CALC / Function_Result]

SOAP-conformant Mapping

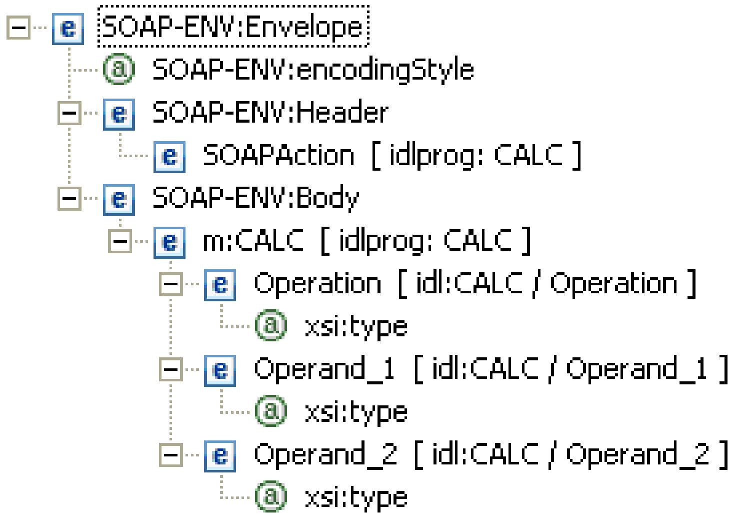
In SOAP-conformant mapping or the SOAP-conformant mode, the above Software AG IDL is mapped to the incoming XML document:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
   <SOAP-ENV:Header></SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <CALC>
         <Operation xsi:type="SOAP-ENC:string">...</Operation>
         <Operand_1 xsi:type="SOAP-ENC:int">...</Operand_1>
         <Operand_2 xsi:type="SOAP-ENC:int">...</Operand_2>
      </CALC>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

and the outgoing XML document:

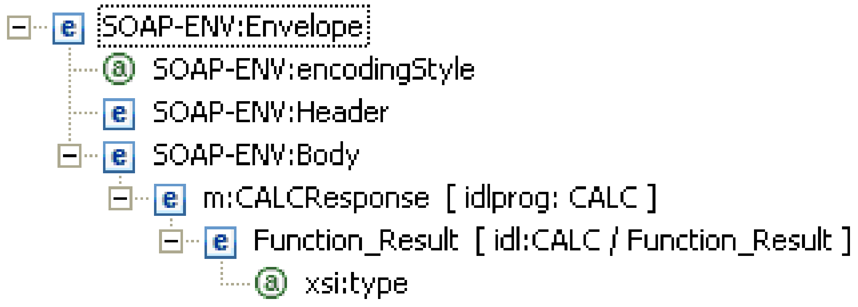
```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
   <SOAP-ENV:Header></SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <CALCResponse>
         <Function_result xsi:type="SOAP-ENC:int">...</Function_result>
      </CALCResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The corresponding XML structure trees:



```
SOAP-ENV:Envelope
    SOAP-ENV:encodingStyle
    SOAP-ENV:Header
        SOAPAction [ idlprog: CALC ]
    SOAP-ENV:Body
        m:CALC [ idlprog: CALC ]
            Operation [ idl:CALC / Operation ]
                xsi:type
            Operand_1 [ idl:CALC / Operand_1 ]
                xsi:type
            Operand_2 [ idl:CALC / Operand_2 ]
                xsi:type
```

and



```
SOAP-ENV:Envelope
    SOAP-ENV:encodingStyle
    SOAP-ENV:Header
    SOAP-ENV:Body
        m:CALCResponse [ idlprog: CALC ]
            Function_Result [ idl:CALC / Function_Result ]
                xsi:type
```

Arrays

The IDL may contain array parameters, i.e. basic data types that are to be repeated for a specified number of times. See following example:

```
...
1 Operation (A1/5) In
...
```

The `Operation` may be repeated 5 times. In this case `Operation` must correspond to an XML element, because with attributes the repetition cannot be modelled. The corresponding XML document may then contain up to 5 `Operation` elements:

```
...
<Operation> ... </Operation>
<Operation> ... </Operation>
<Operation> ... </Operation>
<Operation> ... </Operation>
<Operation> ... </Operation>
...
```

or it may contain a surrounding parent element (envelope) `Operation`:

```
...
<Operations>
   <Operation> ... </Operation>
   <Operation> ... </Operation>
   <Operation> ... </Operation>
   <Operation> ... </Operation>
   <Operation> ... </Operation>
</Operations>
...
```

A switch in the XML Mapping Editor determines whether arrays get a surrounding parent element or not.

There is a maximum of three dimensions per IDL array. Each of the dimensions can have upper and lower bounds defined. The format is as follows:

```
1 Operation (A1/3,6,8) In
```

which corresponds to a possible XML document:

```
<Operations1>
   <Operations2>
      <Operations3>
         <Operation> ... </Operation>
         ...
      </Operations3>
      ...
   </Operations2>
   ...
</Operations1>
```

and XML structure:

```
...
Operations1 (element)
   Operations2 (element, minocc=1, maxocc=3)
      Operations3 (element, minocc=1, maxocc=6)
         Operation (element, minocc=1, maxocc=8)
```

Groups

The IDL may contain entries that represent groups of parameters.

Example

```
Library 'EXAMPLE' Is
 Program 'ADD' Is
  Define Data Parameter
   1 Parameters In
    2 Operand_1 (I4)
    2 Operand_2 (I4)
   1 Function_result (I4) Out
  End-Define
```

The entry parameters are a group of two Operands. Groups will always be converted to XML elements; the group elements will be mapped either to elements or to attributes.

Element-preferred mode

```
<ADD>
   <Parameters>
      <Operand_1> "..." </Operand_1>
      <Operand_2> "..." </Operand_2>
   </Parameters>
</ADD>
```

Attribute-preferred mode

```
<ADD>
   <Parameters Operand_1="..." Operand_2="..."/>
</ADD>
```

Array of Groups

Groups can be used in arrays, for example:

```
1 myparams (/5,4) In
  2 Operation (I4)
  2 Description (A80)
  2 Mygroup
    3 Operand_1 (I2)
    3 Operand_2 (I2)
  2 Options (A1/4)
```

Using the element-preferred mode, this IDL structure may be mapped to:

```
<Myparms1>
   <Myparms2>
     <Myparms>
       <Operation> "..." </Operation>
       <Description> "..." </Description>
       <Mygroup>
         <Operand_1> "..." </Operand_1>
         <Operand_2> "..." </Operand_2>
       </Mygroup>
       <Options1>
         <Options>"..."</Options>
         ...
       </Options1>
     </Myparms>
     ...
   </Myparms2>
   ...
</Myparms1>
```

The corresponding XML structure for the element-preferred strategy is then:

```
...
Myparms1 (element)
  Myparms2 (element, minocc=1, maxocc=5)
    Myparms (element, minocc=1, maxocc=4)
    Operation (element, I4)
    Description (element, A80)
    Mygroup (element, group, minocc=1, maxocc=1)
      Operand_1 (element, I2)
      Operand_2 (element, I2)
   Options1 (element, minocc=1, maxocc=4)
      Options (element, A1)
```

Grouping XML Elements or Attributes

You can introduce new elements by grouping one or more existing elements or attributes.

This is especially useful when the IDL contains many simple data types that could be semantically grouped. This will increase the level of hierarchies in the XML document without affecting the IDL.

Example

```
Library 'EXAMPLE' Is
  Program 'SIMPLE' Is
    Define Data Parameter
      1 par1 (A1) In
      1 par2 (A1) In
      1 par21 (A1) In
      1 par22 (I1) In
      1 par23 (I4) In
      1 par3 (A1) In
      1 par31 (I2) In
      1 par32 (I4) In
      1 par4 (I1) In
      1 par5 (A1) In
    End-Define
```

will be transformed by the attribute-preferred strategy to:

```
<SIMPLE par1="..." par2="..." par21="..."
par22="..."

par3="..." par31="..." par32="..."
par4="..." par5="..."/>
```

or with the element-preferred strategy to:

```
<SIMPLE>
   <par1> ... </par1>
   <par2> ... </par2>
   <par21> ... </par21>
   <par22> ... </par22>
   <par23> ... </par23>
   <par3> ... </par3>
   <par31> ... </par31>
   <par32> ... </par32>
   <par4> ... </par4>
   <par5> ... </par5>
<SIMPLE>
```

You can now reorganize the XML structure, for example to:

```
<SIMPLE par1="..." par4="..." par5="...">
     <par2 par21="..." par22="..." par23="...">... </par2>
     <par3 par31="..." par32="..."> ... </par3>
</SIMPLE>
```

IN / OUT / IN OUT Parameters

The incoming XML request must correspond to the incoming `IN` and `IN OUT` IDL parameters, and the (created) outgoing XML response must contain the `IN OUT` and `OUT` IDL parameters. In the incoming XML structure, there is no difference between `IN` and `IN OUT` parameters; the same applies to `IN OUT` and `OUT` parameters for the outgoing XML document.

Make sure that all IDL parameters marked as `IN` are properly mapped to (incoming) XML parts. Otherwise, the XML/SOAP Runtime can only assign a null representation (value “0”, empty string, Boolean “false” etc.) to the respective unmapped IDL parameters. This is probably not the desired value to be sent to the server. The XML Mapping Editor issues a warning when unmapped IN parameters are found.

21 XML Schema Standards Conformance (XML/SOAP Wrapper)

XML Schema Parser Standards Conformance

The XML Schema styles "Garden of Eden", "Russian Doll", "Salami Slice" and "Venetian Blind" are supported. Each `xsd:element` declaration containing at least one `xsd:element` or `xsd:attribute` will be interpreted as an IDL program.

Supported Features

- Element declaration
- Model group definition: group
- Model groups: all, choice, sequence
- Attribute declarations
- Attribute group definitions
- Simple type definitions
- Complex type definitions
- Wildcards: any (limited)
- Type derivation (limited)
- Anonymous types
- Nested element declaration (Russian doll design)
- Separate symbol spaces for elements, types, groups and attribute groups
- Abstract and/or Equivalency classes (limited)
- Target namespace resolver
- xsi:type
- Built-in simple types (primitive; simple derivation such as “restriction”)
- Constraining facets
- Date/time as per ISO 8601
- Import
- Include (limited)

Unsupported Features

- Identity constraints: unique, key, keyref: cannot be translated into XML structure nodes
- Block
- Built-in simple types (e.g. union)
- Regular expressions in data types or patterns (only supported for date/time values)
- Substitution group
- Recursive data type definition

XML Schema Writer Standards Conformance

Supported Features

- Element declaration
- Attribute declarations
- Model groups: choice, sequence
- Nested element declaration (Russian doll design)
- Type derivation (limited)
- Anonymous types
- Simple type definitions
- Complex type definitions
- Built-in simple types (primitive; simple derivation such as “restriction”)
- Constraining facets
- Date/time as per ISO 8601 subset
- Annotations for XML mapping information
- Separate symbol spaces for elements, types, groups and attribute groups
- xsi:type (limited)

Unsupported Features

- Other Model groups or group definition (e.g. attribute groups)
- Wildcards: any
- Abstract and/or Equivalency classes (limited)

22 Reliable RPC for XML/SOAP Wrapper

Introduction to Reliable RPC

Reliable RPC is used to send messages to a persisted Broker service. The messages are described by an IDL program and contain only IN parameters. The client interface object and the server interface object are generated from the IDL file, using the EntireX XML/SOAP Wrapper. For the generation there are no options to set.

Reliable RPC is enabled at runtime. The client has to set the mode for reliable RPC.

For XML/SOAP Wrapper, the only supported mode is `AUTO_COMMIT`, which commits each message directly after sending it.

The server is implemented and configured in the same way as for normal RPC.

Writing a Client

The client has to set the parameter `exx-reliable` in the HTTP header or in the XML/SOAP payload. For more information see ***Writing Advanced Applications with the XML/SOAP Wrapper***.

Broker Configuration

A Broker configuration with `PSTORE` is recommended. This enables the Broker to store the messages for more than one Broker session. These messages are still available after Broker restart. The attributes `STORE`, `PSTORE`, and `PSTORE-TYPE` in the Broker attribute file can be used to configure this feature. The lifetime of the messages and the status information can be configured with the attributes `UWTIME` and `UWSTAT-LIFETIME`. Other attributes such as `MAX-MESSAGES-IN-UOW`, `MAX-UOWS` and `MAX-UOW-MESSAGE-LENGTH` may be used in addition to configure the units of work. See *Broker Attributes* in the administration documentation.

23 SOAP and Web Services (XML/SOAP Listener)

The EntireX XML/SOAP Wrapper supports SOAP version 1.1 and 1.2 and enables you to create Web services easily.

See also *EntireX Web Services Wrapper*.

SOAP Support

The XML Mapping Editor provides a default mapping for SOAP, both for the currently displayed program as well as for all programs in all libraries of the current IDL file. That is, the latter function can be used to easily SOAP-enable all programs in an IDL file. See *IDL to XML Mapping with the XML Mapping Editor* in the XML Mapping Editor documentation. The following mapping parameters can be chosen to fine-tune the SOAP default mapping:

- XML Schema URN for the XML Schema version to be used. This will define the xsd prefix in the `SOAP-ENV:Envelope` element, at the beginning of the SOAP message.
- Namespace URI of the "payload" element node (this is the singleton element under the `SOAP-ENV:Body` element). Various SOAP toolkits require certain namespace settings. The namespace prefix will always be "m".

Note that the SOAP default mapping also works for the "error" direction, where it defines the "standard" SOAP fault message structure, as described in the specification.

The XML/SOAP Runtime detects whether SOAP is used in incoming messages. If so, the special elements and attributes of the SOAP specification (`SOAP-ENV:Envelope`, `SOAP-ENV:Body`, `type`, `arrayType` etc.) are handled accordingly.

Web Services

Web services are programmable, distributed application components accessible on the Web using solely standard internet protocols. See *Introduction to Web Services in EntireX* in the Web Services Wrapper documentation.

You may create Web services with the EntireX XML/SOAP Listener by wrapping IDL programs in SOAP message structures, as described in the preceding section, and then generating service descriptions for them using the WSDL Wizard. The service descriptions use the de facto standard WSDL (Web Service Description Language) format.

To make Web services out of your IDL programs

1 Select the IDL file that you want to turn into one or more Web services.

2 From the context menu, choose **Open With ... > EntireX XML Mapping Editor** and choose SOAP for mapping. Choose **Generate**.

3 Set up a servlet engine and configure the XML/SOAP Listener. See *Administering the EntireX XML/SOAP Listener* in the UNIX and Windows administration sections.

4 Select the XMM or IDL file. Open the context menu and choose the **Generate Web Service from EntireX Mapping** (**Generate Web Service from Software AG IDL**).

5 The generated WSDL file and the deployment unit (AAR file) can be found in the same folder as the IDL file.

See *Software AG IDL Extractor for WSDL*.

24 Support of Representational State Transfer (REST)

The XML/SOAP Listener supports REST architecture.

GET Manner

The XML/SOAP Listener supports simple requests in GET manner. In this context, GET means retrieving information (in the form of an entity) that is identified by the request URI.

A service accepts a request in plain XML, so the response format is also plain XML. The parameter names *must* match those of the service.

The request is formed as follows:

```
<URL_of_the_XML_service>/<program_name>?<parameter_1>=<value_1>&<parameter_2>=<value_2>...
```

Limitations

- The request document cannot contain groups, arrays, other nested elements, or attributes.
- Elements cannot be namespace-qualified.
- Only services that accept requests in plain XML can be called.

Example

Enter the following GET request in the URL field of your Web browser:

```
http://localhost:8080/wsstack/services/example/CALC?Operation=+&Operand_1=20&Operand_2=36
```

The actual request is:

```
GET /wsstack/services/example/CALC?Operation=+&Operand_1=20&Operand_2=36 HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset=UTF-8;action="CALC";
SOAPAction: CALC
User-Agent: Axis2
Host: localhost:8080
```

Your browser will display something like the following:

```
<CALC>
    <Function_Result>56</Function_Result>
</CALC>
```

The actual response is:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Thu, 07 Aug 2008 12:25:44 GMT

32
<CALC><Function_Result>56</Function_Result></CALC>
0
```