webMethods EntireX

EntireX Web Services Wrapper

Version 10.8

October 2022

WEBMETHODS

This document applies to webMethods EntireX Version 10.8 and all subsequent releases.

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

Document ID: EXX-EEXXWEBSERVICESWRAPPER-108-20220601

Table of Contents

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

In addition, you can also access the cloud product documentation via **https://www.softwareag.cloud**. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at **https://knowledge.softwareag.com**.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Introduction to Web Services in EntireX

Introduction

The EntireX Web Services Wrapper is a wizard that generates and optionally deploys Web services (Designer file with extension .aar) to offer an RPC server - for example a COBOL or Natural RPC server - as a Web service. The generated XML/SOAP mapping file (Designer file with extension .xmm) can also be used to enable RPC clients - for example a COBOL or Natural client - consuming (or calling) a Web service. This section covers the following topics:

- Generating Web Services
- Deploying Web Services
- Consuming (or Calling) Web Services from RPC Clients

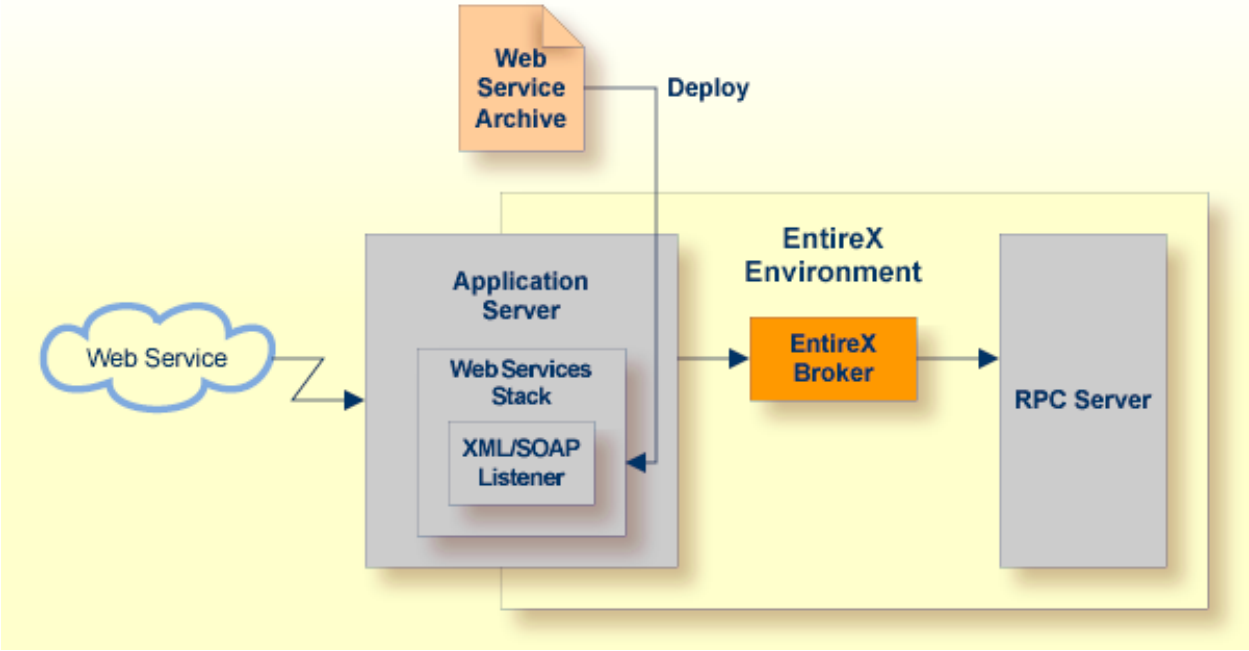
Generating Web Services

Web services are generated from Software AG IDL, XML/SOAP mapping files or Natural subprograms. The generated result is a Web service archive (Designer file with extension .aar) that contains the relevant artifacts of the Web service such as an XML/SOAP mapping file (Designer file with extension .xmm), WSDL file and additional configuration files, for example *services.xml*:

See *Generating Web Services from Software AG IDL File*.

Deploying Web Services

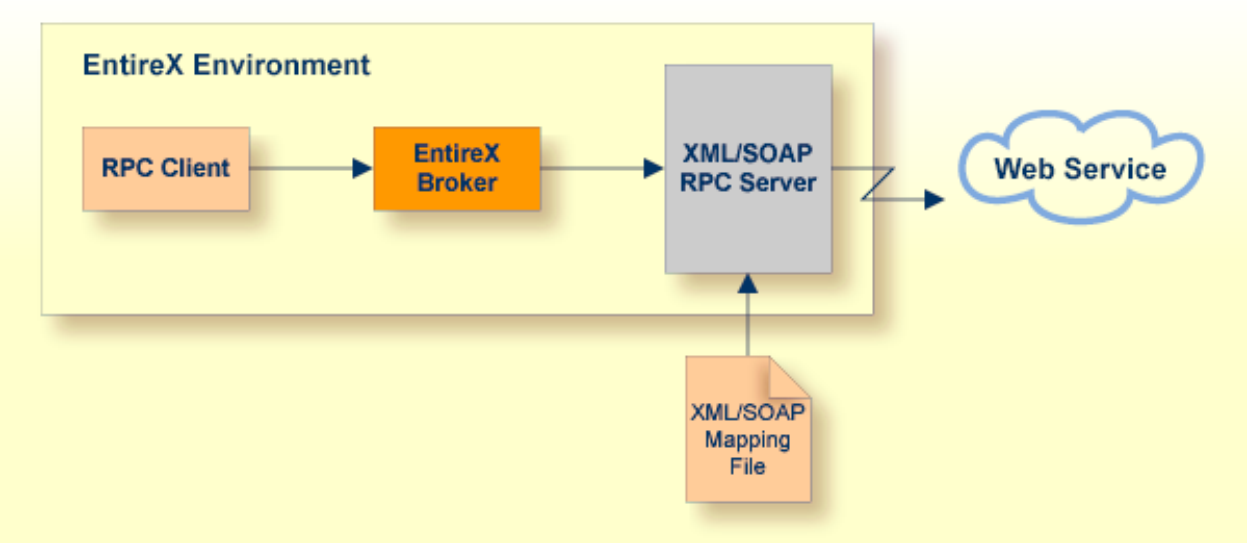
The Web service archive is deployed for execution by the wizard or - in an extra deployment step - in a Web Services Stack with the Listener for XML/SOAP running, for example, in an application server:



See *Deploying Web Services* under *Using the EntireX Web Services Wrapper*.

Consuming (or Calling) Web Services from RPC Clients

To enable an RPC client - for example a COBOL or Natural client - consuming (or calling) a Web service, the generated XML/SOAP mapping file (Designer file with extension .xmm) is used together with RPC Server for XML/SOAP:



See also *Developing Web Service Client Applications*.

Supported Features

EntireX supports a number of advanced Web services features in combination with the Web Services Stack. This includes support for

- SOAP 1.2 according to *http://www.w3.org/TR/soap12-part1/* in addition to SOAP 1.1. No extra configuration is needed.
- SOAP 1.2 messaging
- SOAP 1.2 binding in WSDL 1.1
- Multiple transports (HTTP, HTTPS, TCP). See *Transports*.
- WSDL 1.1; The generated descriptions are compliant with the Web Services Description Language (WSDL 1.1 - *http://www.w3.org/TR/wsdl*). They contain both SOAP 1.1 and SOAP 1.2 binding definitions and endpoints. Example (excerpt from a WSDL file):

```
...
    <wsdl:service name="Calc">
      <wsdl:port name="CalcSOAP11port_http" binding="ns0:CalcSOAP11Binding">
        <soap:address location="http://host:port/wsstack/services/Calc" />
      </wsdl:port>
      <wsdl:port name="CalcSOAP12port_http" binding="ns0:CalcSOAP12Binding">
        <soap12:address location="http://host:port/wsstack/services/Calc" />
      </wsdl:port>
    </wsdl:service>
    ...
    ↩
```

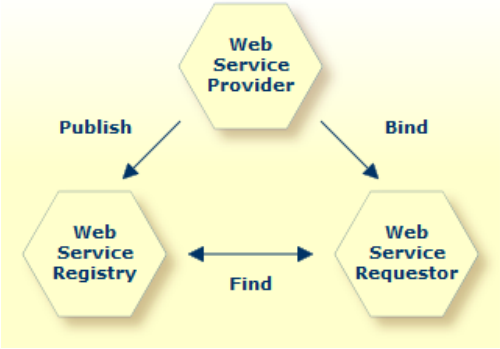
- WS-I Basic Profile: If the WSDL generation format document/literal is used, the generated Web service is compliant with WS-I Basic Profile 1.1 (see *http://www.ws-i.org*).
- WS-Policy (*WS-Addressing*, *WS-Security*, *WS-ReliableMessaging*)
- WS-Policy Attachment to WSDL 1.1

Web Services

Web services are programmable, distributed application components accessible on the Web using solely standard internet protocols. In contrast to the current "document Web", which specializes in human interaction, Web services are designed to be accessed by programs to form a new application architecture, the "application Web".

Generally speaking, a Web service application consists of three major Web service components:

- A Web service registry, which stores information about Web service providers and Web services.
- A Web service client, which makes use of a service offered on the Web using a standard messaging and transport protocol. Web service clients can search Web service registries to find desired services.
- A Web service, which is accessible via a standard messaging and transport protocol. Web services publish information about themselves in a Web service registry. A Web service must provide a precise technical description of its interfaces to be used by clients.



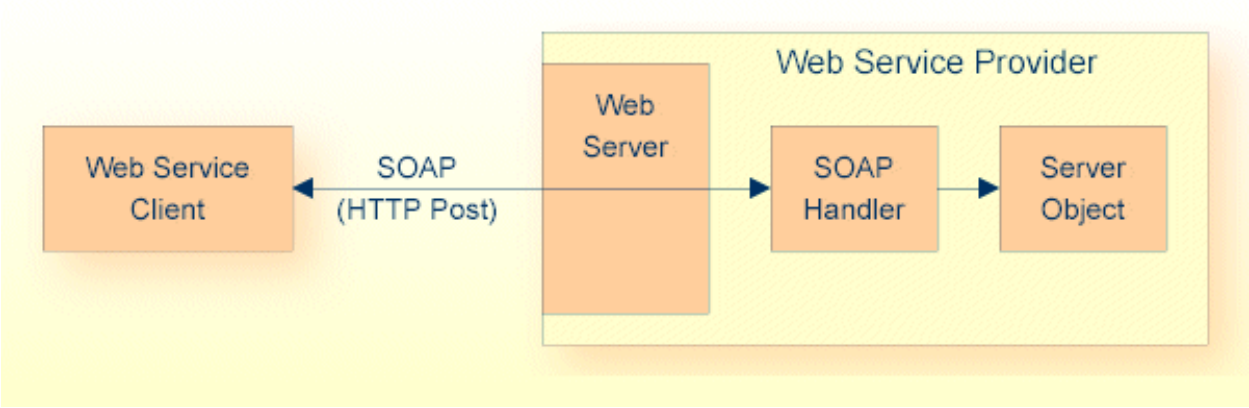
The standards on which Web services are based today are:

- HTTP and SMTP for basic network transport services,
- XML as data format,
- the Simple Object Access Protocol (SOAP) for XML messaging and RPC,
- the Web Service Description Language (WSDL) for service descriptions and
- Universal Description, Discovery and Integration (UDDI) for Web service registries.

The Simple Object Access Protocol (SOAP)

SOAP (originally Simple Object Access Protocol) (SOAP 1.1) is a messaging and RPC protocol designed for integrating heterogeneous Web services in the internet. It defines a message format in the Extensible Markup Language (XML) that can be transported over existing internet transport protocols (HTTP, SMTP, FTP or others). By using standard XML, SOAP messages are self-describing, that is, they carry enough information for a receiver to decompose and process the message in a standard way. By using standard internet protocols, SOAP seamlessly fits into existing internet infrastructure (for example, routers, firewalls, Web servers).

For more details, see the World Wide Web Consortium's note at *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/*.



Web Services Registries and CentraSite

Web services created with EntireX can be registered in any UDDI registry, including CentraSite. CentraSite offers enhanced registry functionality, and also repository functionality that enables you to store Web services artifacts and register interdependencies for impact analysis.

Web Service Architecture

SOAP is one of the basic technologies required to build Web services. It is combined with the related technologies Web services description language (e.g. WSDL) for describing Web services, and Web service registries (e.g. UDDI based) for storing information about Web services.

- A Web service provider publishes a description of the service it offers to a Web services registry;
- A Web service client contacts a Web services registry to find the service, and

- uses the Web service description to actually bind to the Web service.

SOAP can be used for publish, find and bind operations.

The following level of SOAP and Web services functionality is provided:

- SOAP enabling of EntireX RPC servers
- generation of WSDL service descriptions for EntireX RPC servers
- generation, configuration and deployment of Web services into the Software AG Web Services Stack runtime

General SOAP Architecture

EntireX uses the Software AG Web Services Stack. WSS is a toolkit that provides functionality for execution, configuration and management of Web services.

The core part of the WSS runtime is the SOAP engine, which is based on Apache Axis2.

The Software AG Designer provides functionality to create, configure, and deploy EntireX Web services. EntireX Web services are packaged into a service archive (extension .aar).

Incoming SOAP requests are processed by the WSS SOAP engine. The SOAP request is given to the XML/SOAP Runtime, which validates the request and transforms it into an RPC request. The result of the RPC request in turn is transformed into a SOAP response message and sent back to the client. If an error occurs, a SOAP fault message is sent back to the client.

3 Using the EntireX Web Services Wrapper

Generating Web Services from Software AG IDL File

A typical scenario starts from an existing (legacy) server application "wrapped" with EntireX technology and accessible to RPC clients via the EntireX RPC protocol. The interface of the (legacy) server is described by an IDL file (see *Software AG IDL File* in the IDL Editor documentation). If there is a related server mapping file (Natural | COBOL), this is also used (internally). Using the EntireX Web Services Wrapper, the (legacy) server is exposed as a Web service. For example, the following files are generated from *example.idl*:

- a SOAP mapping (*example.xmm*)
- a WSDL description (*example.wsdl*)
- a service archive for the Web Services Stack (*example.aar*)

This section covers the following topics:

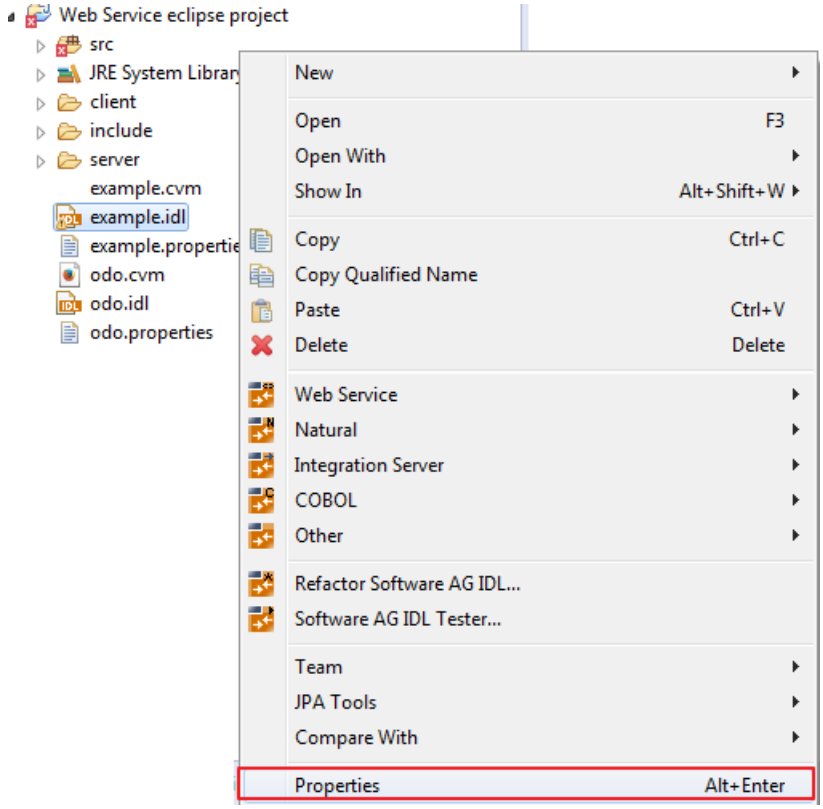
Generating a Web Service

This section describes the general approach for generating a Web service archive with the Web Services Wrapper.

To generate a Web service

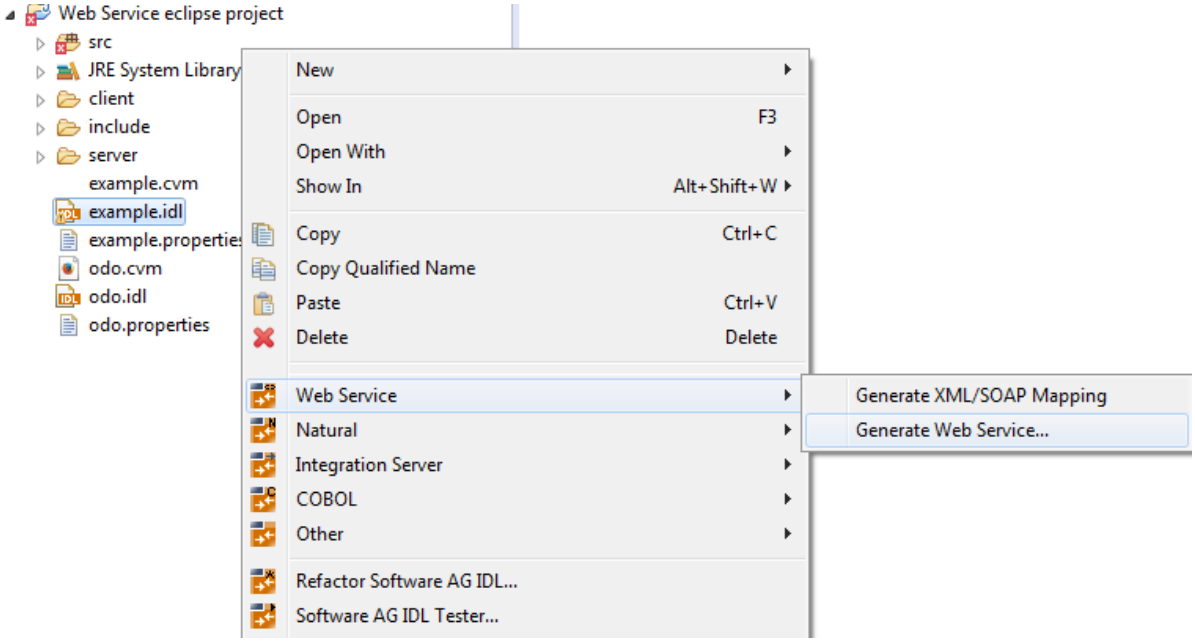
1 Before the wizard is started for the first time, initialize the preference pages **Window** > **Preferences** > **Software AG** > **EntireX** and **Window** > **Preferences** > **Software AG** > **EntireX** > **Web Services Wrapper** with values appropriate for your environment.

2 Select the IDL file to be processed. From the context menu of this IDL file, choose **Properties**.



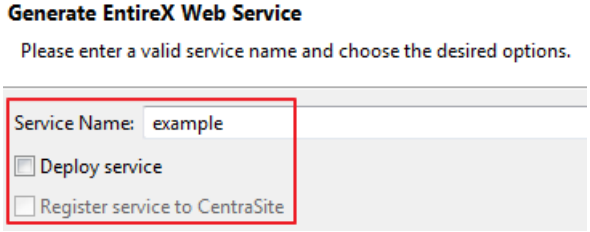
- Change the EntireX settings if necessary.
- If necessary, change the Web service generation settings using the WSDL tab (**Service Name** and **Service URL**).
- Choose **OK** to leave the **Properties** dialog.

3 Select the IDL file to be processed. If there is a related server mapping file (Natural | COBOL), this is also used (internally). From the context menu of the IDL file, choose **Web Service** > **Generate Web Service...**.



You can select more than one *Software AG IDL File* in the IDL Editor documentation to merge all IDL files into one Web service. As a result you will get multiple XML mapping files, one WSDL file and one Web service archive. Merging does not support the use of the same IDL program name in different IDL libraries.

4 The EntireX Web Services Wrapper is launched:

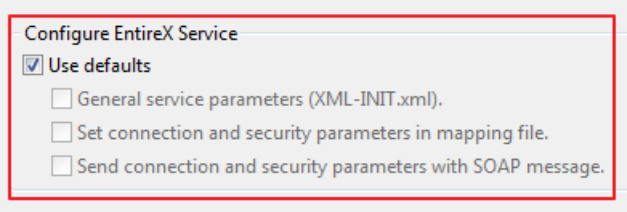


You can enter a service name. The default name is the name of the selected IDL file.

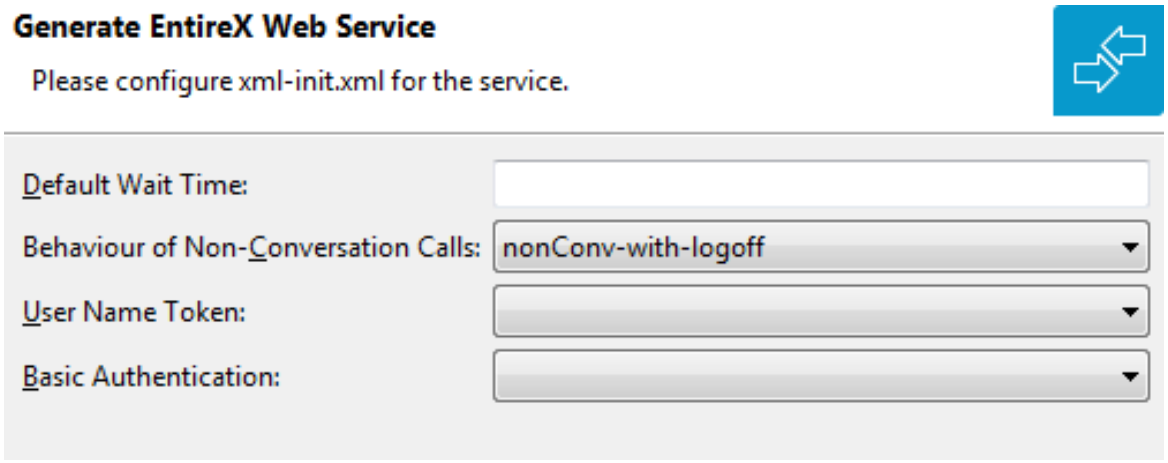
If you check **Deploy service**, an additional confirmation page is displayed. See *Deploying Web Services* for this dialog.

If you check **Register service to CentraSite**, a confirmation page is displayed. See *EntireX CentraSite Integration* for this dialog.

If you uncheck **Use defaults** for the **Configure EntireX Service** section, you can select the following configuration items:

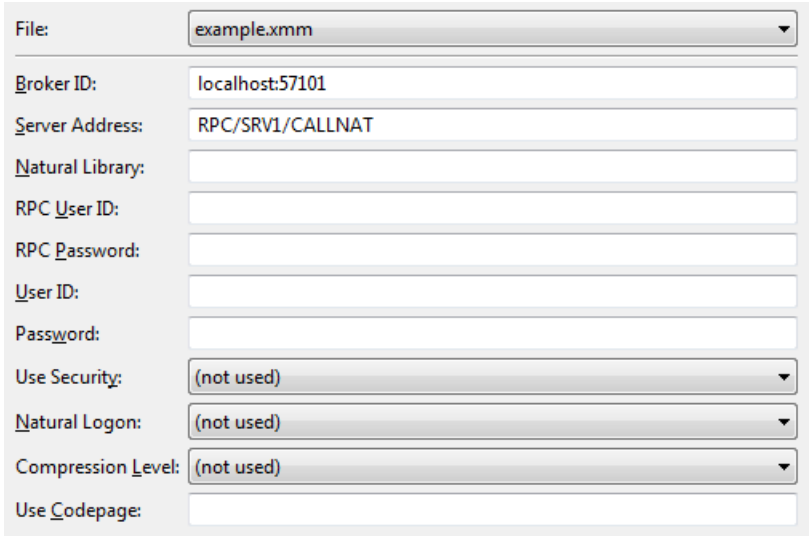


- **General service parameters (XML-INIT.xml)**
  An additional configuration page is appended.



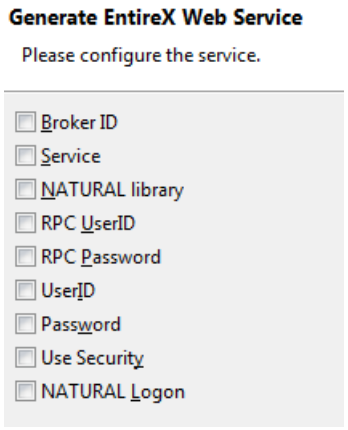
  The parameters on this page are described in the *Web Services Stack Configuration Editor*. See also *Initialization Parameters for the Listener for XML/SOAP*.

- **Set connection and security parameters in mapping file**
  An additional configuration page is appended.



The parameters on this page are described in the *Web Services Stack Configuration Editor*. See also *Service Parameters*.

- **Send connection and security parameters with SOAP message**
  An additional configuration page is appended.



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

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

A web service client will then be able to set these parameters in the SOAP header of the SOAP message. See also *The HTTP Interface* in the XML/SOAP Wrapper documentation.

5 Choose **Next**, enter your configuration parameters and select the methods for which the Web service is to be generated.

6 Choose **Finish** to generate the Web service (XML/SOAP mapping file (Designer file with extension .xmm), WSDL file and Web service archive (Designer file with extension .aar)).

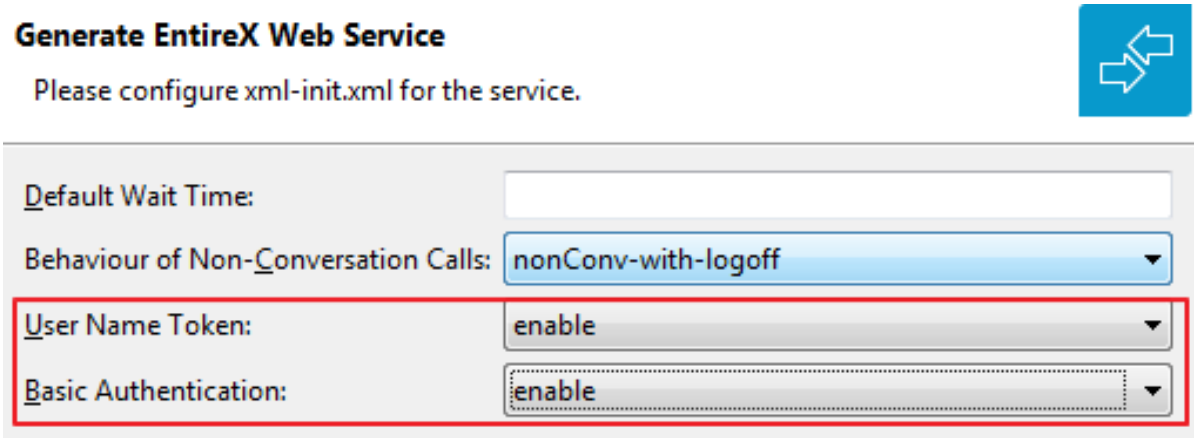
Generating a Web Service with HTTP Basic Authentication and UsernameToken Authentication for EntireX Authentication

This section describes specific settings required when you generate a Web service archive (Designer file with extension .aar) with the Web Services Wrapper for HTTP Basic Authentication and UsernameToken Authentication.

To generate a Web service with HTTP Basic Authentication and UsernameToken Authentication

- In general, follow the steps under *Generating a Web Service*.

  In step 4, check **General service parameters (XML-INIT.xml)** and in the additional configuration page enable **User Name Token** and **Basic Authentication**.



  The priority of credentials settings is as follows:

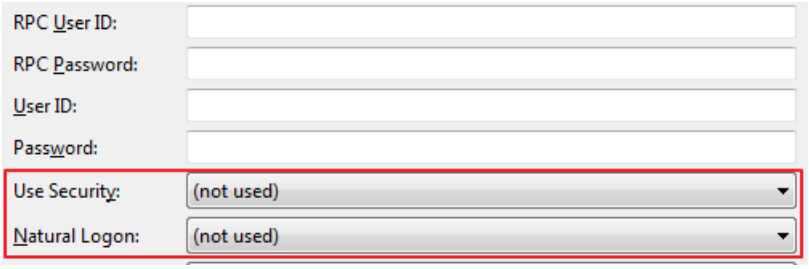
  1. `exx-userID`, `exx-password`, `exx-rpc-userID`, `exx-rpc-password` (highest priority)
  2. `UsernameToken`
  3. Basic Authentication (lowest priority)

Generating a Web Service for EntireX Security or Natural Security

This section describes specific settings required when you generate a Web service archive (Designer file with extension .aar) with the Web Services Wrapper for EntireX Security or Natural Security.

To generate a Web service for EntireX Security or Natural Security

- In general, follow the steps under *Generating a Web Service*. In step 4, check **Set connection and security parameters in mapping file** and in the additional configuration page enable **Use Security** and/or **Natural Logon**. If required, also set the Natural library.



Deploying Web Services

Prerequisites

The following resources are required to deploy and run a Web service:

- An application server where the Web Services Stack is installed (wsstack.war). The Web Services Stack is accessible by default at the URL *http://<host-name>:<port-number>/wsstack/sagdeployer* with port number 10010. The default port can be changed during installation. In the case of deployment in custom application servers, the port is configured by the corresponding server administration tools. For more details see the Web Services Stack documentation in the *Software AG Infrastructure Administrator's Guide*, also available under ***http://documentation.software-ag.com*** > *Guides for Tools Shared by Software AG Products*.
- The EntireX Runtime (*entirex.jar*) containing the Listener for XML/SOAP. This must be located in the *WEB-INF\lib* folder of the Web Services Stack Web application. See *Listener for XML/SOAP*.
- The Eclipse plug-ins of the Web Services Stack must be installed.
- EntireX Broker and the RPC server hosting the server implementation are up and running. See *Setting up Broker Instances* in the platform-specific Administration documentation and *EntireX RPC Servers and Listeners*.

Note: Default password handling was enhanced in EntireX version 10.8 and all Software AG products that are installed with the Software AG Installer. During installation, on the **Administrator Password** panel, enter a default product administrator password for the products you are installing, and choose whether to require the password to be changed on first product login. We strongly recommend adopting the new password in the Web Services Stack preferences of the Designer. The password must be at least 8 characters long. The maximum number of consecutive identical characters (for example "aaa") or sequential characters (for example "123") is 3.

Deploying the Web Service

Deploying a Web service means sending a Web service archive (Designer file with extension .aar) to a running Web Services Stack Web application. The Web Services Stack Web application stores the Web service archive in the *WEB-INF/services* folder of the Web Services Stack Web application.

To deploy a Web service

1 From the context menu of the generated Web service archive, choose **Software AG Web Services Stack** > **Deploy Web Service Package**. In a wizard you can select hostname, port number, and a servlet address of the Web Services Stack Deployment Servlet. You also need to supply your credentials (user ID and password).

2 Choose **Finish** to send the Web service archive to the selected deployment connection point.

Notes:

1. For more information, see *Deploy Web Services Stack* in the *Software AG Infrastructure Administrator's Guide*, also available under ***http://documentation.softwareag.com*** > *Guides for Tools Shared by Software AG Products*.
2. You can verify the deployment of your service with context menu item **Software AG Web Services Stack >View Web Services Stack...** or **Software AG Web Services Stack > View Web Service**.
3. An advanced Web service application (e.g. requiring WS-Security) may need special settings in the Web service archive before you deploy it to the Web Services Stack. You can manage the settings with the ***Web Services Stack Configuration Editor***.

Deploying Web Services Stack Runtime to WebSphere 8.5.5

If you want to deploy the Web Services Stack Web application to WebSphere 8.5.5, perform the following steps:

Note: Java 8 Extension is required.

To deploy to WebSphere 8.5.5

1 Copy `Software AG_directory`*/WS-Stack/webapp/wsstack.war* to a temporary location.

2 Unpack the WAR file.

3 Copy all MAR files from *WEB-INF/modules* to *WEB-INF/lib* and change their extensions to JAR.

Important: There might be an issue with mapping of MAR files when using Microsoft Office. When you have Microsoft Office installed, you cannot rename those files using Windows Explorer. In this case, use the *ren* command prompt command. For example, `<TEMP_Directory>\WEB-INF\modules>copy addressing-1.41.mar addressing-1.41.jar` copies the MAR file and changes its extension from MAR to JAR.

4 Copy *entirex.jar* to folder*../wsstack/WEB-INF/lib*.

5 Recreate the WAR file. You can use WinZip or any other application with support for ZIP files.

6 Log on to the Administrative Console and navigate to **Applications > New Application > New Enterprise Application**.

7 Enter the location of the *wsstack.war* file or browse for it using the **Browse** button. Then click **Next**.

8 Select the **Fast Path - Prompt only when additional information is required** radio button and then click **Next**.

9 Click **Next** and leave the default values for the options in the **Step 1 Select installation options**, **Step 2 Map modules for servers**, and **Step 3 Map virtual hosts for Web modules** screens.

10 Click **Next** to go to the **Step 4 Map context roots for Web modules** screen, and type in wsstack in the **Context root** field.

11 Click **Save** to save the changes to the master configuration.

12 Navigate to **Applications > Application Types > WebSphere Enterprise Applications**.

13 Click **wsstack_war** to open the configuration dialog.

14 In the configuration dialog, click the **Class loading and update detection** link.

15 For **Class loader order**, select **Classes loaded with local class loader first (parent last)** radio button.

16 For **WAR class loader policy**, select **Single class loader for application** radio button.

17 Navigate to **Applications > Application Types > WebSphere Enterprise Applications**.

18 Start the Web Services Stack Web application.

Testing Web Services

Testing a Web Service with the EntireX XML Tester

To test a Web Service with the EntireX XML Tester

- From the context menu of the generated Web service archive (Designer files with extension .aar), choose **Test EntireX Web Service**. This starts the *EntireX XML Tester*. If the Web service archive contains multiple XMM/SOAP mapping files (Designer file with extension .xmm), select the one you want. Refer to the documentation of the EntireX XML Tester to create a sample document.

WSDL Query of Web Services

You can retrieve the WSDL of a Web service deployed in the Web Services Stack running in a servlet engine.

To query the WSDL of a Web Service

- Use a browser and append "?wsdl" to the Web service URI. Example:

  ```
  http://host:port/wsstack/service/myService?wsdl
  ```

  The returned WSDL will return to the requestor all relevant configuration information of the Web service, for example all endpoints through which the Web service is accessible and policies that are in effect for the Web service.

Developing Web Service Client Applications

Once the Web service is up and running and its WSDL is accessible (using HTTP), Web service client applications can be developed. See also *Writing Web Service Client Applications* in the IDL Extractor for WSDL documentation.

Undeploying Web Services

Undeploying a Web service means informing a running Web Services Stack Web to remove a deployed Web service. The Web Services Stack Web removes the corresponding Web service archive from the WEB-INF/services folder of the Web Services Stack Web.

To undeploy a Web service

- Choose **Windows** > **Preferences** > **Software AG** > **Web Services Stack** > **Undeploy Web Service Package**...

Notes:

1. For more information, see *Deploy Web Services Stack* in the *Software AG Infrastructure Administrator's Guide*, also available under ***http://documentation.softwareag.com*** > *Guides for Tools Shared by Software AG Products*.
2. You can verify the undeployment with the help of a browser. The undeployed Web service should disappear from the list of the deployed Web services (e.g. *http://localhost:10010/wsstack/services/listServices*).

Removing Web Services

When a Web service is removed from an Eclipse project, using **Web Services Stack > Remove Web Service**, the following artifacts are additionally deleted depending on the source of the generated Web service.

Generated from	Additionally Deleted Artifacts
Natural/COBOL	■ IDL file ■ XMM/SOAP mapping file ■ Server mapping file (if applicable, see *Server Mapping Files in the Designer* for Natural | COBOL) ■ WSDL file
IDL File	■ XMM/SOAP mapping file ■ WSDL file
XMM File	■ WSDL file (if applicable)

4 Using the Web Services Wrapper in Command-line Mode

The Web Services Wrapper generates a WSDL file, a mapping file (extension .xmm) and a service archive (extension .aar) to deploy into the common Web Services Stack.

Command-line Options

See *Using EntireX in the Designer Command-line Mode* for the general command-line syntax. The table below shows the command-line option for the Web Services Wrapper.

Task	Command	Option	Description
Generate WSDL, mapping and archive files from specified IDL file.	`-wsdl`	`-out`	Output directory, absolute path (fully qualified, must exist). Ignored if the input is part of a project in the Eclipse workspace. Same as `-o`.
		`-url`	Service URL. Same as `-u`.
		`-service=<service>`	Service name.
		`-properties`	Use the file-specific properties. This option makes the others superfluous, but is only available if the input is part of an Eclipse project.

Example for Generating Web Services

```
<workbench> -wsdl  /Demo/example.idl  -properties
```

where `<workbench>` is a placeholder for the actual EntireX design-time starter as described under *Using EntireX in the Designer 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 within 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.

The generated mapping file gets the name of the IDL file. The WSDL file and the service archive get the name of the service, if specified, otherwise they get the name of the IDL file.

```
<workbench> -wsdl  /Demo/example.xmm  -properties
```

This command generates the WSDL file and the service archive from the mapping file. If a service name is specified, the WSDL file and the service file get the name of the service, otherwise they get the name of the mapping file.

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> -wsdl C:\Temp\example.idl
```

Uses the IDL file *C:\Temp\example.idl* and generates the files (*EXAMPLE.wsdl* and *example.xmm*) in parallel to the IDL file. Slashes and backslashes are permitted in the file name. Output to standard output:

```
Using workspace file:/C:/myWorkspace/.
LIBRARY = EXAMPLE
        Program   = CALC
        Program   = SQUARE
WSDL file "C:\Temp\EXAMPLE.wsdl" created.
Exit value: 0
```

Example 2

```
<workbench> -wsdl -help
```

or

```
<workbench> -help -wsdl
```

Both show a short help for the Web Services Wrapper.

Linux

Example 1

```
<workbench> -wsdl /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. The generated output (*EXAMPLE.wsdl* and *example.xmm*) will be stored in */Demo*, parallel to the IDL file.

Example 2

```
<workbench> -wsdl -help
```

or

```
<workbench> -help -wsdl
```

Both show a short help for the Web Services Wrapper.

5 Web Services Stack Configuration Editor

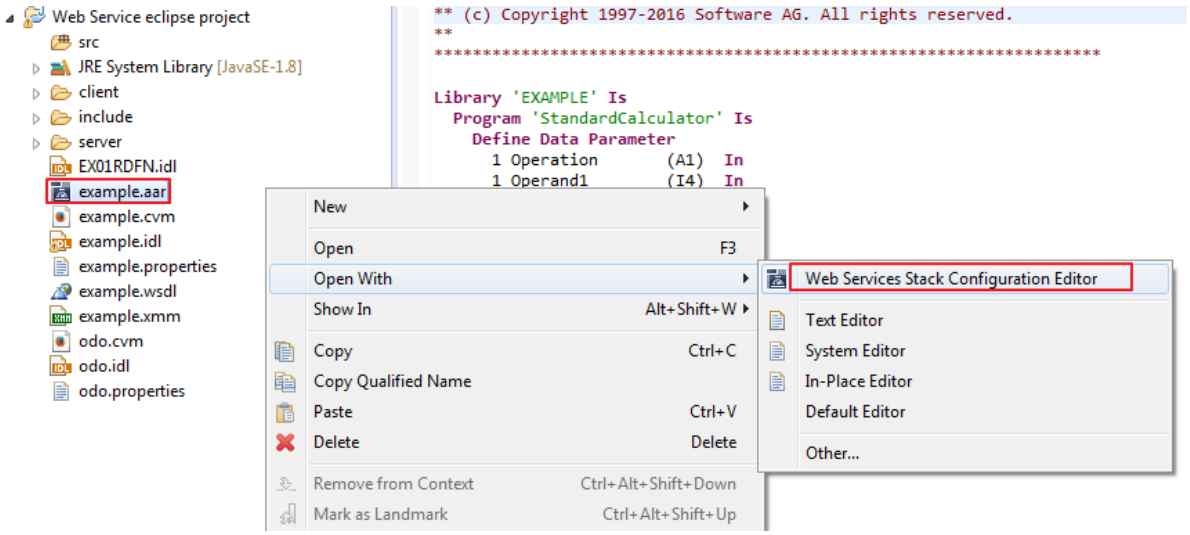
With the Web Services Stack Configuration Editor, an Eclipse plug-in, you can configure individual Web services or groups of Web services in the *services.xml* file that is part of a Web service archive (Designer file with extension .aar). Using an external configuration file of the Listener for XML/SOAP allows you later to override settings of a Web service archive for different Web server environments without modifying the archive itself. See also *Configuring Web Services*.

For more information on the Configuration Editor see the separate Web Services Stack documentation in the *Software AG Infrastructure Administrator's Guide*, also available under ***http://documentation.softwareag.com*** > *Guides for Tools Shared by Software AG Products*.

Introduction

To invoke the Web Services Stack Configuration Editor

- Use the context menu of a Web service archive (Designer file with extension .aar) that was generated with the Designer to open the Web Services Stack Configuration Editor:



The following pages are provided to configure different aspects of the Web service:

- **Archive Page**
  Displays the contents of the Web service archive (Designer files with extension .aar). In general it allows you to add additional files to the archive or remove files from the archive. Specifically you can add additional EntireX files (Designer files with extension .idl, and .xmm) to the Web service.

- **Services Page**
  See *Services Page*. Allows you to update and provide further configuration settings that apply to a Web service contained in the Web service archive (Designer file with extension .aar).
- **Operations Page**
  Allows you to provide additional configuration settings that apply to an operation of a Web service contained in the Web service archive.
- **services.xml Page**
  Allows you to view the Web services archive's configuration file in text form (XML format).
- **EntireX Settings Page**
  See *EntireX Settings Page*. Provides configuration and settings for the Listener for XML/SOAP. See *Listener for XML/SOAP*.

Notes:

1. The **Services Page** corresponds to the **Operation Page**. This means most settings and configuration options of the services and operation page are identical. You can override Web Service configuration settings for service and/or operation.
2. Web services created with the Web Services Wrapper have specific configuration settings defined during generation. See *Generating Web Services from Software AG IDL File*. These are:
   - ServiceLifecycle class: `com.softwareag.entirex.xml.rt.WSSServiceLifeCycle`
   - Session Scope: Application
   - MessageReceiver class: `com.softwareag.entirex.xml.rt.EntireXMessageReceiver`

   Do *not* modify these settings.
3. The global configuration for the Web services engine is done in the configuration file *axis2.xml*. See also *Configuring Web Services* in the Listener for XML/SOAP documentation.

Services Page

This section covers the following topics:

Transports

Web services can be configured to be accessible over multiple transport protocols. The default transport is HTTP.

- **HTTP**
  No additional configuration is required.
- **HTTPS**
  This requires that HTTPS is configured for the servlet engine that is running the Web Services Stack.
- **TCP**
  Additional configuration of the Web Services Stack in *axis2.xml* is necessary to enable support of this transport.

For more details see the Web Services Stack documentation in the *Software AG Infrastructure Administrator's Guide*, also available under ***http://documentation.softwareag.com*** > *Guides for Tools Shared by Software AG Products*.

WS-Addressing

To enable WS-Addressing headers for a Web service, check the **Enable WS-Addressing** check box in section **Modules**. This inserts a WS-Addressing policy into *services.xml* and enables the addressing module of the Web Services Stack that processes addressing SOAP headers.

```
<wsp:Policy wsu:Id="User defined"
xmlns:wsp=http://schemas.xmlsoap.org/ws/2004/09/policy
  xmlns:wsu="http://docs.oasis-open.org/.../...wssecurity-utility-1.0.xsd">
  <wsp:ExactlyOne>
    <wsp:All>
      <wsaws:UsingAddressing
        xmlns:wsaws="http://www.w3.org/2006/05/addressing/wsdl"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
<module ref="addressing"/>
```

WS-Security

WS-Security can be configured to ensure integrity, confidentiality and allow authentication of messages exchanged between Web services clients and Web services. To enable WS-Security for a Web service, check the **Enable WS-Security** check box in section **Modules**. This enables further configuration options in section **Security** on the **Services** page. This section covers the following topics:

Overview

WS-Security policy assertions can be defined for a service to accept and enforce SOAP messages containing a WS-Security SOAP header. With WS-Security the message exchange between a Web service client and a service can be secured in the following aspects:

- confidentiality: messages (or parts of messages) are encrypted on transport or on message level
- integrity: messages (or parts of messages) are signed on transport or on message level
- authentication: the sender of a message supplied authentication information on transport or on message level that allows the service to perform authentication

The following security policies are supported:

- **Security bindings**
  TransportBinding, SymmetricBinding and AsymmetricBinding, which specify the mechanism used to ensure confidentiality and integrity.
  - **TransportBinding**
    The message exchange is secured on transport level (HTTPS). As a prerequisite, the secure transport needs to be enabled and configured for the servlet engine that hosts the Web Services Stack service runtime.
  - **SymmetricBinding**
    The confidentiality of the message exchange is achieved on message level, using a symmetric encryption key that is shared between Web service client and service.

- **AsymmetricBinding**
  The confidentiality of the message exchange is achieved on message level using, an asymmetric encryption key (that is, client and service use different private/public key pairs for encryption and decryption).

- **Timestamps**
  A service can have a policy that requires that timestamps are added to messages.

- **Authentication**
  Policies can be defined that require messages exchanged contain authentication information such that receivers can authenticate the sender. The following authentication methods are supported:
  - HTTP basic authentication
  - client certificates for the HTTPS transport
  - user-name token contained in the message
  - digital signatures and X509 tokens contained in the message

Security Binding

Message exchange can be secured either on transport level or on message level. You can configure three different "bindings" for secure message exchange:

- **No Binding**
  Message exchange is not secured.

- **Transport Security with SSL**
  Message exchange is secured on transport level using HTTPS transport (SSL/TLS). To be able to configure transport security, the servlet engine must have HTTPS configured and enabled as a prerequisite. In addition, HTTPS must be configured for the Web Services Stack in the global configuration file axis2.xml. This is not configured by default. As an option you can specify whether a client certificate has to be provided on the transport.

- **Message-level Security with Symmetric Binding**
  Message exchange is secured using a symmetric key. Additional keystore configuration is required for symmetric binding, see Encryption/Signing. See *Encryption/Signing*.

- **Message level Security with Asymmetric Binding**
  Message exchange is secured using an asymmetric key. Additional keystore configuration is required for asymmetric binding see Encryption/Signing. See *Encryption/Signing*.

Message-level Security Options

- **Encrypt body**
  The message body must be encrypted.
- **Sign body**
  The message body must be signed.
- **Sign entire headers and body**
  The message headers and body must be signed

Token Assertions

- **Username Token**
  The Web service requires a username token in the message header.
- **Secure Conversation**
  The Web service provides secure communication over one or more messages.

Encrypt/Sign Message Part

- Xpath expressions can be specified to identify parts of a message that are signed and/or encrypted.

Security Model Configuration

- **User**
  The alias of the public key in the keystore that is used for encryption. For decryption, a private key is required. The password for accessing the private key is queried at runtime, using the *Password Callback Class*.
- **Password Callback Class**
  This is the name of a class that implements a password callback handler that is called by the Web Services Stack runtime to query a password for accessing a private key in the keystore for signing, or decrypting or a password for username token authentication. The password callback handler class implementation needs to be provided by the application writer. See *Password Callback Class*.

Encryption/Signing

- **Certificate Alias**
  The alias of the private key in the keystore that is used for signing outgoing messages. The alias name is also used as the username that is used for authentication. The password for accessing the private key is queried at runtime using the *Password Callback Class*. To verify a signature, a corresponding public key is used.

- **Keystore**
  The location of a Java keystore. This can be a relative path to a Java keystore contained in the Web service archive (Designer file with extension .aar), or an absolute path to a keystore located in the file system.
- **Keystore Password**
  The password required to access keys in the keystore.
- **Truststore**
  The location of a Java truststore. This can be a relative path to a Java truststore contained in the Web service archive (Designer file with extension .aar), or an absolute path to a truststore located in the file system.
- **Truststore Password**
  The password required to access keys in the truststore.

Other Security Options

- **Include Timestamp**
  The Web service requires a (signed) timestamp in the message header.
- **Use Client Certificate**

WS-ReliableMessaging

A WS-ReliableMessaging policy assertion can be defined for a service. This service then only accepts SOAP requests using the WS-ReliableMessaging protocol.

Example: WS-ReliableMessaging policy assertion

```
<wsp:Policy wsu:Id="ReliableMessaging" ↩
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
      ↩
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <wsp:ExactlyOne>
        <wsp:All>
<wsrm:RMAssertion  xmlns:wsrm=          ↩
"http://schemas.xmlsoap.org/ws/2005/02/rm/policy">
            <wsrm:InactivityTimeout Milliseconds="600000"/>
          </wsrm:RMAssertion>
        </wsp:All>
      </wsp:ExactlyOne>
    </wsp:Policy>
```

EntireX Settings Page

The EntireX Settings page allows you to specify EntireX specific settings of the Web Services archive (Designer file with extension .aar). The page contains two sections:

- Service Parameters
- Initialization Parameters for the Listener for XML/SOAP

Service Parameters

Under **Configuration**, a combo box is available with general settings for all XMM/SOAP mapping files in the Web service archive (Designer file with extension .aar); Specific settings for an XMM/SOAP mapping file supersede the general settings.

Parameter	Description	More Information
`Broker ID`	The broker ID to be used.	
`User ID`	The user ID for access to the broker.	***Using the Broker and RPC User ID/Password***
`Password`	The password for secured access to the broker.	
`Compression Level`	Sets the compression level.	*Using Compression* under *Writing Advanced Applications - EntireX Java ACI*
`Use Codepage`	Determines the translation processing of the broker. Valid values: `true|false|<character encoding>`. If a character encoding is set, this character encoding is used for the RPC message.	Method `setCharacterEncoding` of class `BrokerService` (EntireX Java ACI).
`Use Security`	Possible values: `true|false`.	
`Server Address`	This is the triplet of server class/server name/service.	
`RPC User ID`	The RPC user ID sent to the RPC server.	***Using the Broker and RPC User ID/Password***
`RPC Password`	The RPC password sent to the RPC server.	
`Natural Library`	By default the library name sent to the RPC server is retrieved from the IDL file (see `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation). The library name can be overwritten. This is useful if communicating with a Natural RPC server.	
`Natural Logon`	Enable to send RPC user ID/password pair. Possible values: `true|false`.	***Using the Broker and RPC User ID/Password***

Initialization Parameters for the Listener for XML/SOAP

Parameter	Description
Default Wait Time	Sets the value of the default wait time field to the argument (see `setDefaultWaittime` of class `BrokerService`).
Behavior of Non-conversation Calls	The parameter indicates whether a non-conversational call is finalized with a logoff call to free Broker resource (default), or by means of timeout. The default value for this parameter is "nonConv-with-logoff", which defines that a non-conversational call will finish with an additional logoff call (two calls per message). Set to "nonConv-without-logoff" to specify that a non-conversational call will finish without logoff call (one call per message); Broker will clean up resources by means of timeout.
User Name Token:	Use credentials retrieved from Username Token for EntireX server call.
Basic Authentication:	Use credentials retrieved from Basic Authentication for EntireX server call.

Using the Broker and RPC User ID/Password

EntireX supports two user ID/password pairs: a broker user ID/password pair and an (optional) RPC user ID/password pair sent from RPC clients to RPC servers. With EntireX Security, the broker user ID/password pair can be checked for authentication and authorization.

The RPC user ID/password pair is designed to be used by the receiving RPC server. This component's configuration determines whether the pair is considered or not. Useful scenarios are:

- Credentials for Natural Security
- Impersonation in the respective RPC Server documentation
- Web Service Transport Security with the RPC Server for XML/SOAP, see *XML Mapping Files*
- Service execution with client credentials for EntireX Adapter Listeners, see *Configuring Listeners*
- etc.

Sending the RPC user ID/password pair needs to be explicitly enabled by the RPC client. If it is enabled but no RPC user ID/password pair is provided, the broker user ID/password pair is inherited to the RPC user ID/password pair.

With the parameter `Natural Logon` (see ***Service Parameters***) sending the RPC user ID/password pair is enabled for the Web Services Wrapper. If you do so, we strongly recommend using SSL/TLS. See *SSL/TLS, HTTP(S), and Certificates with EntireX*.

To use the broker and RPC user ID/password

1 Specify a broker user ID with parameter `User ID`.

2 Optional. Specify broker password with parameter `Password`.

3 Enable to send the RPC user ID/password pair with parameter `Natural Logon` set to `true`.

4 If different user IDs and/or passwords are used for broker and RPC, use the parameters `RPC User ID` and `RPC Password` to provide a different RPC user ID/password pair.

5 By default the library name sent to the RPC server is retrieved from the IDL file (see `library-definition` under *Software AG IDL Grammar* in the IDL Editor documentation). The library name can be overwritten. This is useful if communicating with a Natural RPC server. Specify a library in parameter `Library`.

Password Callback Class

This section provides an example of a password callback handler.

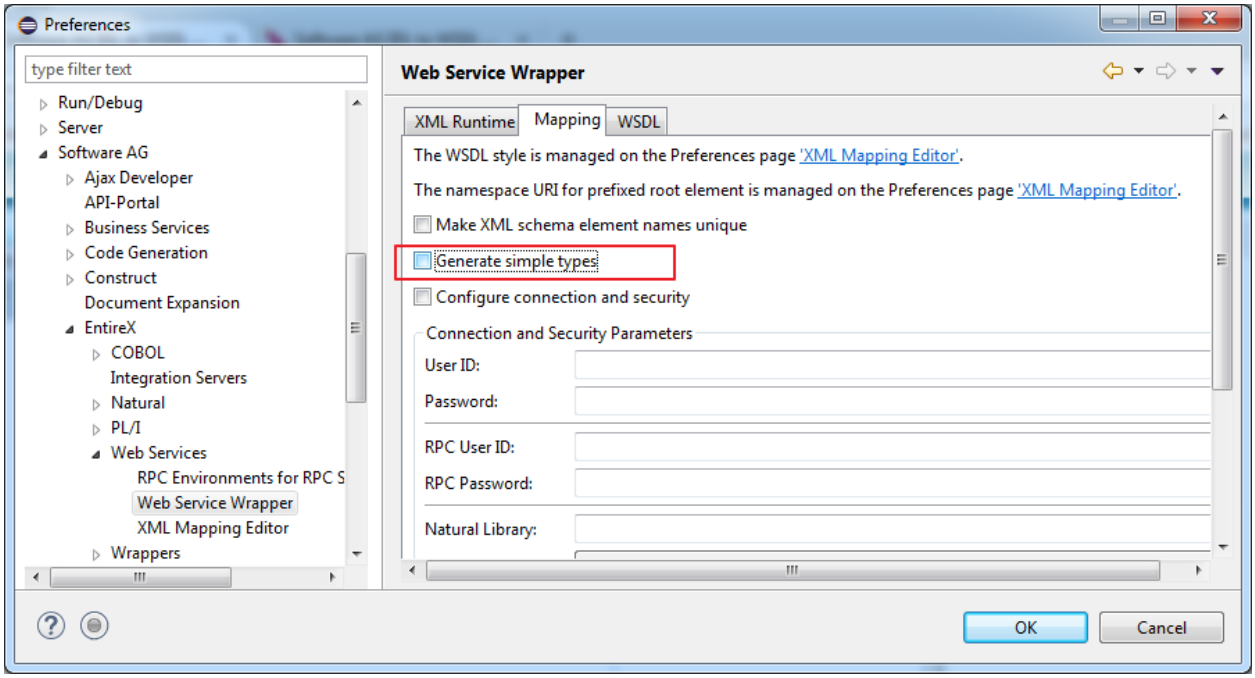
```
/*
/*
 * PasswordCallbackHandler.java -
 *     com.softwareag.wsstack.test.PasswordCallbackHandler class
 *
 *  Server/Client Password Callback Handler, responsible for delivering
 *  passwords for accessing a private signing or decryption key from a
 *  keystore or a password for a username token.
 */

package com.softwareag.wsstack.test;

import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import org.apache.ws.security.WSPasswordCallback;

public class PasswordCallbackHandler implements CallbackHandler
{
  /*
   * Handles all supported callbacks
   * @see javax.security.auth.callback.CallbackHandler#handle(
   *           javax.security.auth.callback.Callback[])
   */

  public void handle(Callback[] callbacks) throws IOException,
            UnsupportedCallbackException
  {
    try {
      for (int i = 0; i < callbacks.length; i++) {
        WSPasswordCallback pwcb = (WSPasswordCallback)callbacks[i];
        //get the type of the callback: SIGNATURE, DECRYPT, USERNAME_TOKEN
```

```
        int usage = pwcb.getUsage();
        String id = pwcb.getIdentifer();
        if (usage == WSPasswordCallback.SIGNATURE) {
          //supply password for signing key
          if ("client".equals(id))  pwcb.setPassword("apache"); else
          if ("service".equals(id)) pwcb.setPassword("apache");
        } else
        if (usage == WSPasswordCallback.DECRYPT) {
          //supply password for decryption key
          if ("client".equals(id))  pwcb.setPassword("apache"); else
          if ("service".equals(id)) pwcb.setPassword("apache");
        } else
        if (usage == WSPasswordCallback.USERNAME_TOKEN_UNKNOWN) {
          // verify username token on the server side
          if (id != null) {
            //get the password from the request
            String pass = pwcb.getPassword();
            // authenticate the user
            if (id.equals("client") && pass.equals("apache")) {
              return;
            } else {
              throw new UnsupportedCallbackException(callbacks[i],
                          "authentication failed");
            }
          }
        } else
        if (usage == WSPasswordCallback.USERNAME_TOKEN) {
          // supply password for username token on the client side
          if (id != null) {
            // supply the password
            String pass = pwcb.getPassword();
            if (pass == null) {
              if ("client".equals(id))  pwcb.setPassword("apache"); else
              if ("service".equals(id)) pwcb.setPassword("apache");
              pass = pwcb.getPassword();
            }
          }
        }
      } // for
    }
    catch (Throwable e) {
      throw new RuntimeException(e);
    }
    return;
  } // handle
}
```

6 Software AG IDL to WSDL Mapping

Mapping IDL Data Types to WSDL Data Types

The generation of WSDL depends on the option **Generate simple types** under **Preferences > Software AG > EntireX > Web Services > Web Service Wrapper > Mapping**.



- If **Generate simple types** is *checked*, the description is extended by `xsd:simpleType` definition if more detailed information such as length or format is available for an element.
- If this option is *not checked*, an element is represented with name and type; no further information is available.

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

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

IDL Data Type	Description	XMM	WSDL (Generate simple types Checked)	WSDL (Generate simple types not Checked)
A*number*	Alpha-numeric	string	`<xsd:element name="name">` `<xsd:simpleType>` `<xsd:restriction base="xsd:string">` `<xsd:maxLength value="number"/>` `</xsd:restriction>` `</xsd:simpleType>` `</xsd:element>`	`<xsd:element name="name" type="xsd:string"/>`
AV	Alpha-numeric variable length	string	`<xsd:element name="name" type="xsd:string"/>`	`<xsd:element name="name" type="xsd:string"/>`
AV[*number*]	Alpha-numeric variable length with maximum length	string	`<xsd:element name="name">` `<xsd:simpleType>` `<xsd:restriction base="xsd:string">` `<xsd:maxLength value=" number "/>` `</xsd:restriction>` `</xsd:simpleType>` `</xsd:element>`	`<xsd:element name="name" type="xsd:string"/>`
B*number*	Binary	binary	`<xsd:element name="name">` `<xsd:simpleType>` `<xsd:restriction base="xsd:base64Binary">` `<xsd:length value="base64Length"/>` `</xsd:restriction>` `</xsd:simpleType>` `</xsd:element>` **Note:** *base64Length* = 4 * rounded up(*number* /3)	`<xsd:element name="name" type="xsd:base64Binary"/>`
BV	Binary variable length	binary	`<xsd:element name="name" type="xsd:base64Binary"/>`	`<xsd:element name="name" type="xsd:base64Binary"/>`
BV[*number*]	Binary variable length with maximum length	binary	`<xsd:element name="name">` `<xsd:simpleType>` `<xsd:restriction base="xsd:base64Binary">` `<xsd:maxLength value="base64Length"/>` `</xsd:restriction>`	`<xsd:element name="name" type="xsd:base64Binary"/>`

IDL Data Type	Description	XMM	WSDL (Generate simple types Checked)	WSDL (Generate simple types not Checked)
			</xsd:simpleType> </xsd:element> **Note:** *base64Length* = 4 * rounded up(*number* /3)	
D	Date	date:yyyy-MM-dd	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:date"> <xsd:pattern ↩ value="[0-9]{4}- ↩ ((0[1-9])|(1[012]))-((0[1-9])| ([12][0-9])| (3[01]))"/> </xsd:restriction> </xsd:simpleType> </xsd:element>	<xsd:element name="name" type="xsd:date"/>
F4	Floating point (small)	float	<xsd:element name="*name*" type="xsd:float"/>	<xsd:element name="name" type="xsd:float"/>
F8	Floating point (large)	float	<xsd:element name="*name*" type="xsd:double"/>	<xsd:element name="name" type="xsd:double"/>
I1	Integer (small)	integer	<xsd:element name="*name*" type="xsd:byte"/>	<xsd:element name="name" type="xsd:byte"/>
I2	Integer (medium)	integer	<xsd:element name="*name*" type="xsd:short"/>	<xsd:element name="name" type="xsd:short"/>
I4	Integer (large)	integer	<xsd:element name="*name*" type="xsd:int"/>	<xsd:element name="name" type="xsd:int"/>
K*number*	Kanji	string	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:string"> <xsd:maxLength ↩ value="*number*"/> </xsd:restriction> </xsd:simpleType> </xsd:element>	<xsd:element name="name" type="xsd:string"/>

IDL Data Type	Description	XMM	WSDL (Generate simple types Checked)	WSDL (Generate simple types not Checked)
KV	Kanji variable length	string	`<xsd:element name="name" type="xsd:string"/>`	`<xsd:element name="name" type="xsd:string"/>`
KV[*number*]	Kanji variable length with maximum length	string	`<xsd:element name="name"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:string"> <xsd:maxLength ↩ value="number"/> </xsd:restriction> </xsd:simpleType> </xsd:element>`	`<xsd:element name="name" type="xsd:string"/>`
L	Logical	boolean	`<xsd:element name="name" type="xsd:boolean"/>`	`<xsd:element name="name" type="xsd:boolean"/>`
N*number1* [.*number2*]	Unpacked decimal	numeric	`<xsd:element name="name"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:decimal"> <xsd:totalDigits ↩ value="number1 + number2"/> <xsd:fractionDigits ↩ value="number2"/> </xsd:restriction> </xsd:simpleType> </xsd:element>` **Note:** default of *number2* is 0.	`<xsd:element name="name" type="xsd:decimal"/>`
NU*number1* [.*number2*]	Unpacked decimal unsigned	numeric	`<xsd:element name="name"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:decimal"> <xsd:totalDigits ↩ value="number1 + number2"/> <xsd:fractionDigits ↩ value="number2"/> </xsd:restriction> </xsd:simpleType> </xsd:element>` **Note:** default of *number2* is 0.	`<xsd:element name="name" type="xsd:decimal"/>`

IDL Data Type	Description	XMM	WSDL (Generate simple types Checked)	WSDL (Generate simple types not Checked)
P*number1*[.*number2*]	Packed decimal	numeric	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:decimal"> <xsd:totalDigits ↩ value="*number1* + *number2*"/> <xsd:fractionDigits ↩ value="*number2*"/> </xsd:restriction> </xsd:simpleType> </xsd:element> **Note:** default of *number2* is 0.	<xsd:element name="name" type="xsd:decimal"/>
PU*number1*[.*number2*]	Packed decimal unsigned	numeric	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:decimal"> <xsd:totalDigits ↩ value="*number1* + *number2*"/> <xsd:fractionDigits ↩ value="*number2*"/> </xsd:restriction> </xsd:simpleType> </xsd:element> **Note:** default of *number2* is 0.	<xsd:element name="name" type="xsd:decimal"/>
T	Time	dateTime :yyyy-MM-dd 'T'H:mm:ss	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:dateTime"> <xsd:pattern ↩ value="[0-9]{4}-((0[1-9])| (1[012]))-((0[1-9])| ↩ ([12][0-9])| (3[01]))T(([01][0-9])| (2[0-3]))(:[0-5][0-9]) ↩ {2}"/> </xsd:restriction> </xsd:simpleType> </xsd:element>	<xsd:element name="name" type="xsd:dateTime"/>

IDL Data Type	Description	XMM	WSDL (Generate simple types Checked)	WSDL (Generate simple types not Checked)
U*number*	Unicode	unicode	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:string"> <xsd:maxLength ↩ value="*number*"/> </xsd:restriction> </xsd:simpleType> </xsd:element>	<xsd:element name="name" type="xsd:string"/>
UV	Unicode variable length	unicode	<xsd:element name="*name*" type="xsd:string"/>	<xsd:element name="name" type="xsd:string"/>
UV*number*	Unicode variable length with maximum length	unicode	<xsd:element name="*name*"> <xsd:simpleType> <xsd:restriction ↩ base="xsd:string"> <xsd:maxLength ↩ value="*number*"/> </xsd:restriction> </xsd:simpleType> </xsd:element>	<xsd:element name="name" type="xsd:string"/>

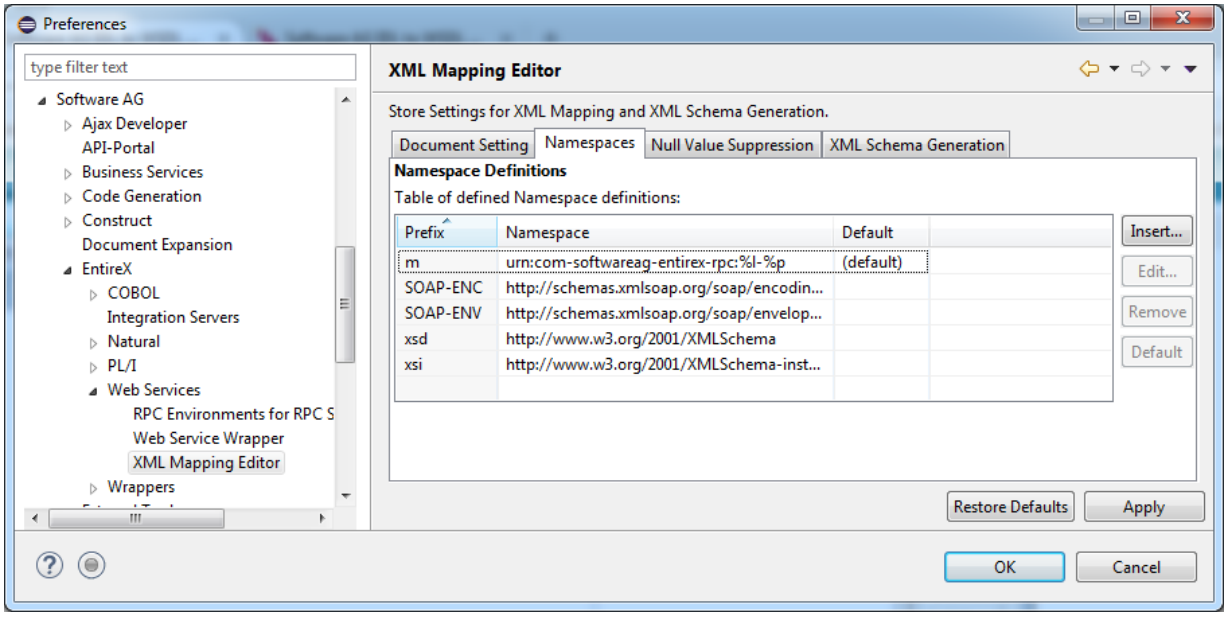
Default Namespace

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

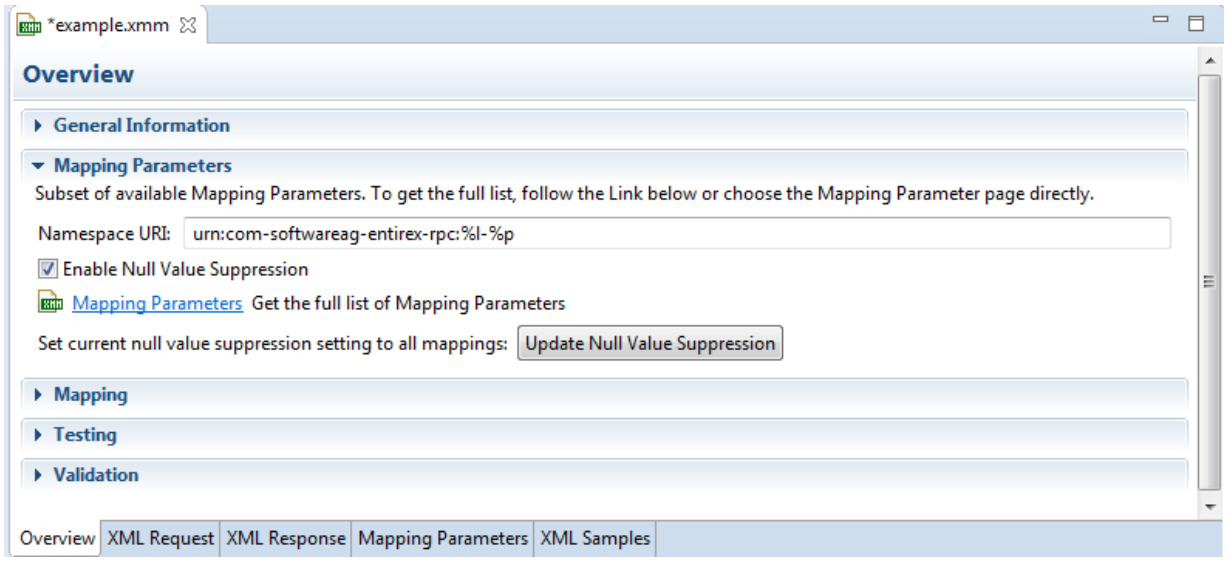
where `%l` is replaced by the IDL library name, and
`%p` is replaced by the IDL program name

If another namespace is required

- Change the setting on **Preferences** page of XML Mapping Editor.



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



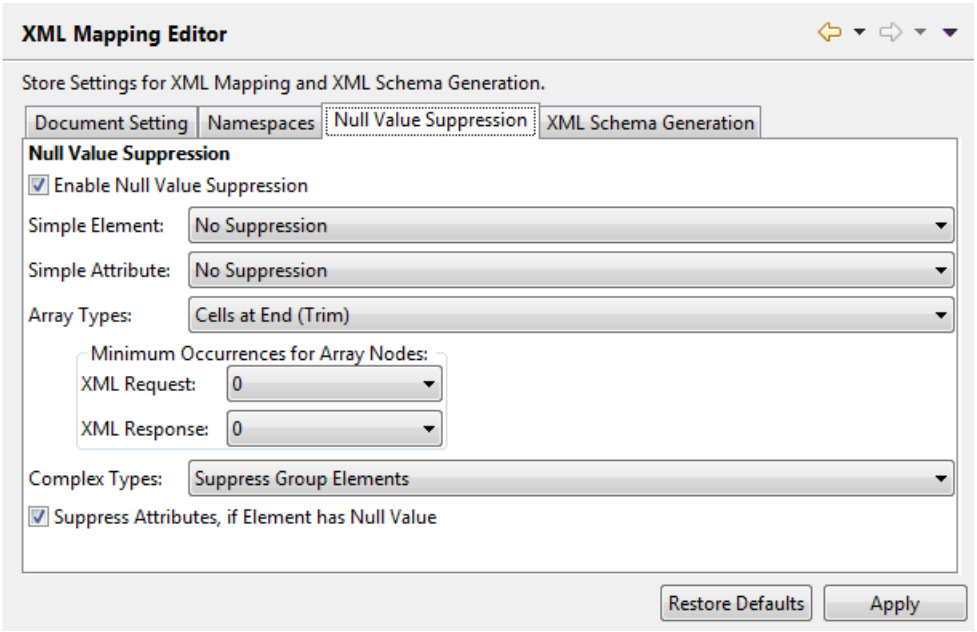
Min/Max Occurrence

minOccurs/maxOccurs in WSDL

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

minOccurs/maxOccurs for Arrays

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



Default Service Name

The default of service name is IDL file name. The service name can be changed in the Web Services Wrapper Wizard.