webMethods EntireX

EntireX Web Services Wrapper

Version 9.5 SP1

November 2013

This document applies to webMethods EntireX Version 9.5 SP1.

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

Document ID: EXX-EEXXWEBSERVICESWRAPPER-95SP1-20140628

Table of Contents

EntireX Web Services Wrapper

The EntireX Web Services Wrapper is a wizard that generates Web services from Software AG IDL, XML/SOAP mapping files or Natural subprogram files. The generated result is a Web service archive (.aar) that contains the relevant artifacts of the Web service such as an XML/SOAP mapping file (.xmm), WSDL file and additional configuration files. The Web service archive can be deployed for execution by the wizard or - in an extra deployment step - in a Web Services Stack with the EntireX XML/SOAP Listener runtime.

Introduction	Introduction to Web Services in EntireX.
Using	Using EntireX Web Services Wrapper.
Command-line Mode	Using Web Services Wrapper in Command-line mode.
Writing Applications	Writing applications with EntireX Web Services Wrapper.
Writing Advanced Applications	Writing advanced applications with EntireX Web Services Wrapper in combination with the Software AG Common Web Services Stack.

Related Literature

- *Software AG IDL Extractors*
- *IDL Editor*
- *EntireX XML Mapping Editor*
- *Administering the EntireX XML/SOAP RPC Server* in the UNIX and Windows administration documentation
- *EntireX RPC Servers*
- *RPC-ACI Bridge*

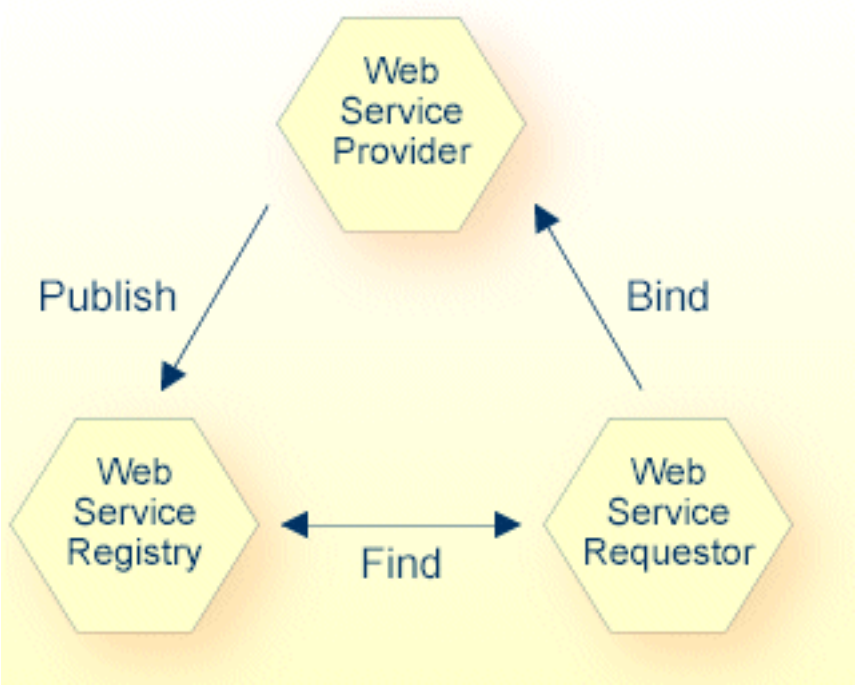
1 Introduction to Web Services in EntireX

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”. EntireX supports the creation of Web services from existing EntireX RPC servers.

Web Services

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:

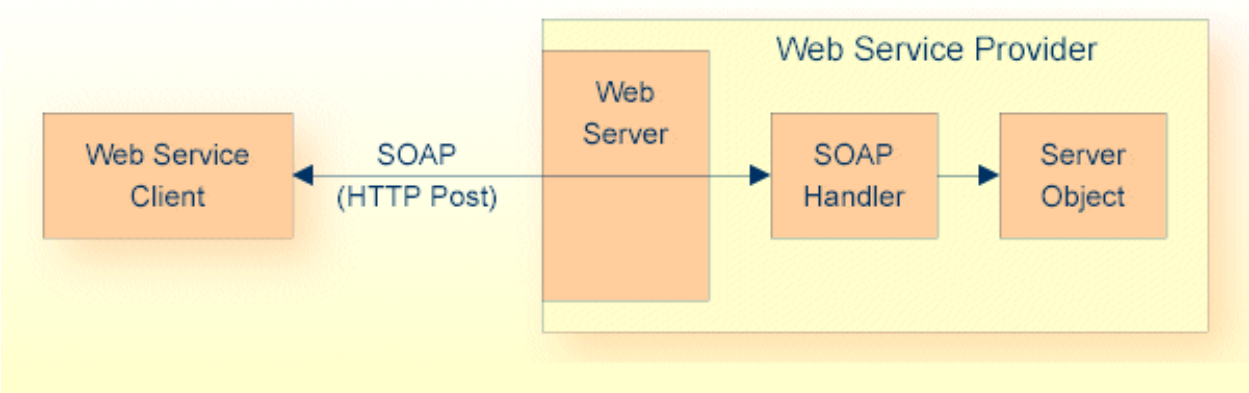
- HTTP and SMTP for basic network transport services,
- XML as data format,

- 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 Common Web Services Stack runtime

General SOAP Architecture

EntireX uses the Software AG Common Web Services Stack (WSS). 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.

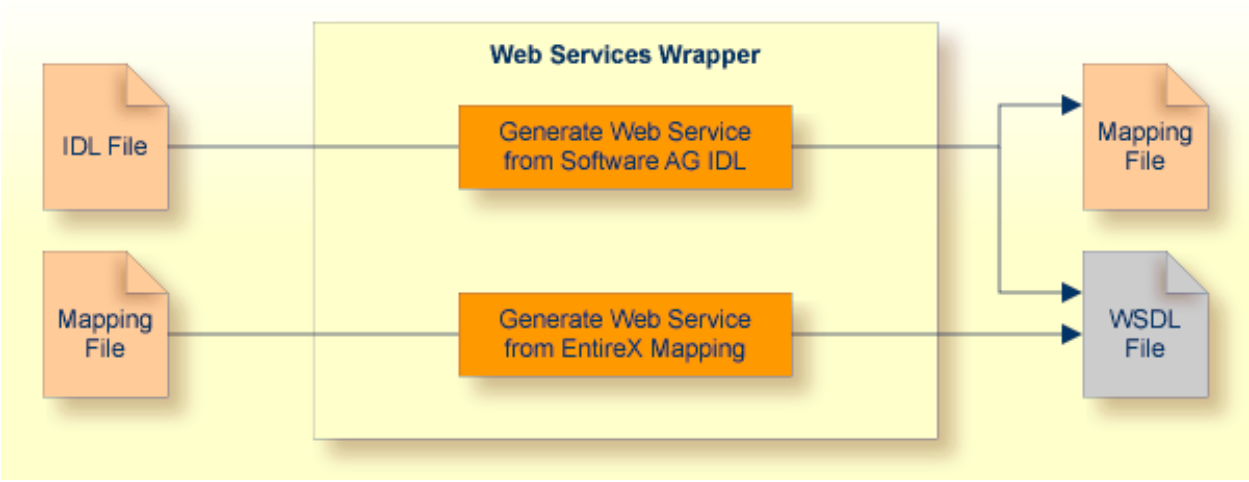
The *EntireX Workbench* provides functionality to create, configure, and deploy EntireX Web services. EntireX Web services are packaged into a service archive (extension .aar).

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.

2 Using the EntireX Web Services Wrapper

Introduction

The EntireX Web Services Wrapper is a wizard that generates Web services from Software AG IDL, XML/SOAP mapping files or Natural subprogram files. The generated result is a Web service archive (.aar) that contains the relevant artifacts of the Web service such as an XML/SOAP mapping file (.xmm), WSDL file and additional configuration files. The Web service archive can be deployed for execution by the wizard or - in an extra deployment step - in a Web Services Stack with the EntireX XML/SOAP Listener runtime.



The format of the generated descriptions is compliant with the Web Services Description Language (WSDL 1.1 - http://www.w3.org/TR/wsdl).

Note: 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).

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

Note also that some of the preferences of the XML Mapping Editor are applicable to Web service generation, in particular WSDL style and namespace definitions. See *Mapping Parameters* under *Using the XML Mapping Editor*.

Generate Web Services from Software AG IDL File

To generate a Web service

1 Select the IDL file to be processed. If there is a related *CVM File*, it is also used (internally).

2 From the context menu of the IDL file, choose **Properties**.

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

3 From the context menu of the IDL file, choose **Generate Web Service from Software AG IDL**.

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

5 If you deselect **Use defaults** for the **Configure EntireX service** section, you can select the following configuration items:

- If you select **general service parameters** (XML-INIT.xml), an additional configuration page will be appended. The parameters on this page are described in *Administering the EntireX XML/SOAP Listener* in the UNIX and Windows administration sections.
- If you select **Set connection and security parameters in mapping file**, an additional configuration page will appear by clicking **Next**.
- If you select **Send connection and security parameters with SOAP message**, an additional configuration page will be appended. The selected parameters will be generated into the SOAP header section of the generated WSDL file. A Web service client will then be able to set these parameters in the SOAP header of the SOAP message.

6 If you select **Deploy service**, an additional configuration page will be appended. See *Deploying EntireX Web Services* under *Writing Web Services Applications* for this dialog.

7 If you select **Register service to CentraSite**, a configuration page will be appended. See *CentraSite Integration* for this dialog.

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

9 Choose **Finish** to generate the Web service (mapping file, WSDL file and service archive).

Note: You can select more than one IDL file. All selected methods will be merged into one Web service. As result you will get multiple mapping files, one WSDL file and one service archive. Merging does not support the use of the same program name in different libraries.

3 Broker Command-line Utilities

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 the EntireX Workbench in 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 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 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.

4 Writing Web Services Applications

This chapter describes the typical steps required to create and register a Web service with the *EntireX Workbench* and how it is made available to Web service client applications.

See also Web Services Stack documentation, also available under webMethods Product Documentation on the **Software AG Documentation** website.

Generation of EntireX Web Services

The EntireX plug-ins for Eclipse provide support to expose EntireX services as Web services. We assume a typical scenario where an existing (legacy) server application has been "wrapped" with EntireX technology and is accessible to clients via the EntireX RPC protocol. The interface of the service is described by an IDL file (see *Software AG IDL File* in the IDL Editor documentation) and possibly a related client-side server mapping file (see *CVM File*) if such a file has been generated. Using the EntireX Web service plug-ins for Eclipse, application developers can generate from an IDL file (and optionally a CVM file), for example, *example.idl* (and *example.cvm*), the following:

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

These files are used by the EntireX XML/SOAP Runtime to implement a Web service interface for the given service.

From the context menu of your IDL file in the Package Explorer view, choose **Generate Web Service from Software AG IDL**.

Deploying EntireX Web Services

Requirements

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

- An application server where the Web Services Stack Web runtime is installed (wsstack.war). The Web Services Stack is typically accessible at the URL *http://<host-name>:<port-number>/wsstack/*. The default port number is 10010. This 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, also available under webMethods Product Documentation on the **Software AG Documentation** website.
- The EntireX XML/SOAP Runtime (packaged as Java library *entirex.jar*). File *entirex.jar* must be located in the *WEB-INF\lib* folder of the Web Services Stack Web application.
- The Eclipse plug-ins of the Web Services Stack must be installed.
- EntireX Broker and RPC server hosting the server implementation are up and running.

Deploying the Web Service

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

From the context menu of the generated service archive, choose **Software AG Common Web Services Stack >Deploy Web Service Package**. Now a wizard starts where you can select hostname, port number, and a servlet address of the Web Services Stack Deployment Servlet. You also need to authenticate with your credentials (user ID and password). On **Finish**, the service archive is sent to the selected deployment connection point.

Note: For more information, see *Deploying and Undeploying Web Service Archives* in section *Software AG Designer Plug-in* of the *Web Services Stack* documentation.

You can verify the deployment of your service with context-menu item **Software AG Common Web Services Stack >View Web Services Stack...** or **Software AG Common Web Services Stack >View Web Service**.

Note: After the installation of EntireX, only the default deployment connections are available in a new workspace. The default values are:
Hostname: localhost

Port number: 10010 (this number can be changed during installation)
Servlet address: /wsstack/sagdeployer

Choose **Windows > Preferences > Software AG > Web Services Stack > Deployment** to add, edit, or remove deployment connections.

Undeploying the Web Service

Undeploying a EntireX Web service means informing a running Web Services Stack Web application to remove a deployed service. The Web Services Stack Web application removes the corresponding service archive from the *WEB-INF/services* folder of the Web Services Stack Web application. You can verify the undeployment with the help of a browser. The undeployed service should disappear from the list of the deployed services (e.g. *http://localhost:10010/wsstack/services/listServices*).

Deploying Web Services Stack Runtime

This section covers the following topics:

- Deploying Web Services Stack Runtime to WebSphere 8.5
- Deploying Web Services Stack Runtime to WebLogic 12c

Deploying Web Services Stack Runtime to WebSphere 8.5

If you want to deploy the Web Services Stack Web application to WebSphere 8.5, perform the following steps:

To deploy to WebSphere 8.5

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.

Deploying Web Services Stack Runtime to WebLogic 12c

To deploy *wsstack.war* as Web application for WL server

1 Explode *wsstack.war* in a directory named `wsstack`.

2 Add a file named *weblogic.xml* to the folder *../wsstack/WEB-INF*. *weblogic.xml* has the following content:

```
<weblogic-web-app>
     <container-descriptor>
 <prefer-web-inf-classes>true</prefer-web-inf-classes>
     </container-descriptor>
</weblogic-web-app>
```

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

4 Open the weblogic console (e.g. *http://localhost:7001/console*.

5 Select **Deployments**.

6 Choose **Install**.

7 Select **wsstack(open directory)** as current location and choose **next**.

8 Select **Install this deployment as an application** and choose **next**.

9 Select **DD only** for **security** and **I will make the deployment accessible from the following location** for **Source accessibility**.

10 Enter the path to the *wsstack* folder and choose **next**.

11 Choose **Finish**.

To deploy `example.aar` in wsstack Web application

- As a prerequisite, `Basic Authentication` for your WL server has to be disabled. There are two ways to do this:
  - Edit *config.xml* of your WL server and add `<enforce-valid-basic-auth-credentials>false</enforce-valid-basic-auth-credentials>` under `<security-configuration>`.
  - Use the WebLogic scripting tool to modify the `enforce-valid-basic-auth-credentials` attribute.

```
connect('weblogic','weblogic','t3://localhost:7001')
edit()
startEdit()
ls()
cd('SecurityConfiguration')
ls()
cd('base_domain')
ls()
set('EnforceValidBasicAuthCredentials','false')
ls()
save()
activate()
print 'Now Restart Your Server...'
```

Developing Web Service Client Applications

Once the EntireX Web service is up and running and its WSDL is accessible (via HTTP), Web service client applications can be developed. For example, in XMLSpy, in the menu bar choose **SOAP > Create new SOAP request** and enter the URL of the service WSDL (e.g. *http://localhost:10010/wsstack/services/example?wsdl*). XMLSpy reads the WSDL and displays a list of methods. Select one, e.g. CALC, and fill in parameter values in the displayed SOAP message template. Finally select **SOAP > Send request to server** to execute the Web service call.

Testing EntireX Web Services

From the context menu of the generated service archive, choose **Test EntireX Web Service**. This starts the XML Tester.

Note: The XML Tester must already be configured for the generated and deployed service.

If the archive contains more than one XMM file describing the service, select the XMM file you want. Follow the instructions for the XML Tester to create a sample document.

See *XML Tester* in the XML/SOAP Wrapper documentation.

Removing Web Services

When a Web service is removed from the 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	■ IDL file ■ XMM file ■ CVM file (if applicable) ■ WSDL file
IDL File	■ XMM file ■ WSDL file
XMM File	■ WSDL file (if applicable)

5 Writing Advanced Web Services Applications

See also *Writing Web Service Client Applications* in the IDL Extractor for WSDL documentation.

Supported Features

EntireX version 8.1 and above supports a number of advanced Web services features in combination with the Web Services Stack. This includes support for

- SOAP 1.2 messaging
- SOAP 1.2 binding in WSDL 1.1
- multiple transports (HTTP, HTTPS, TCP, JMS)
- WS-Policy (WS-Addressing, WS-Security, WS-ReliableMessaging)
- WS-Policy Attachment to WSDL 1.1

SOAP 1.2

Web services created with the EntireX Workbench support by default SOAP 1.2 (*http://www.w3.org/TR/soap12-part1/*) in addition to SOAP 1.1. No extra configuration is needed for this.

WSDL 1.1 descriptions generated for EntireX services with the EntireX Workbench 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-Stack also supports the Representational State Transfer (REST) style of messaging.

WSDL Query

The WSDL of an EntireX service that has been generated, configured and deployed in a Web Services Stack runtime running in a servlet engine can be retrieved using the service URI appended with "?wsdl".

Example: *http://host:port/wsstack/service/myService?wsdl*

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

Transports

Services can be configured to be accessible over multiple transport protocols. The default transport is HTTP. Using the Configuration Editor, you can configure different transports via which the service can be accessed.

- HTTPS: This requires that HTTPS is configured for the servlet engine that is running the Web Services Stack service runtime.
- TCP: Additional configuration of the Web Services Stack runtime in *axis2.xml* is necessary to enable support of this transport. See the separate Web Services Stack documentation for more information.
- JMS: Additional configuration of the Web Services Stack runtime in *axis2.xml* is necessary to enable support of this transport. See the separate Web Services Stack documentation for more information.

Policies

For services created with EntireX and Web Services Stack, additional policies can be defined per service. These include WS-Addressing, WS-Security and WS-ReliableMessaging.

WS-Addressing

A WS-Addressing policy assertion can be defined for a service to accept SOAP messages containing a WS-Addressing SOAP header.

Example: WS-Addressing policy assertion

```
<wsp:Policy wsu:Id="Addressing"
  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>
      <wsaws:UsingAddressing
        xmlns:wsaws="http://www.w3.org/2006/05/addressing/wsdl"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

WS-Addressing can be configured for a service using the Configuration Editor.

WS-Security

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

WS-Security can be configured for a service using the Web Services Stack configuration editor.

The following security policies are supported:

- Security bindings: TransportBinding, SymmetricBinding and AsymmetricBinding, which specify by which mechanism confidentiality and integrity are ensured.
  - TransportBinding: specifies that 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: specifies that 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: specifies that 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

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>
```

Configuring Web Services

Introduction

The global configuration for the Web services engine is done in the configuration file *axis2.xml*. See the separate Web Services Stack documentation for more on configuring the Web Services Stack engine. For the services runtime, this configuration file is located in the Web Services Stack Web application's configuration directory *<servlet_engine_root>\webapps\wsstack\WEB-INF\conf*. The default location of the configuration folder of the Software AG Web Server is `<SuiteInstallDir>`/profiles/CTP/configuration.

Individual services or services group are configured in the *services.xml* file that is part of a services archive. The Web Services Stack configuration editor provides an Eclipse user interface to configure a service. Open a Web service archive (.aar) that was generated with the EntireX Workbench with the Web Services Stack configuration editor. There are five views on different aspects of the service:

- The **Archive** view displays the contents of a service archive and allows you to add additional files to the archive or remove files from the archive. You can add additional Web service files (*.idl, *.xmm) to the EntireX service. If an XMM file is selected, the mapping of the file must match the mapping of the service.

  If an IDL file is selected and a corresponding XMM file is available in the project, you are prompted to

  - overwrite the existing mapping file on the basis of the IDL file, or
  - use the existing mapping file.
- The **Service** view allows you to view and modify various settings that apply to a service contained in the archive.
- The **Operations** view allows you to view and modify settings that apply to an operation of a service in the archive and corresponds to the Service view.
- File *services.xml* allows you to view the services archive's configuration file in clear text (XML format).
- Configuration parameters for the XML/SOAP Listener; see ***EntireX Settings View***.

See the separate Web Services Stack documentation for more information on the Configuration Editor.

Services Configuration View

EntireX Web services have some specific configurations that are defined by the Web Services wizard of the EntireX Workbench. The ServiceLifeCycleClass, the EntireXMessageReceiver and the session scope Application. You should not modify these settings for EntireX Web services.

- WS-Addressing Configuration
- WS-Security Configuration
- Security Bindings
- Keystore Configuration
- Additional Security Options

WS-Addressing Configuration

To enable WS-Addressing headers for a service, check the **Enable WS-Addressing** check box. This inserts a WS-Addressing policy into *services.xml* and enables the addressing module of the Web Services Stack engine 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 Configuration

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 service, check the **Enable WS-Security** check box, which then displays additional configuration options. Message exchange can be secured either on transport level or on message level. You can configure three different "bindings" for secure message exchange.

Security Bindings

- Transport Security with SSL: message exchange is secured on transport level using SSL (HTTPS transport). 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.
- Message level security with asymmetric binding: Message exchange is secured using an asymmetric key. Additional keystore configuration is required for asymmetric binding.

Keystore Configuration

- Location: the location of a Java keystore. This can be a relative path to a Java keystore contained in the service archive, or an absolute path to a keystore located in the file system.
- Keystore Password: the password required to access keys in the keystore.
- 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 Handler (see below). To verify a signature, a corresponding public key is used.
- Encryption 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 Handler (see below).
- 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 singing or decrypting or a password for username token authentication. The password callback handler class implementation needs to be provided by the application writer.

Additional Security Options

- Username Token authentication: the services requires a username token in the message header.
- Include timestamp: the service requires a (signed) timestamp in the message header.
- Sign header: the message header must be signed
- Sign body: the message body must be signed
- Encrypt body: the message body must be encrypted
- Encrypt/sign message part: Xpath expressions can be specified to identify parts of a message that are signed and/or encrypted.

Example:

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) {
            // suppy 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
}
}
```

EntireX Settings View

The **EntireX Settings** view allows you to modify file *xml-init.xml*, which is part of the Web Services archive. The view contains two sections:

- EntireX Service Parameters

- XML/SOAP Listener Initialization Parameters

EntireX Service Parameters

The service section contains a combo box with one entry for the general settings and one entry for each XMM file describing the service. The general settings are for all XMM files in the archive; special settings for an XMM file supersede the general settings for this file.

Parameter	Description
Broker ID	The broker to be used.
User ID	The user ID used for calling the broker.
Password	The password used for calling the broker.
Encryption Level	Possible values: 0|1|2. See `ENCRYPTION-LEVEL`, class `Broker` in the Javadoc documentation of the Java ACI and method `BrokerSecurity` in the Javadoc documentation of the Java ACI.
Compression Level	Sets the compression level. See *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 RPC message. See method `useCodePage` and `setCharacterEncoding` in the documentation on class `BrokerService` in the Javadoc documentation of the Java ACI.
Use security	Possible values: true|false. To use EntireX Security. See *EntireX Security for EntireX Broker* in the EntireX Security documentation.
Server Address	This is the triplet of server class/server name/service.
RPC User ID	The RPC user ID specified here is used for EntireX Security.
RPC Password	The RPC Password specified here is used for EntireX Security.
Natural Library	The Natural library. Works only if `exx-natural-security` is true. See *Using Natural Security* in the Java Wrapper documentation.
Natural Logon	Possible values: true|false. To use Natural Security. See *Using Natural Security* in the Java Wrapper documentation.

XML/SOAP Listener Initialization Parameters

Parameter	Description
Default Wait Time	Sets the value of the default wait time field to the argument (see `setDefaultWaittime` of class `BrokerService` in the Javadoc documentation of the Java ACI).
Servlet Internal Sweep Time	Interval in which the servlet checks and frees unused resources. The default is 60 seconds.
Enable Character Reference	Enable/disable the character reference for the XML payload.

Parameter	Description
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.