Writing Advanced Web Services Applications

This chapter covers the following topics:

- Supported Features
- SOAP 1.2
- WSDL Query
- Transports
- Policies
- WS-ReliableMessaging
- Configuring Web Services

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
- Services Configuration View
- EntireX Settings View

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 based on Apache Tomcat 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) {
            // 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
}
}
```

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` and method `setSecurity`.
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` (EntireX Java ACI).
Use security	Possible values: true|false. To use EntireX Security. See *EntireX Security for EntireX Broker*.
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`).
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.
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.