Oracle® Fusion Middleware

Extensibility Guide for Oracle Web Services Manager

11*g* Release 1 (11.1.1)

E13882-01

January 2011

This document describes how to develop custom assertions.

ORACLE®

Oracle Fusion Middleware Extensibility Guide for Oracle Web Services Manager 11*g* Release 1 (11.1.1)

E13882-01

Contents

4   Implementing Advanced Features in Custom Assertions

A   Schema Reference for Custom Assertions

Preface

This preface describes the document accessibility features and conventions used in this guide—*Extensibility Guide for Oracle Web Services Manager*.

Intended Audience

This manual is intended for software developers who are interested in creating custom assertions for Web services.

How to Use This Guide

This document describes the following:

About Policies and Assertions

- Learn about predefined policies and assertions

  For more information, see "Predefined Policies and Assertions" on page 1-1 and "About Custom Assertions" on page 1-2.

- Learn about parts and types of custom assertions

  For more information, see "Policies, Assertions, Expressions, and Operators" on page 1-2 and "Types of Custom Assertions" on page 1-5.

- Review the rules and restrictions about categories and bindings

  For more information, see"Understanding Assertion Categories" on page 1-3 and "Rules for Binding of Custom Assertions" on page 1-4.

- Review the lifecycle of a custom assertion

  For more information, see"Life Cycle of a Custom Assertion" on page 1-4.

- Understand the types of custom assertions

  For more information, see"Types of Custom Assertions" on page 1-5.

Creation of Custom Assertions

- Review the naming conventions for policies and assertions

  For more information, see "Naming Conventions for Policies and Assertions" on page 2-1.

- Create the custom assertions

  For more information, see "Developing Custom Assertions for Web Service" on page 2-1.

- Test the custom assertion

  For more information, see "Testing the Web Service" on page 2-7.

- Generate a client policy from a Web service policy created using custom assertions

  For more information, see "Creating a Custom Client Policy with Custom Assertions" on page 2-7.

Code Examples

- "IP Address Validation Custom Assertion Sample" on page 3-1

- "Encryption and Decryption Custom Assertion Sample" on page 3-6

- "Authentication Custom Assertion Sample" on page 3-21

Advanced Topics

- Implement advanced features:

  - input parameters to the assertion

  - access Oracle WSM context properties

  - access request, response, and fault message objects

  - access specific parts of a message based on XPath

  - access certificate used by container for SSL

  - access transport properties such as HTTP requests and responses

  - access CSF keys

  For more information, see "Implementing Advanced Features in Custom Assertions" on page 4-1.

- Handling exceptions

  For more information, see "Handling Exceptions in Custom Assertions" on page 4-7.

References

- XML schema for creating custom assertions

  For more information, see "Schema Reference for Custom Assertions" on page A-1.

- Java API reference for Oracle WSM

  For more information, see Java API Reference for Oracle Web Services Manager.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Related Documents

For more information, see the following manuals:

- *Oracle Fusion Middleware Release Notes for Linux x86*

- *Oracle Fusion Middleware Release Notes for Microsoft Windows*

- Oracle Fusion Middleware Security and Administrator's Guide

- Oracle Fusion Middleware Library on OTN

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Understanding Policies and Assertions

Oracle Web Services Manager (Oracle WSM) delivers a set of predefined policies and assertion templates that are automatically available when you install Oracle Fusion Middleware. You may need to develop custom assertions when specific functionality is not provided with the standard policies that come with the product. For example, an application may require the use of a type of authentication that is not available in Oracle WSM. In this situation, a custom authentication assertion needs to be created. This manual provides information on how to develop, deploy, and test custom assertions.

This chapter contains the following topics:

- Predefined Policies and Assertions
- Creating Policies
- About Custom Assertions

1.1 Predefined Policies and Assertions

Oracle Fusion Middleware 11g Release 1 (11.1.1) is packaged with predefined policies. For information on the Oracle Fusion Middleware Web Services policies, see Appendix B, "Predefined Policies" in *Security and Administrator's Guide for Web Services*.

Each policy consists of one or more assertions, defined at the domain-level, that define the security requirements. A set of predefined assertions are provided out-of-the-box. The predefined assertions are described in Appendix C, "Predefined Assertion Templates" in *Security and Administrator's Guide for Web Services*.

For more information on using the existing predefined assertions, see "Working with Assertions" in *Security and Administrator's Guide for Web Services*. In addition to using existing assertions, you can develop your own custom assertions using the APIs for Oracle WSM.

1.2 Creating Policies

You can create a Web service policy in one of the following ways:

- Create a new policy using existing assertion templates
- Create a new policy from an existing policy
- Import a policy from a file
- Create policies from custom assertions

The first three methods of creating policies are described in "Managing Web Service Policies" in the *Security and Administrator's Guide for Web Services*. This guide describes how to create policies using custom assertions that you develop.

1.3 About Custom Assertions

This section describes the following topics:

- Section 1.3.1, "Policies, Assertions, Expressions, and Operators"
- Section 1.3.2, "Understanding Assertion Categories"
- Section 1.3.3, "Rules for Binding of Custom Assertions"
- Section 1.3.4, "Life Cycle of a Custom Assertion"
- Section 1.3.5, "Types of Custom Assertions"

1.3.1 Policies, Assertions, Expressions, and Operators

Web services use policies to describe their capabilities and requirements. Policies define how a message must be secured and delivered reliably, whether a message must flow a transaction, and so on. A policy is a set of assertions (rules, requirements, obligations) that express specific policy requirements or properties of a Web service.

A policy assertion is a basic unit representing an individual requirement, capability or property in a policy. Assertions use domain-specific semantics to enable interoperability.

A policy expression is an XML representation of a policy. The policy expression consists of various logical combinations of the basic policy assertions that form the content of the wsp:Policy element. The logical combinations are created using policy operators (wsp:Policy, wsp:All, and wsp:ExactlyOne elements).

Example 1–1 shows a policy (defined by the wsp:Policy element) with one assertion (orasp:AssertionOne). For more information on the elements and Oracle extensions of their attributes, see the Appendix A, "Schema Reference for Custom Assertions".

Example 1–1   Policy with One Assertion

```
<wsp:Policy
    xmlns="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:orasp="http://schemas.oracle.com/ws/2006/01/securitypolicy" >

         <orasp:AssertionOne orawsp:Silent="true" orawsp:Enforced="true"
         orawsp:name="Validator"
         orawsp:category="security/authentication">
         ...
         </orasp:AssertionOne>
</wsp:Policy>
```

Example 1–2 shows a list of policy assertions wrapped by the policy operator wsp:All element (all of the policy assertions in the list must evaluate to true). The wsp:Policy element is semantically equivalent to wsp:All.

Example 1–2   Policy with Two Assertions and wsp:All Operator

```
<wsp:Policy
    xmlns="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:orasp="http://schemas.oracle.com/ws/2006/01/securitypolicy" >
```

```
            <wsp:All>
             <orasp:AssertionOne orawsp:Silent="true" orawsp:Enforced="true"
             orawsp:name="SAMLValidator"
             orawsp:category="security/authentication">
             ...
            </orasp:AssertionOne>
            <orasp:AssertionTwo orawsp:Silent="true" orawsp:Enforced="true"
             orawsp:name="UserNameValidator"
             orawsp:category="security/authentication">
             ...
            </orasp:AssertionTwo>
           </wsp:All>

</wsp:Policy>
```

In Example 1–3, each line within the operators <wsp:All>...</wsp:All> represents a valid policy alternative. The policy is satisfied if one set of the policy alternatives is true.

Example 1–3   Policy with Many Assertions and wsp:ExactlyOne Operator

```
<wsp:Policy
    xmlns="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:orasp="http://schemas.oracle.com/ws/2006/01/securitypolicy" >

     <wsp:ExactlyOne>
        <wsp:All>
         <orasp:SAML orawsp:Silent="true" orawsp:Enforced="true"
         orawsp:name="SAML"
         orawsp:category="security/authentication">
         ...
        </orasp:SAML>
        </wsp:All>
        <wsp:All>
        <orasp:Username orawsp:Silent="true" orawsp:Enforced="true"
         orawsp:name="Username"
         orawsp:category="security/authentication">
         ...
         </orasp:Username>
        </wsp:All>

    </wsp:ExactlyOne>
</wsp:Policy>
```

Using the operators and combinations of policy assertions, you can create complex policy alternatives.

1.3.2 Understanding Assertion Categories

Table 1–1 provides a list of the assertion categories based on the Web service specification they conform to and their significance. You specify the category using the orawsp:category, an attribute of the orasp:Assertion element. Chapter 3, "Stepping Through Sample Custom Assertions" provides samples for various categories of security custom assertions.

Table 1–1   Assertion Categories

Category	Description	Valid Values for orawsp:category
Security	Policies that implement the WS-Security 1.0 and 1.1 standards, which allow the communication of various security token formats, such as SAML, Kerberos, and X.509. They enforce authentication and authorization of users, identity propagation, and message protection.	security/authentication, security/msg-protection, security/authorization, security/logging
MTOM Attachments	Policies with attachments that are in MTOM format. Message Transmission Optimization Mechanism (MTOM) format enables efficient transfer of binary data to be sent to and from Web services.	mtom
Reliable Messaging	Policies that support the WS-ReliableMessaging protocol. This guarantees the delivery of SOAP messages. WS-ReliableMessaging makes use of a reliable messaging source and a reliable messaging destination and supports a number of Delivery Assurances. For more information, see WS-ReliableMessaging specification at: http://docs.oasis-open.org/ws-rx/wsrm/200702	wsrm
WS-Addressing	Policies in which simple object access protocol (SOAP) messages include WS-Addressing headers and that allow Web services to communicate addressing information.	addressing
Management	Policies used to store request, response, and fault messages to a message log.	management

Using Enterprise Manager Fusion Middlware Control, you can view, edit, and manage policies of different categories. Note that policies of a category can be attached to specific endpoints. See the next section for rules for binding custom assertions.

1.3.3 Rules for Binding of Custom Assertions

For a description of the <oraswsp:bindings> element used to define the bindings in the assertion, and all its subelements, see oraswsp:bindings in *Security and Administrator's Guide for Web Services*.

1.3.4 Life Cycle of a Custom Assertion

The life cycle of a custom assertion involves four phases of initialization, execution, post-execution and termination. The Oracle WSM run time manages the life cycle of custom assertion. For each phase, the Oracle WSM run time invokes a method of custom assertion executor. The following methods must be implemented inside the custom assertion executor of the custom policy:

1. void init(IAssertion ia, IExecutionContext context, IContext msgContext): This method is called after the AssertionExecutor is created or after one of the properties has been changed.

2. IResult execute(IContext mcontext): This method is invoked at stages such as request, response, and in the event of a fault. This method must always return a non-null IResult object. To find the stage of policy execution, see Section 4.3, "Accessing Request, Response, and Fault Message Objects".

3. IResult postExecute(IContext messageContext): Executes any task required after policy execution.

4. void destroy(): The destroy method is invoked by Oracle WSM run time when application is shutting down. Its invoked for a cleaner ending of the assertion lifecycle.

1.3.5 Types of Custom Assertions

The policy object model defines two types of assertions:

- "Simple Assertions" on page 1-5

- "Multi-Element Simple Assertions" on page 1-5

1.3.5.1 Simple Assertions

A Simple Assertion contains only one assertion (defining a single behavior) and cannot contain other assertions. A Simple Assertion maps to an org.w3c.dom.Element that does not have any nested elements except for extensions defined by Oracle. The class for the Simple Assertion extends the class oracle.wsm.policy.model.ISimpleAssertion. ISimpleOracleAssertion provides the extensions defined by Oracle for a WS-Policy Assertion.

Usage

Simple Assertions:

- Cannot contain nested elements other than Oracle extension element <orawsp:bindings>

- Cannot contain other assertions

- Use the default Serializer and De-Serializer

- Use the default Implementation class

- Must extend the base class SimpleAssertion. The method getAssertionType must return the appropriate value if you introduce a new Class.

- Do not need Serializer and De-Serializer if you introduce a new Class.

Pseudo-schema

The pseudo-schema for Simple Assertions with only binding elements is as follows:

```
<Assertion>
[ wsp:Optional="xsd:boolean" ]?
[ orawsp:Silent="xsd:boolean" ]?
[ orawsp:Enforced="xsd:boolean" ]?
[ orawsp:description="xsd:string" ]?
[ orawsp:category="xsd:string" ]?...>
<orawsp:bindings>?
</Assertion>
```

1.3.5.2 Multi-Element Simple Assertions

A Multi-element Simple Assertion cannot contain other assertions; however, it may contain nested XML elements. A Multi-element Simple Assertion maps to an

org.w3c.dom.Element which has nested XML elements and extension elements defined by Oracle. The class for the Multi-element Simple Assertion extends the class oracle.wsm.policy.model.IMultiElementSimpleAssertion.

Usage

Multi-element Simple Assertions:

- May contain nested XML elements other than Oracle extension element <orawsp:bindings>

- Cannot contain other assertions and are defined in domain-specific specifications. Nested XML elements participate in policy intersections only if domain-specific intersection semantics are defined.

Pseudo-schema

Multi-element Simple Assertion may contain elements other than the bindings. The psuedo-schema is:

```
<Assertion>
[ wsp:Optional="xsd:boolean" ]?
[ orawsp:Silent="xsd:boolean" ]?
[ orawsp:Enforced="xsd:boolean" ]?
[ orawsp:description="xsd:string" ]?
[ orawsp:category="xsd:string" ]?...>
<other-xml-elements>+
<orawsp:bindings>?
</Assertion>
```

2

Creating Custom Assertions

This chapter describes how to develop custom assertions. It includes the following sections:

- Naming Conventions for Policies and Assertions
- Developing Custom Assertions for Web Service
- Testing the Web Service
- Creating a Custom Client Policy with Custom Assertions

2.1 Naming Conventions for Policies and Assertions

The policy name is specified by the name attribute of the policy content. The policy name must not already exist in the Policy Store. Once you import the policy to the Policy Store, you cannot edit the name of a policy. To change the policy name, you will need to copy the policy and assign it a different name.

Oracle recommends that you follow the policy naming conventions described in "Recommended Naming Conventions for Policies" in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services.* The same conventions are used to name assertions.

2.2 Developing Custom Assertions for Web Service

To develop a custom assertion, you need to create the following files:

- Custom assertion executor—Implements the Java class and its parsing and enforcement logic.
- Custom policy file—Enables you to define the bindings for and configure the custom assertion.
- policy-config.xml file—Registers the custom policy file.

You can specify the custom assertion executor in one of the following files:

- Custom policy file
- policy-config.xml file

You package the custom assertion executor and policy-config.xml file as a JAR file and make the JAR file available in the CLASSPATH for your domain. Then, you import the custom policy file and attach it to your Web service or client, as required.

The following sections describe each step in the process.

2.2.1 Step 1: Create the Custom Assertion Executor

Create the custom assertion executor to execute and validate the logic of your policy assertion. The custom assertion executor must extend `oracle.wsm.policyengine.impl.AssertionExecutor`.

When building the custom assertion executor, ensure that the following JAR files are in your CLASSPATH: wsm-policy-core.jar, wsm-agent-core.jar, and oracle.logging-utils_11.1.1.jar (located at `oracle_common/modules/oracle.wsm.common_11.1.1`, `oracle_common/modules/oracle.wsm.agent.common_11.1.1`, and `oracle_common/module` respectively). Add the files to the classpath.

Example 2–1 is an example of a custom assertion executor that can be used to validate the IP address of the request to the Web service. If the IP address of the request is invalid, a `FAULT_FAILED_CHECK` exception is thrown.

For more information about the APIs that are available to you for developing your own custom assertion executor, see the Java API Reference for Oracle Web Services Manager.

Example 2–1   Example Custom Assertion Executor

```
package sampleassertion;

import oracle.wsm.common.sdk.IContext;
import oracle.wsm.common.sdk.IMessageContext;
import oracle.wsm.common.sdk.IResult;
import oracle.wsm.common.sdk.Result;
import oracle.wsm.common.sdk.WSMException;
import oracle.wsm.policy.model.IAssertionBindings;
import oracle.wsm.policy.model.IConfig;
import oracle.wsm.policy.model.IPropertySet;
import oracle.wsm.policy.model.ISimpleOracleAssertion;
import oracle.wsm.policy.model.impl.SimpleAssertion;
import oracle.wsm.policyengine.impl.AssertionExecutor;

public class IpAssertionExecutor extends AssertionExecutor {
    public IpAssertionExecutor() {
    }
    public void destroy() {
    }

    public void init(oracle.wsm.policy.model.IAssertion assertion,
                     oracle.wsm.policyengine.IExecutionContext econtext,
                     oracle.wsm.common.sdk.IContext context) {
        this.assertion = assertion;
        this.econtext = econtext;
    }
    public oracle.wsm.policyengine.IExecutionContext getExecutionContext() {
        return this.econtext;
    }
    public boolean isAssertionEnabled() {
        return ((ISimpleOracleAssertion)this.assertion).isEnforced();
    }
    public String getAssertionName() {
        return this.assertion.getQName().toString();
    }

    /**
     * @param context
     * @return
     */
    public IResult execute(IContext context) throws WSMException {
```

```
        try {
            IAssertionBindings bindings =
                ((SimpleAssertion)(this.assertion)).getBindings();
            IConfig config = bindings.getConfigs().get(0);
            IPropertySet propertyset = config.getPropertySets().get(0);
            String valid_ips =
                propertyset.getPropertyByName("valid_ips").getValue();
            String ipAddr = ((IMessageContext)context).getRemoteAddr();
            IResult result = new Result();
            if (valid_ips != null && valid_ips.trim().length() > 0) {
                String[] valid_ips_array = valid_ips.split(",");
                boolean isPresent = false;
                for (String valid_ip : valid_ips_array) {
                    if (ipAddr.equals(valid_ip.trim())) {
                        isPresent = true;
                    }
                }
                if (isPresent) {
                    result.setStatus(IResult.SUCCEEDED);
                } else {
                  result.setStatus(IResult.FAILED);
                  result.setFault(new WSMException(WSMException.FAULT_FAILED_CHECK));
                }
            } else {
                result.setStatus(IResult.SUCCEEDED);
            }
            return result;
        } catch (Exception e) {
            throw new WSMException(WSMException.FAULT_FAILED_CHECK, e);
        }
    }

    public oracle.wsm.common.sdk.IResult postExecute(oracle.wsm.common.sdk.IContext p1) {
        IResult result = new Result();
        result.setStatus(IResult.SUCCEEDED);
        return result;
    }
}
```

2.2.2 Step 2: Create the Custom Policy File

Create the custom policy file to define the bindings for and configure the custom assertion. "Schema Reference for Custom Assertions" on page A-1 describes the schema that you can use to construct your custom policy file and custom assertion.

The following example defines the oracle/ip_assertion_policy custom policy file. The assertion defines a comma-separated list of IP addresses that are valid for a request.

Example 2–2   Example Custom Policy File

```
<?xml version = '1.0' encoding = 'UTF-8'?>

<wsp:Policy xmlns="http://schemas.xmlsoap.org/ws/2004/09/policy"
   xmlns:orasp="http://schemas.oracle.com/ws/2006/01/securitypolicy"
   orawsp:status="enabled"

xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" orawsp:category="security"
   orawsp:attachTo="binding.server" wsu:Id="ip_assertion_policy"
   xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
   wsp:Name="oracle/ip_assertion_policy">
     <orasp:ipAssertion orawsp:Silent="true" orawsp:Enforced="true"
```

```
                    orawsp:name="WSSecurity IpAssertion Validator"
          orawsp:category="security/authentication">
                    <orawsp:bindings>
                        <orawsp:Config orawsp:name="ipassertion" orawsp:configType="declarative">
                            <orawsp:PropertySet orawsp:name="valid_ips">
                                <orawsp:Property orawsp:name="valid_ips" orawsp:type="string"
                                 orawsp:contentType="constant">
                                    <orawsp:Value>127.0.0.1,192.168.1.1</orawsp:Value>
                                </orawsp:Property>
                            </orawsp:PropertySet>
                        </orawsp:Config>
                    </orawsp:bindings>
            </orasp:ipAssertion>
</wsp:Policy>
```

2.2.3 Step 3: Specify the Custom Assertion Executor

Specify the custom assertion executor in any one of the following files:

- Custom policy file
- policy-config.xml file

Specifying the Custom Assertion Executor in the Custom Policy File

Update the custom policy to specify the custom executor information in the orawsp:Implementation element as shown in Example 2–3.

Example 2–3   Specifying the Custom Assertion Executor in the Custom Policy File

```
<?xml version = '1.0' encoding = 'UTF-8'?><wsp:Policy
 xmlns="http://schemas.xmlsoap.org/ws/2004/09/policy"
 xmlns:orasp="http://schemas.oracle.com/ws/2006/01/securitypolicy"
orawsp:status="enabled"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" orawsp:category="security"
 orawsp:attachTo="binding.server" wsu:Id="ip_assertion_policy"
xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
wsp:Name="oracle/ip_assertion_policy">
            <orasp:ipAssertion orawsp:Silent="true" orawsp:Enforced="true"
 orawsp:name="WSSecurity IpAsertion validator" orawsp:category="security/authentication">
                <orawsp:bindings>

<orawsp:Implementation>sampleassertion.IpAssertionExecutor</orawsp:Implementation>
                    <orawsp:Config orawsp:name="ipassertion" orawsp:configType="declarative">
                        <orawsp:PropertySet orawsp:name="valid_ips">
                            <orawsp:Property orawsp:name="valid_ips" orawsp:type="string"
 orawsp:contentType="constant">
                                <orawsp:Value>140.87.6.143,10.178.93.107</orawsp:Value>
                            </orawsp:Property>
                        </orawsp:PropertySet>
                    </orawsp:Config>
                </orawsp:bindings>
            </orasp:ipAssertion>
</wsp:Policy>
```

Specifying the Custom Assertion Executor in the policy-config.xml file

Create a policy-config.xml file that defines an entry for the new assertion and associates it with its custom assertion executor.

The format for the policy-config.xml file is shown in Example 2–4.

Example 2–4   policy-config.xml File Format

```
<?xml version="1.0" encoding="UTF-8"?>
<policy-config>
    <policy-model-config>
        <entry>
            <key namespace="namespace" element-name="elementname"/>
            <executor-classname>assertionclass</executor-classname>
        </entry>
    </policy-model-config>
</policy-config>
```

Table 2–1 lists the attributes for the key element.

Table 2–1   Attributes for Key Element

Attribute	Description
namespace	Namespace of the policy. This value must match the namespace defined in the custom policy file (in Step 1).
	In Example 2–2, the namespace is defined as part of the <wsp:Policy> tag as follows:
	`xmlns:orasp="`**`http://schemas.oracle.com/ws/2006/01/secur`** **`itypolicy`**`"`
element-name	Name of the element. This value must match the assertion name defined in the custom policy file (in Step 1).
	In Example 2–2, the element name ipAssertion is defined in the following tag:
	`<orasp:`**`ipAssertion`** `orawsp:Silent="true"` `orawsp:Enforced="true" orawsp:name="WSSecurity IpAssertion Validator"` `orawsp:category="security/authentication">`

Example 2–5 provides an example of a policy-config.xml file with an entry for the ipAssertion policy.

Example 2–5   Example policy-config.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<policy-config>
    <policy-model-config>
        <entry>
            <key namespace="http://schemas.oracle.com/ws/2006/01/securitypolicy"
 element-name="ipAssertion"/>

<executor-classname>sampleassertion.IpAssertionExecutor</executor-classname>
        </entry>
    </policy-model-config>
</policy-config>
```

> **Note:**   The policy-config.xml file must be in the classpath of server. This file is also added to the custom executor jar file as mentioned in Section 2.2.4, "Step 4: Create the JAR File".

2.2.4 Step 4: Create the JAR File

Create the custom assertion JAR file that includes the custom assertion executor and the policy-config.xml file. You can use Oracle JDeveloper, other IDE, or the jar tool to generate the JAR file.

2.2.5 Step 5: Add the Custom Policy to the Policy Store

Add the custom policy to the policy store using Fusion Middleware Control or WLST, as described in the following sections.

Using Fusion Middleware Control

Before you can attach the custom policy to a Web service, you must import it using the procedure described in "Importing Web Service Policies" in *Security and Administrator's Guide for Web Services*.

Using WLST

Use the WebLogic Scripting Tool (WLST) commands to import the custom policy. For information, see "Migrating Policies" in Security and Administrator's Guide for Web Services.

2.2.6 Step 6: Deploy the Custom Assertion

Add the custom assertion JAR to your CLASSPATH by performing the following steps:

1. Stop the WebLogic Server.

   For more information on stopping the WebLogic Server, see *Managing Server Startup and Shutdown for Oracle WebLogic Server*.

2. Copy the custom assertion JAR file created in Step 4 to the following directory: $DOMAIN_HOME/lib.

3. Restart the WebLogic Server.

   For more information on restarting the WebLogic Server, see *Managing Server Startup and Shutdown for Oracle WebLogic Server*.

2.2.7 Step 7: Attach the Custom Policy to a Web Service

Create a Web service using the information described in "Roadmap for Implementing WebLogic Web Services" in *Introducing WebLogic Web Services for Oracle WebLogic Server*.

Attach the custom policy to the Web service, as described in the following sections.

Using Fusion Middleware Control

Attach the custom policy to a Web service using Fusion Middleware Control as described in "Attaching a Policy to a Web Service Using Fusion Middleware Control" in Security and Administrator's Guide for Web Services.

Using WebLogic Administration Console

Attach the custom policy to a Web service using WebLogic Administration Console as described in "Attach a WS-Policy file to a Web service" in the *WebLogic Administration Console Help*.

Using WLST

Attach the custom policy to a Web service using WLST as described in "Attaching a Policy to a Web Service Using WLST" in Security and Administrator's Guide for Web Services.

Using JDeveloper

Attach the custom policy to a Web service using JDeveloper as described in "Using Custom Web Service Policies" in JDeveloper Online Help.

2.2.8 Step 8: Generate the Client

Create a client proxy for the Web service using clientgen.

For more information, see "Using the clientgen Ant Task to Generate Client Artifacts" in *Oracle Fusion Middleware Getting Started With JAX-WS Web Services for Oracle WebLogic Server*.

2.3 Testing the Web Service

Use the Fusion Middleware Control Test Web Service page to test the operations and view results of the Web service without deploying the Web service. For more information, see "Testing Web Services" in *Security and Administrator's Guide for Web Services*.

2.4 Creating a Custom Client Policy with Custom Assertions

Create a custom client policy from the Web service custom assertions you created in Section 2.2, "Developing Custom Assertions for Web Service". For more information on generating client policies, see "Generating Client Policies" in *Security and Administrator's Guide for Web Services*.

3

Stepping Through Sample Custom Assertions

This chapter describes sample custom assertions. It includes the following sections:

- Overview
- IP Address Validation Custom Assertion Sample
- Encryption and Decryption Custom Assertion Sample
- Authentication Custom Assertion Sample

3.1 Overview

This chapter describes some sample custom assertions and provides sample code for each custom assertion. Each sample also includes a section that describes the key features that are implemented in the sample.

3.2 IP Address Validation Custom Assertion Sample

The IP Validation Custom Assertion sample validates whether a request that is made to the Web service is from a set of valid IP addresses. In the custom policy assertion you can include the valid IP addresses as a comma-separated list of values. Any request coming from any other IP address results in a FAILED_CHECK response.

 The following sections provide samples that demonstrate how to code the custom assertion:

- Section 3.2.1, "Code Samples"
- Section 3.2.2, "Running This Sample"

3.2.1 Code Samples

This section includes the following sample codes to create and implement the custom assertion:

- "Sample Custom Assertion Executor" on page 3-2
- "Sample Policy File" on page 3-3
- "Sample policy-config.xml" on page 3-4
- "Sample Web Service" on page 3-4
- "Sample JSE Client" on page 3-5

Sample Custom Assertion Executor

Example 3–1 provides a sample custom assertion executor that you can use to validate the IP address.

Example 3–1   Sample Custom Assertion Executor

```
package sampleassertion;

import oracle.wsm.common.sdk.IContext;
import oracle.wsm.common.sdk.IMessageContext;
import oracle.wsm.common.sdk.IResult;
import oracle.wsm.common.sdk.Result;
import oracle.wsm.common.sdk.WSMException;
import oracle.wsm.policy.model.IAssertionBindings;
import oracle.wsm.policy.model.IConfig;
import oracle.wsm.policy.model.IPropertySet;
import oracle.wsm.policy.model.ISimpleOracleAssertion;
import oracle.wsm.policy.model.impl.SimpleAssertion;
import oracle.wsm.policyengine.impl.AssertionExecutor;

public class IpAssertionExecutor extends AssertionExecutor {

    public IpAssertionExecutor() {
    }

    public void destroy() {
    }

    public void init(oracle.wsm.policy.model.IAssertion assertion,
                     oracle.wsm.policyengine.IExecutionContext econtext,
                     oracle.wsm.common.sdk.IContext context) {
        this.assertion = assertion;
        this.econtext = econtext;
    }

    public oracle.wsm.policyengine.IExecutionContext getExecutionContext() {
        return this.econtext;
    }

    public boolean isAssertionEnabled() {
        return ((ISimpleOracleAssertion)this.assertion).isEnforced();
    }

    public String getAssertionName() {
        return this.assertion.getQName().toString();
    }

    /**
     * @param context
     * @return
     */
    public IResult execute(IContext context) throws WSMException {
        try {
         oracle.wsm.common.sdk.IMessageContext.STAGE stage =
    ((oracle.wsm.common.sdk.IMessageContext)context).getStage();

            if (stage  == IMessageContext.STAGE.request)  {


                IAssertionBindings bindings =
```

```
 ((SimpleAssertion)(this.assertion)).getBindings();
                IConfig config = bindings.getConfigs().get(0);
                IPropertySet propertyset = config.getPropertySets().get(0);
                String valid_ips = propertyset.getPropertyByName("valid_
ips").getValue();
                String ipAddr = ((IMessageContext)context).getRemoteAddr();
                IResult result = new Result();

                if (valid_ips != null && valid_ips.trim().length() > 0) {
                    String[] valid_ips_array = valid_ips.split(",");
                    boolean isPresent = false;
                    for (String valid_ip : valid_ips_array) {
                      if (ipAddr.equals(valid_ip.trim())) {
                          isPresent = true;
                      }
                    }
                    if (isPresent) {
                      result.setStatus(IResult.SUCCEEDED);
                    } else {
                      result.setStatus(IResult.FAILED);
                      result.setFault(new WSMException(WSMException.FAULT_FAILED
_CHECK));
                    }
                } else {
                 result.setStatus(IResult.SUCCEEDED);
                }
                return result;
          }
        } catch (Exception e) {
            throw new WSMException(WSMException.FAULT_FAILED_CHECK, e);
        }
    }

    public oracle.wsm.common.sdk.IResult
 postExecute(oracle.wsm.common.sdk.IContext p1) {
        IResult result = new Result();
        result.setStatus(IResult.SUCCEEDED);
        return result;
    }
}
```

Sample Policy File

Example 3–2 describes a sample policy file.

Example 3–2   Sample Policy File

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<wsp:Policy xmlns="http://schemas.xmlsoap.org/ws/2004/09/policy"
 xmlns:orasp="http://schemas.oracle.com/ws/2006/01/securitypolicy"
orawsp:status="enabled"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-uti
lity-1.0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" orawsp:category="security"
 orawsp:attachTo="binding.server" wsu:Id="ip_assertion_policy"
xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
wsp:Name="oracle/ip_assertion_policy">
            <orasp:ipAssertion orawsp:Silent="true" orawsp:Enforced="true"
 orawsp:name="WSSecurity IpAsertion validator"
 orawsp:category="security/authentication">
```

```
                    <orawsp:bindings>
                        <orawsp:Config orawsp:name="ipassertion"
 orawsp:configType="declarative">
                            <orawsp:PropertySet orawsp:name="valid_ips">
                                <orawsp:Property orawsp:name="valid_ips"
 orawsp:type="string" orawsp:contentType="constant">

<orawsp:Value>140.87.6.143,10.178.93.107</orawsp:Value>
                                </orawsp:Property>
                            </orawsp:PropertySet>
                        </orawsp:Config>
                    </orawsp:bindings>
                </orasp:ipAssertion>
</wsp:Policy>
```

Sample policy-config.xml

Example 3–3 demonstrates how to use the policy-config.xml file to specify the custom
assertion executor. You can also use the custom policy to specify the custom policy
executor, as described in Section 2.2.3, "Step 3: Specify the Custom Assertion
Executor".

Example 3–3   Sample policy-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<policy-config>
    <policy-model-config>
      <entry>
        <key namespace="http://schemas.oracle.com/ws/2006/01/securitypolicy"
 element-name="ipAssertion"/>

<executor-classname>sampleassertion.IpAssertionExecutor</executor-classname>
      </entry>
    </policy-model-config>
</policy-config>
```

Sample Web Service

Example 3–4 describes a sample Web service. You can attach the custom assertion to
Web service as described in Section 2.2.7, "Step 7: Attach the Custom Policy to a Web
Service".

Example 3–4   Sample Web Service

```
package project1;
import javax.jws.WebService;
import weblogic.wsee.jws.jaxws.owsm.SecurityPolicies;
@WebService
@SecurityPolicy(uri="policy:oracle/ip_assertion_policy")
public class Class1 {
    public Class1() {
        super();
    }

    public String echo1() {
        return "one";
    }
}
```

Sample JSE Client

Example 3–5 is a sample of a JSE client generated from the Web service in
Example 3–4.

Example 3–5   Sample JSE Client

```
package project1;
import javax.xml.ws.WebServiceRef;

public class Class1PortClient
{
  @WebServiceRef
  private static Class1Service class1Service;
  public static void main(String [] args)
  {
    class1Service = new Class1Service();
    Class1 port = class1Service.getClass1Port();
    // Add your code to call the desired methods.

    System.out.println(port.echo1());
  }
}
```

3.2.2 Running This Sample

To use the IP Validation custom assertion sample, perform the following steps:

1. Create the custom assertion and custom assertion executor with the sample codes
    as described in Section 3.2.1, "Code Samples". These samples demonstrate the
    following key features:

    -   Specify the valid values of IP addresses in the custom assertion:

        ```
        <orawsp:PropertySet orawsp:name="valid_ips">
          <orawsp:Property orawsp:name="valid_ips" orawsp:type="string"
         orawsp:contentType="constant">
          <orawsp:Value>140.87.6.143,10.178.93.107</orawsp:Value>
          </orawsp:Property>
        </orawsp:PropertySet>
        ```

         For more information, see Section 4.1, "Inputing Parameters to Custom
        Assertions".

    -   View the execution stage in the custom assertion executor:

        ```
        oracle.wsm.common.sdk.IMessageContext.STAGE stage =
         ((oracle.wsm.common.sdk.IMessageContext)context).getStage()
        ```

        For more information, see Section 4.3, "Accessing Request, Response, and
        Fault Message Objects".

    -   Access the IP address from the custom assertion executor:

        ```
        IConfig config = bindings.getConfigs().get(0);
        IPropertySet propertyset = config.getPropertySets().get(0);
        String valid_ips = propertyset.getPropertyByName("valid_ips").getValue();
        ```

         For more information, see Section 4.1, "Inputing Parameters to Custom
        Assertions".

    -   Access the context property remote address:

        ```
        String ipAddr = ((IMessageContext)context).getRemoteAddr();
        ```

> For more information, see Section 4.2, "Accessing Oracle WSM Context Properties".

- In the event of failure, set the fault that caused the request execution to fail:

```
result.setFault(new WSMException(WSMException.FAULT_FAILED_CHECK));
```

> For more information, see Section 4.8, "Handling Exceptions in Custom Assertions".

2. Create the JAR file as described in Section 2.2.4, "Step 4: Create the JAR File".

3. Add the custom policy to the policy store as described in Section 2.2.5, "Step 5: Add the Custom Policy to the Policy Store".

4. Update the classpath as described in Section 2.2.6, "Step 6: Deploy the Custom Assertion".

5. Attach the custom policy to the Web service by any one of the methods described in Section 2.2.7, "Step 7: Attach the Custom Policy to a Web Service".

3.3 Encryption and Decryption Custom Assertion Sample

In this sample, a set of custom assertions are used to encrypt data from an inbound message, making it unreadable from consoles, audit trail, or logs. The encrypted data is decrypted for outbound messages for downstream services that require access to the data.

The following sections provide samples that demonstrate how to code the custom assertion and describe the sample in detail:

- Section 3.3.1, "Code Samples"

- Section 3.3.2, "Running the Sample"

3.3.1 Code Samples

This section includes the following sample codes to create and implement the custom assertion:

- "Sample Custom Assertion Executor" on page 3-6

- "Sample Inbound Encryption Class" on page 3-9

- "Sample Outbound Decryption Class" on page 3-13

- "Sample Policy File for Encryption" on page 3-17

- "Sample Policy File for Decryption" on page 3-17

- "Sample Composite for Receiving the Encrypted Message" on page 3-18

- "Sample External Service Composite" on page 3-19

Sample Custom Assertion Executor

Example 3–6 describes a sample custom assertion executor used for encryption and decryption of data. For more information on the implementation, see Section 3.3.2, "Running the Sample".

Example 3–6   Sample Custom Assertion Executor

```
package owsm.custom.soa;
```

```java
import java.util.HashMap;
import java.util.Iterator;

import javax.xml.namespace.NamespaceContext;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import oracle.wsm.common.sdk.IContext;
import oracle.wsm.policy.model.IAssertion;
import oracle.wsm.policyengine.IExecutionContext;
import oracle.wsm.policyengine.impl.AssertionExecutor;

import org.w3c.dom.Element;
import org.w3c.dom.Node;

/**
 * A base class for OWSM custom assertions that specific classes can extend. It
 * contains common code and variables used accross all custom assertions.
 *
 * All custom assertions must extend the AssertionExecutor class.
 */
public abstract class CustomAssertion
  extends AssertionExecutor
{


  protected final static String PROP_DEBUG = "debugFlag";

  protected final static String DEBUG_START =
  "=============================================================>>>";
  protected final static String DEBUG_END =
  "<<<=============================================================";

  protected IAssertion mAssertion = null;
  protected IExecutionContext mEcontext = null;
  protected oracle.wsm.common.sdk.IContext mIcontext = null;

  /**
   * A tag or text to display when printing debug information to identify the
   * content.
   */
  protected String mTag;

  /**
   * Constructor
   */
  public CustomAssertion(String tag)
  {
    super();
    mTag = tag;
  } // CustomAssertion()

  /**
   * Implemented from parent class
   */
  public void init(IAssertion iAssertion,
                     IExecutionContext iExecutionContext,
```

```
                        IContext iContext)
{
  mAssertion = iAssertion;
  mEcontext = iExecutionContext;
  mIcontext = iContext;
  //IAssertionBindings bindings = ((SimpleAssertion)

(mAssertion)).getBindings();
} // init()
/**
 * Implemented from parent class
 */
public void destroy()
{
  // Nothing to do.
} // destroy()

/**
 * A utility method for extracting the node specified by <code>xpathStr</code>
 * (with namespaces defined by <code>namespaces</code>) from
*<code>payload</code>
 *
 * This method will print an stack trace if their is an exception. If you want
*to
 * return the exception instead then modify the method appropriately.
 *
 * @param payload the payload
 * @param namespaces the namespaces referenced by <code>xpathStr</code>
 * @param xpathStr an XPath query defining how to extract a node from
*<code>payload</code>
 * @return
 */
public static Node getDataNode(Element payload, final HashMap<String, String>

namespaces, String xpathStr)
{
  Node node = null;

  try
  {
    // Create a namespace context based on the namespaces passed in.
    //
    NamespaceContext ctx = new NamespaceContext()
    {
      public String getNamespaceURI(String prefix)
      {
        return namespaces.get(prefix);
      }
      // Dummy implementation - not used
      public Iterator getPrefixes(String val)
      {
        return null;
      }
      // Dummy implemenation - not used
      public String getPrefix(String uri)
      {
        return null;
      }
    };
    XPathFactory xpathFact = XPathFactory.newInstance();
```

```
      XPath xpath = xpathFact.newXPath();
      xpath.setNamespaceContext(ctx);

      node = (Node)xpath.evaluate(xpathStr, payload, XPathConstants.NODE);
    }
    catch (XPathExpressionException ex)
    {
      ex.printStackTrace();
      return null;
    }

    return node;
  } // getDataNode()

}
```

Sample Inbound Encryption Class

Example 3–7 provides a sample custom assertion executor for encrypting an element
of an inbound message.

Example 3–7   Sample Inbound Encryption Class

```
package owsm.custom.soa;

import java.util.HashMap;

import javax.xml.soap.SOAPBody;

import oracle.wsm.common.sdk.IContext;
import oracle.wsm.common.sdk.IMessageContext;
import oracle.wsm.common.sdk.IResult;
import oracle.wsm.common.sdk.Result;
import oracle.wsm.common.sdk.SOAPBindingMessageContext;
import oracle.wsm.common.sdk.WSMException;
import oracle.wsm.policy.model.IAssertionBindings;
import oracle.wsm.policy.model.IConfig;
import oracle.wsm.policy.model.IProperty;
import oracle.wsm.policy.model.IPropertySet;
import oracle.wsm.policy.model.impl.SimpleAssertion;

import oracle.xml.parser.v2.XMLElement;

import org.w3c.dom.Node;


/**
 * A custom assertion class for encrypting an element of an inbound message.
 */
public class InboundEncryptor
  extends CustomAssertion
{
  /**
   * Constructor
   */
  public InboundEncryptor()
  {
    super("[InboundEncryptor] ");
  } // InboundEncryptor()
```

```
/**
 * Process an inbound message for the given invocation context
 *
 * @param iContext the invocation context for a specific message
 * @return the result of any actions taken. It can return a success message or
 *         a fault with an exception message
 */
public IResult execute(IContext iContext)
{
  // Create the result that's going be populate with success or failure
  // depending on what happens.
  //
  IResult result = new Result();

  // Specify whether we are in debug mode or not. If true then debug statements
  // will be printed to the log file.
  // This can be set to true in the OWSM policy assertion.
  //
  boolean debug = false;

  try
  {
    // Get the property set which contains properties defined in the OWSM
    // policy assertion.
    //
    IAssertionBindings bindings = ((SimpleAssertion)
(mAssertion)).getBindings();
    IConfig config = bindings.getConfigs().get(0);
    IPropertySet propertyset = config.getPropertySets().get(0);

    // Now that we have the property set, let's check it for the specific
    // properties we care about.
    //

    // Check for the debug flag property.
    //
    IProperty debugProp = propertyset.getPropertyByName(PROP_DEBUG);
    if (debugProp != null)
    {
      String debugStr = debugProp.getValue();
      debug = Boolean.valueOf(debugStr).booleanValue();
    }
    if (debug)
    {
      System.out.println(mTag+DEBUG_START);
      System.out.println(mTag+this.getClass().getSimpleName()+".execute()
Starting...");
      System.out.println(mTag+"In debug mode");
    }

    // Check for the stage. We only care about the request stage for this
    // implementation.
    //
    IMessageContext.STAGE stage = ((IMessageContext) iContext).getStage();
    if (debug) System.out.println(mTag+"stage="+stage);
    if (stage != IMessageContext.STAGE.request)
    {
      result.setStatus(IResult.SUCCEEDED);
```

```
      if (debug)
      {
        System.out.println(mTag+"Nothing to process in this stage. Returning");
        System.out.println(mTag+DEBUG_END);
      }
      return result;
    }
    // Get the encryption key, which is a property on the assertion.
    //
    String key = propertyset.getPropertyByName("encryption_key").getValue();
    if (debug) System.out.println(mTag+"key=[" + key + "]");
    if (key == null || key.trim().length() == 0)
    {
      result.setStatus(IResult.FAILED);
      result.setFault(new WSMException("Invalid key"));
      if (debug)
      {
        System.out.println(mTag+"Invalid key");
        System.out.println(mTag+DEBUG_END);
      }
      return result;
    }
    // As as point of interest, you can get the service URL. This could be used
    // by this class to know which service this message is bound for, and
//therefore
    // which XPath expression to use.
    // In this example no such logic is needed as we only have one servcie
    // to worry about.
    //
    if (debug)
    {
      String serviceURL = ((IMessageContext) iContext).getServiceURL();
      System.out.println(mTag+"serviceURL=[" + serviceURL+"]");
    }
    // Get the message.
    //
    SOAPBindingMessageContext soapbindingmessagecontext =
(SOAPBindingMessageContext) iContext;
    javax.xml.soap.SOAPMessage soapMessage =
soapbindingmessagecontext.getRequestMessage();
    SOAPBody soapElem = soapMessage.getSOAPBody();
    if (debug)
    {
      System.out.println(mTag+"----- Start ORIGINAL inbound message -----");
      ((XMLElement) soapElem).print(System.out);
      System.out.println(mTag+"-----  End ORIGINAL inbound message  -----");
    }
    // Create the XPath to reference the element which is to be encrypted.
    //
    String xpathStr =
"/soap:Envelope/soap:Body/ns1:process/ns1:order/ns1:ccNum";
    // Build up a namespace list for any namespaces referenced by the
    // XPath expression. This will be the basis for a namespace context
    // created later.
    //
    final HashMap<String, String> namespaces = new HashMap<String, String>();
    namespaces.put("soap", "http://schemas.xmlsoap.org/soap/envelope/");
    namespaces.put("ns1", "http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionComposite/ProcessCustomer");
```

```
      // Extract the node that should be encrypted.
      //
      Node inputNode = getDataNode(soapElem, namespaces, xpathStr);
      if (inputNode == null)
      {
        // Something went wrong, but getDataNode() would've printed out a
//stacktrace
        // so print out a debug statement and exit.
        //
        result.setStatus(IResult.FAILED);
        result.setFault(new WSMException("Cannot find node with XPath expression:
"+xpathStr));
        if (debug)
        {
          System.out.println(mTag+"Cannot find node with XPath expression:
"+xpathStr);
          System.out.println(mTag+DEBUG_END);
        }
        return result;
      }

      // Extract the string value of the element to be encrypted.
      //
      String inputValue = inputNode.getTextContent();

      // Get an instance of EncDec and perform the actual encryption.
      //
      EncDec ed = EncDec.getInstance();
      String encryptedInput = ed.encryptStrToStr(inputValue, key);
      if (debug) System.out.println(mTag+"result of encryption=[" + encryptedInput
+ "]");
      // Replace the value of the node with the encrypted value.
      //
      try
      {
        inputNode.setTextContent(encryptedInput);
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
      if (debug)
      {
        System.out.println(mTag+"----- Start MODIFIED inbound message -----");
        ((XMLElement) soapElem).print(System.out);
        System.out.println(mTag+"-----  End MODIFIED inbound message  -----");
      }
      // Set a happy result.
      //
      result.setStatus(IResult.SUCCEEDED);
    }
    catch (Exception e)
    {
      // This is a general or catchall handler.
      //
      result.setStatus(IResult.FAILED);
      result.setFault(new WSMException(WSMException.FAULTCODE_QNAME_FAILED_CHECK,
e));
      if (debug) System.out.println(this.getClass().getName()+": ERROR: Got an
exception somewhere...");
```

```
    }

    if (debug) System.out.println(mTag+DEBUG_END);
    return result;

  } // execute()

} // InboundEncryptor
```

Sample Outbound Decryption Class

Example 3–8 provides a sample custom assertion executor for decrypting an element of an outbound message.

Example 3–8   Sample Outbound Decryption Class

```java
package owsm.custom.soa;

import java.util.HashMap;

import javax.xml.soap.SOAPBody;

import oracle.wsm.common.sdk.IContext;
import oracle.wsm.common.sdk.IMessageContext;
import oracle.wsm.common.sdk.IResult;
import oracle.wsm.common.sdk.Result;
import oracle.wsm.common.sdk.SOAPBindingMessageContext;
import oracle.wsm.common.sdk.WSMException;
import oracle.wsm.policy.model.IAssertionBindings;
import oracle.wsm.policy.model.IConfig;
import oracle.wsm.policy.model.IProperty;
import oracle.wsm.policy.model.IPropertySet;
import oracle.wsm.policy.model.impl.SimpleAssertion;

import oracle.xml.parser.v2.XMLElement;

import org.w3c.dom.Node;


/**
 * A custom assertion class for decrypting an element of an outbound message.
 */
public class OutboundDecryptor
  extends CustomAssertion
{

  /**
   * Constructor
   */
  public OutboundDecryptor()
  {
    super("[OutboundDecryptor] ");
  } // OutboundDecryptor()

  //////////////////////////////////////////////////////////////////////////

  /**
   * Process an outbound message for the given invocation context
   *
   * @param iContext the invocation context for a specific message
```

```
 * @return the result of any actions taken. It can return a success message or
 *          a fault with an exception message
 */
public IResult execute(IContext iContext)
{
  // Create the result that's going be populate with success or failure
  // depending on what happens.
  //
  IResult result = new Result();

  // Specify whether we are in debug mode or not. If true then debug statements
  // will be printed to the log file.
  // This can be set to true in the OWSM policy assertion.
  //
  boolean debug = false;

  try
  {
    // Get the property set which contains properties defined in the OWSM
    // policy assertion.
    //
    IAssertionBindings bindings = ((SimpleAssertion)
(mAssertion)).getBindings();
    IConfig config = bindings.getConfigs().get(0);
    IPropertySet propertyset = config.getPropertySets().get(0);

    // Now that we have the property set, let's check it for the specific
    // properties we care about.
    //

    // Check for the debug flag property.
    //
    IProperty debugProp = propertyset.getPropertyByName(PROP_DEBUG);
    if (debugProp != null)
    {
      String debugStr = debugProp.getValue();
      debug = Boolean.valueOf(debugStr).booleanValue();
    }
    if (debug)
    {
      System.out.println(mTag+DEBUG_START);
      System.out.println(mTag+this.getClass().getSimpleName()+".execute()
Starting...");
      System.out.println(mTag+"In debug mode");
    }

    // Check for the stage. We only care about the request stage for this
    // implementation.
    //
    IMessageContext.STAGE stage = ((IMessageContext) iContext).getStage();
    if (debug) System.out.println(mTag+"stage="+stage);
    if (stage != IMessageContext.STAGE.request)
    {
      result.setStatus(IResult.SUCCEEDED);
      if (debug)
      {
        System.out.println(mTag+"Nothing to process in this stage. Returning");
        System.out.println(mTag+DEBUG_END);
      }
      return result;
```

```
    }
    // Get the encryption key, which is a property on the assertion.
    //
    String key = propertyset.getPropertyByName("decryption_key").getValue();
    if (debug) System.out.println(mTag+"key=[" + key + "]");
    if (key == null || key.trim().length() == 0)
    {
      result.setStatus(IResult.FAILED);
      result.setFault(new WSMException("Invalid key"));
      if (debug)
      {
        System.out.println(mTag+"Invalid key");
        System.out.println(mTag+DEBUG_END);
      }
      return result;
    }
    String serviceURL = ((IMessageContext) iContext).getServiceURL();
    if (debug) System.out.println(mTag+"serviceURL=[" + serviceURL+"]");
    // Get the message.
    //
    SOAPBindingMessageContext soapbindingmessagecontext =

(SOAPBindingMessageContext) iContext;
    javax.xml.soap.SOAPMessage soapMessage =
soapbindingmessagecontext.getRequestMessage();
    SOAPBody soapElem = soapMessage.getSOAPBody();

    // As as point of interest, you can get the service URL. This could be used
    // by this class to know which service this message is bound for, and
//therefore
    // which XPath expression to use.
    // In this example no such logic is needed as we only have one servcie
    // to worry about.
    //
    if (debug)
    {
      System.out.println(mTag+"----- Start ORIGINAL inbound message -----");
      ((XMLElement) soapElem).print(System.out);
      System.out.println(mTag+"-----  End ORIGINAL inbound message  -----");
    }
    // Create the XPath to reference the element which is to be encrypted.
    //
    String xpathStr = "/soap:Envelope/soap:Body/ns1:process/ns1:ccNum";

    // Build up a namespace list for any namespaces referenced by the
    // XPath expression. This will be the basis for a namespace context
    // created later.
    //
    final HashMap<String, String> namespaces = new HashMap<String, String>();
    namespaces.put("soap", "http://schemas.xmlsoap.org/soap/envelope/");
    namespaces.put("ns1", "http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionExternalService/ExternalServiceBpel");

    // Extract the node that should be encrypted.
    //
    Node inputNode = getDataNode(soapElem, namespaces, xpathStr);
    if (inputNode == null)
    {
      // Something went wrong, but getDataNode() would've printed out a
//stacktrace
```

```
            // so print out a debug statement and exit.
            //
            result.setStatus(IResult.FAILED);
            result.setFault(new WSMException("Cannot find node with XPath expression:
"+xpathStr));
            if (debug)
            {
              System.out.println(mTag+"Cannot find node with XPath expression:
"+xpathStr);
              System.out.println(mTag+DEBUG_END);
            }
            return result;
          }

          // Extract the string value of the element to be encrypted.
          //
          String inputValue = inputNode.getTextContent();

          // Get an instance of EncDec and perform the actual encryption.
          //
          EncDec ed = EncDec.getInstance();
          String decryptedInput = ed.decryptStrToStr(inputValue, key);
          if (debug) System.out.println(mTag+"result of decryption=[" + decryptedInput
+ "]");
          // Replace the value of the node with the encrypted value.
          //
          try
          {
            inputNode.setTextContent(decryptedInput);
          }
          catch (Exception ex)
          {
            ex.printStackTrace();
          }
          if (debug)
          {
            System.out.println(mTag+"----- Start MODIFIED inbound message -----");
            ((XMLElement) soapElem).print(System.out);
            System.out.println(mTag+"-----  End MODIFIED inbound message  -----");
          }
          // Set a happy result.
          //
          result.setStatus(IResult.SUCCEEDED);
        }
        catch (Exception e)
        {
          // This is a general or catchall handler.
          //
          result.setStatus(IResult.FAILED);
          result.setFault(new WSMException(WSMException.FAULTCODE_QNAME_FAILED_CHECK,
e));
          if (debug) System.out.println(this.getClass().getName()+": ERROR: Got an
exception somewhere...");
        }

        if (debug) System.out.println(mTag+DEBUG_END);
        return result;

    } // execute()
```

```
} // OutboundDecryptor
```

Sample Policy File for Encryption

Example 3–9 provides a sample custom assertion executor for encryption.

Example 3–9   Sample Policy File for Encryption

```
<orasp:Assertion xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
                         orawsp:Id="soa_encryption_template"
                         orawsp:attachTo="generic" orawsp:category="security"
                         orawsp:description="Custom Encryption of payload"
                         orawsp:displayName="Custom Encryption"
                         orawsp:name="custom/soa_encryption"
                         xmlns:custom="http://schemas.oracle.com/ws/soa/custom">
  <custom:custom-executor orawsp:Enforced="true" orawsp:Silent="false"
                  orawsp:category="security/custom"
                  orawsp:name="WSSecurity_Custom_Assertion">
    <orawsp:bindings>

<orawsp:Implementation>owsm.custom.soa.InboundEncryptor</orawsp:Implementation>
      <orawsp:Config orawsp:configType="declarative" orawsp:name="encrypt_soa">
        <orawsp:PropertySet orawsp:name="encrypt">
          <orawsp:Property orawsp:contentType="constant"
                           orawsp:name="encryption_key" orawsp:type="string">
            <orawsp:Value>MySecretKey</orawsp:Value>
          </orawsp:Property>
          <orawsp:Property orawsp:contentType="constant"
                           orawsp:name="debugFlag" orawsp:type="string">
            <orawsp:Value>true</orawsp:Value>
          </orawsp:Property>
        </orawsp:PropertySet>
      </orawsp:Config>
    </orawsp:bindings>
  </custom:custom-executor>
</orasp:Assertion>
```

Sample Policy File for Decryption

Example 3–10 provides a sample custom assertion for decryption.

Example 3–10   Sample Policy File for Decryption

```
<orasp:Assertion xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
                         orawsp:Id="soa_decryption_template"
                         orawsp:attachTo="binding.client"
 orawsp:category="security"
                         orawsp:description="Custom Decryption of payload"
                         orawsp:displayName="Custom Decryption"
                         orawsp:name="custom/soa_decryption"
                         xmlns:custom="http://schemas.oracle.com/ws/soa/custom">
  <custom:custom-executor orawsp:Enforced="true" orawsp:Silent="false"
                  orawsp:category="security/custom"
                  orawsp:name="WSSecurity Custom Assertion">
    <orawsp:bindings>

<orawsp:Implementation>owsm.custom.soa.OutboundDecryptor</orawsp:Implementation>
      <orawsp:Config orawsp:configType="declarative" orawsp:name="encrypt_soa">
        <orawsp:PropertySet orawsp:name="decrypt">
          <orawsp:Property orawsp:contentType="constant"
```

```
                                    orawsp:name="decryption_key" orawsp:type="string">
                    <orawsp:Value>MySecretKey</orawsp:Value>
                </orawsp:Property>
                <orawsp:Property orawsp:contentType="constant"
                                    orawsp:name="debugFlag" orawsp:type="string">
                    <orawsp:Value>true</orawsp:Value>
                </orawsp:Property>
            </orawsp:PropertySet>
        </orawsp:Config>
    </orawsp:bindings>
  </custom:custom-executor>
</orasp:Assertion>
```

Sample Composite for Receiving the Encrypted Message

The following sample is a composite application with a BPEL process used to demonstrate that it received the encrypted value.

Example 3–11   Sample Composite

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SOA Modeler version 1.0 at [5/10/10 9:33 AM]. -->
<composite name="CustomEncryptionComposite"
           revision="1.0"
           label="2010-05-10_09-33-01_807"
           mode="active"
           state="on"
           xmlns="http://xmlns.oracle.com/sca/1.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
           xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
           xmlns:ui="http://xmlns.oracle.com/soa/designer/">
  <import namespace="http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionComposite/ProcessCustomer"
          location="ProcessCustomer.wsdl" importType="wsdl"/>
  <import namespace="http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionExternalService/ExternalServiceBpel"
          location="ExternalServiceBpel.wsdl" importType="wsdl"/>
  <service name="processcustomer_client_ep"
           ui:wsdlLocation="ProcessCustomer.wsdl">
    <interface.wsdl interface="http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionComposite/ProcessCustomer#wsdl.interface(ProcessCustomer)"/>
    <binding.ws port="http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionComposite/ProcessCustomer#wsdl.endpoint(processcustomer
_client_ep/ProcessCustomer_pt)">
       <wsp:PolicyReference URI="SOA/CustomEncryption" orawsp:category="security"
                            orawsp:status="enabled"/>
    </binding.ws>
  </service>
  <component name="ProcessCustomer">
    <implementation.bpel src="ProcessCustomer.bpel"/>
  </component>
  <reference name="ExternalSvc" ui:wsdlLocation="ExternalServiceBpel.wsdl">
    <interface.wsdl interface="http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionExternalService/ExternalServiceBpel#wsdl.interface(ExternalSe
rviceBpel)"/>
    <binding.ws port="http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionExternalService/ExternalServiceBpel#wsdl.endpoint(externalser
vicebpel_client_ep/ExternalServiceBpel_pt)"
                location="externalservicebpel_client_ep.wsdl"
                soapVersion="1.1">
```

```
        <wsp:PolicyReference URI="SOA/CustomDecryption" orawsp:category="security"
                             orawsp:status="enabled"/>
        <property name="weblogic.wsee.wsat.transaction.flowOption"
                  type="xs:string" many="false">WSDLDriven</property>
      </binding.ws>
    </reference>
    <wire>
      <source.uri>processcustomer_client_ep</source.uri>
      <target.uri>ProcessCustomer/processcustomer_client</target.uri>
    </wire>
    <wire>
      <source.uri>ProcessCustomer/ExternalSvc</source.uri>
      <target.uri>ExternalSvc</target.uri>
    </wire>
</composite>
```

Sample External Service Composite

The following sample is a composite application used to represent an external service being called and show that it received the decrypted value.

Example 3–12   Sample External Service Composite

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SOA Modeler version 1.0 at [5/10/10 9:41 AM]. -->
<composite name="CustomEncryptionExternalService"
           revision="1.0"
           label="2010-05-10_09-41-00_810"
           mode="active"
           state="on"
           xmlns="http://xmlns.oracle.com/sca/1.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
           xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
           xmlns:ui="http://xmlns.oracle.com/soa/designer/">
  <import namespace="http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionExternalService/ExternalServiceBpel"
         location="ExternalServiceBpel.wsdl" importType="wsdl"/>
  <service name="externalservicebpel_client_ep"
           ui:wsdlLocation="ExternalServiceBpel.wsdl">
    <interface.wsdl interface="http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionExternalService/ExternalServiceBpel#wsdl.interface(ExternalSe
rviceBpel)"/>
    <binding.ws port="http://xmlns.oracle.com/CustomEncryption
_jws/CustomEncryptionExternalService/ExternalServiceBpel#wsdl.endpoint(externalser
vicebpel_client_ep/ExternalServiceBpel_pt)"/>
  </service>
  <component name="ExternalServiceBpel">
    <implementation.bpel src="ExternalServiceBpel.bpel"/>
  </component>
  <wire>
    <source.uri>externalservicebpel_client_ep</source.uri>
    <target.uri>ExternalServiceBpel/externalservicebpel_client</target.uri>
  </wire>
</composite>
```

3.3.2 Running the Sample

To use the Encryption and Decryption sample, perform the following steps:

1. Create the custom assertion policy file and custom assertion executor with the sample codes as described in Section 3.3.1, "Code Samples". These samples demonstrate the following key features:

   - Specify the properties encryption_key and debugFlag in the encrypt property set for the encryption custom assertion:

```
<orawsp:PropertySet orawsp:name="encrypt">
  <orawsp:Property orawsp:contentType="constant"
                   orawsp:name="encryption_key"
orawsp:type="string">
     <orawsp:Value>MySecretKey</orawsp:Value>
  </orawsp:Property>
  <orawsp:Property orawsp:contentType="constant"
                   orawsp:name="debugFlag" orawsp:type="string">
     <orawsp:Value>true</orawsp:Value>
  </orawsp:Property>
</orawsp:PropertySet>
```

   For more information, see Section 4.1, "Inputing Parameters to Custom Assertions".

   - Set the properties decryption_key and debugFlag in the property set decrypt for decryption custom assertion:

```
<orawsp:PropertySet orawsp:name="decrypt">
  <orawsp:Property orawsp:contentType="constant"
                   orawsp:name="decryption_key"
 orawsp:type="string">
     <orawsp:Value>MySecretKey</orawsp:Value>
  </orawsp:Property>
  <orawsp:Property orawsp:contentType="constant"
                   orawsp:name="debugFlag" orawsp:type="string">
     <orawsp:Value>true</orawsp:Value>
  </orawsp:Property>
</orawsp:PropertySet>
```
   For more information, see Section 4.1, "Inputing Parameters to Custom Assertions".

   - Extract the node specified by XPath expressions with defined namespaces from the custom assertion class:

```
NamespaceContext ctx = new NamespaceContext()
     {
       public String getNamespaceURI(String prefix)
       {
         return namespaces.get(prefix);
       }
       // Dummy implementation - not used
       public Iterator getPrefixes(String val)
       {
         return null;
       }
       // Dummy implemenation - not used
       public String getPrefix(String uri)
       {
         return null;
       }
     };
     XPathFactory xpathFact = XPathFactory.newInstance();
     XPath xpath = xpathFact.newXPath();
```

```
        xpath.setNamespaceContext(ctx);

        node = (Node)xpath.evaluate(xpathStr, payload, XPathConstants.NODE);
    }
```

For more information, see Section 4.4, "Accessing Parts of a Message Using XPath".

2. Create the JAR file as described in Section 2.2.4, "Step 4: Create the JAR File".

3. Add the custom policy to the policy store as described in Section 2.2.5, "Step 5: Add the Custom Policy to the Policy Store".

4. Update the classpath as described in Section 2.2.6, "Step 6: Deploy the Custom Assertion".

5. Attach the custom policy to the Web service by any one of the methods described in Section 2.2.7, "Step 7: Attach the Custom Policy to a Web Service".

3.4 Authentication Custom Assertion Sample

The authentication custom assertion sample is used to authenticate a user using WebLogic authentication providers. The credentials (user name and password) are read from incoming SOAP message and are authenticated against WebLogic authentication provider using a custom login module.

The following sections provide samples that demonstrate how to code the custom assertion and describe the sample in detail:

■ Section 3.4.1, "Code Samples"

■ Section 3.4.2, "Running the Sample"

3.4.1 Code Samples

This section includes the following sample codes to create and implement the custom assertion:

■ "Sample Custom Assertion Executor" on page 3-21

■ "Sample Policy File" on page 3-26

■ "Sample Authentication Files" on page 3-26

Sample Custom Assertion Executor

Example 3–13 describes a sample executor that invokes the custom login module to perform the authentication.

Example 3–13   Sample Custom Assertion Executor

```
package sampleAssertion;
/**
 * <Description>
 * <p>
 * CustomAuthExecutor class. This class is invoked for custom_auth_policy
 * This class expects that wss_username_token_client_policy is attached to the
* calling client
 * This class fetches credentials from incoming SOAP message and performs
 * authentication against a configured login module as specified by the Login
* Config file.
```

```
 * </p>
 * </Description>
 *
 */
import java.util.Iterator;

import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPHeaderElement;
import javax.xml.soap.SOAPMessage;

import oracle.wsm.common.sdk.IContext;
import oracle.wsm.common.sdk.IMessageContext;
import oracle.wsm.common.sdk.IResult;
import oracle.wsm.common.sdk.Result;
import oracle.wsm.common.sdk.SOAPBindingMessageContext;
import oracle.wsm.common.sdk.WSMException;
import oracle.wsm.policy.model.IAssertion;
import oracle.wsm.policy.model.ISimpleOracleAssertion;
import oracle.wsm.policyengine.impl.AssertionExecutor;

import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class CustomAuthExecutor extends AssertionExecutor {

    public void init(IAssertion assertion,
        oracle.wsm.policyengine.IExecutionContext econtext,
        oracle.wsm.common.sdk.IContext context) {
        this.assertion = assertion;
        this.econtext = econtext;
    }

    public oracle.wsm.policyengine.IExecutionContext getExecutionContext() {
        return this.econtext;
    }

    public boolean isAssertionEnabled() {
        return ((ISimpleOracleAssertion) this.assertion).isEnforced();
    }

    public String getAssertionName() {
        return this.assertion.getQName().toString();
    }


    /**
     * <p>
     * This method is invoked for each request/response.
     * This method retrieves credentials from incoming soap message and
     * authenticates
     * them against a configured login module
     * </p>
     * @param context
     * @return IResult
     * @exception WSMException
     */
```

```
@Override
public IResult execute(IContext context) throws WSMException {
    oracle.wsm.common.sdk.IMessageContext.STAGE stage =
((oracle.wsm.common.sdk.IMessageContext) context)
        .getStage();
    IResult result = new Result();

    if (stage == IMessageContext.STAGE.request) {
        try {
            SOAPBindingMessageContext soapMsgCtxt= (SOAPBindingMessageContext)
            context;
            //Intercepts the incoming SOAP message
            SOAPMessage soapMessage = soapMsgCtxt.getRequestMessage();

            MyCallbackHandler callbackhandler = new MyCallbackHandler();

            //initialize CallbackHandler instance which is passed to
            //LoginModule implementation
            initializeCallbackHandler(callbackhandler, soapMessage);


            //In order to authenticate a user, you first need a
            //javax.security.auth.login.LoginContext.
            /**

            * Following parameters are passed to LoginContext
            * 1. name - LoginContext uses the name as the index into the JAAS
            * login configuration file to determine which LoginModules should
            * be used. Such an entry specifies the class(es) that
            * implement the desired underlying authentication
            * technology(ies). The class(es) must implement
            *  the LoginModule interface, which is in the
            * javax.security.auth.spi package.
            * In our case CustomLoginModule refers to
            * sampleAssertion.loginmodule.CustomLoginModule
            *
            * 2. Subject - the subject (javax.security.auth.Subject) to
            * authenticate. LoginContext passes the Subject
            * object to configured LoginModules so they may perform
            * additional authentication and update the Subject.
            *
            * 3. CallbackHandler instance -
            * javax.security.auth.callback.CallbackHandler  object is used by
            * LoginModules to
            * communicate with the user. LoginContext passes the
            * CallbackHandler object to configured LoginModules
            *  so they may communicate with the user. An application
            * typically provides its own CallbackHandler implementation.
            *
            **/


            // Obtain a LoginContext, needed for authentication.
            // Tell it to use the LoginModule implementation
            // specified by the entry named "CustomLoginModule" in the
            // JAAS login configuration file and to also use the
            // specified CallbackHandler.
            Subject subject = new Subject();
            LoginContext lc = new LoginContext("CustomLoginModule", subject,
            callbackhandler);
```

```
                        /**Once the caller has instantiated a LoginContext, it invokes the
                        * login method to authenticate a Subject
                        * The LoginContext instantiates a new empty
                        * javax.security.auth.Subject object (which represents the user or
                        * service being
                        * authenticated).
                        * The LoginContext constructs the configured LoginModule (in our
                        * case CustomLoginModule) and initializes it with this new Subject
                        * and
                        * MyCallbackHandler.
                        * The SampleLoginModule will utilize the MyCallbackHandler to
                        * obtain the user name and password.
                        * If authentication is successful, the CustomLoginModule
                        * populates the Subject with a Principal representing the user.
                        *
                         **/
                        lc.login();

                      //authenticated Subject can be retrieved by calling the
                       // LoginContext's getSubject
                        //method
                         subject = lc.getSubject();

            System.out.println("Authentication succeeded");

                         //sets result to succeeded if authentication succeeds
             result.setStatus(IResult.SUCCEEDED);
             } catch (LoginException e) {
                         //in case there is a failure in authentication sets result to
                         // failed state and
                         // throw Failed authentication exception
                          result.setStatus(IResult.FAILED);
                          throw new WSMException(
                              WSMException.FAULTCODE_QNAME_FAILED_AUTHENTICATION, e);
                 }
             }

             return result;
         }

                       /**
                        * Retrieves credentials from incoming SOAP message and
                        * sets them into call back handler, which is than passed to
                        * Login module
                        * class
                        * via initialize(Subject, Callbackhandler,..) method of
                        *  LoginModule
                        * @param SOAPMessage
                        * @param callbackhandler
                        *
                        */
         private void initializeCallbackHandler(MyCallbackHandler callbackhandler,
        SOAPMessage soapElement) {
             try {
                 SOAPHeader header = soapElement.getSOAPPart().getEnvelope()
                     .getHeader();
                 SOAPHeaderElement hdrElem = null;
                 Iterator iter = header.examineAllHeaderElements();
```

```
            while (iter.hasNext()) {
                hdrElem = (SOAPHeaderElement) iter.next();
                String localName = hdrElem.getLocalName();
                NodeList headerNodeList = hdrElem.getChildNodes();

                for (int i = 0; i < headerNodeList.getLength(); i++) {
                    Node kid = headerNodeList.item(i);
                    String kidName = kid.getLocalName();

                    if (kidName.equals("UsernameToken")) {
                        NodeList nodeList = kid.getChildNodes();

            /**check if incoming SOAP message contains UsernameToken
             *and retrieve credentials from it
            * The user name and password are set into callbackhandler
            * which are passed to Login module
             * for authentication
            **/
                        for (int j = 0; j < nodeList.getLength(); j++) {
                            String nodeName = nodeList.item(j).getLocalName();
                            String nodeValue = nodeList.item(j).getTextContent();

                            if (nodeName.equals("Username")) {
                                callbackhandler.setUserName(nodeValue);
                            }

                            if (nodeName.equals("Password")) {
                                callbackhandler.setPassword(nodeValue);
                            }
                        }
                    }
                }
            }

        } catch (SOAPException se) {
            System.out.println("caught SOAPException: " + se.getMessage());
        }

    }

    /**
     *Executes any task required after policy execution.
     *
     */
    @Override
    public oracle.wsm.common.sdk.IResult postExecute(
        oracle.wsm.common.sdk.IContext context) {
        IResult result = new Result();
        result.setStatus(IResult.SUCCEEDED);
        return result;
    }

    @Override
    public void destroy() {
    }

}
```

Sample Policy File

Example 3–14 is a custom assertion that invokes the executor described in Example 3–13 which is used to authenticate users against WebLogic authentication provider.

Example 3–14   Sample Policy File

```
<?xml version = '1.0'?>
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
 xmlns:oralgp="http://schemas.oracle.com/ws/2006/01/loggingpolicy"
 xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
orawsp:provides="{http://docs.oasis-open.org/ns/opencsa/sca/200903}authentication,
 {http://docs.oasis-open.org/ns/opencsa/sca/200903}clientAuthentication,
 {http://docs.oasis-open.org/ns/opencsa/sca/200903}clientAuthentication.message,
 {http://schemas.oracle.com/ws/2006/01/policy}token.usernamePassword"
 orawsp:status="enabled" xmlns="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-uti
lity-1.0.xsd" wsu:Id="custom_auth_policy"
orawsp:displayName="i18n:oracle.wsm.resources.policydescription.PolicyDescription
Bundle_oracle/custom_auth_policy_PolyDispNameKey"
 xmlns:orasp="http://schemas.oracle.com/ws/2006/01/securitypolicy"
orawsp:description="i18n:oracle.wsm.resources.policydescription.PolicyDescription
Bundle_oracle/custom_auth_policy_PolyDescKey" orawsp:attachTo="binding.server"
 Name="oracle/custom_auth_policy"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" orawsp:category="security"
 orawsp:local-optimization="off">
  <orasp:custom-auth-assertion orawsp:Silent="false" orawsp:name="Custom auth"
 orawsp:Enforced="true" orawsp:category="security/authentication">
      <orawsp:bindings>

<orawsp:Implementation>sampleAssertion.CustomAuthExecutor</orawsp:Implementation>
      </orawsp:bindings>
     </orasp:custom-auth-assertion>
</wsp:Policy>
```

Sample Authentication Files

The following code samples demonstrate the implementation of authentication:

- Example 3–15 describes a Login configuration file that contains an entry specifying the Login Module that is to be used to do the user authentication.

- Example 3–16 is a sample CustomLoginModule class that is specified by the LoginConfig file in Example 3–15. It provides an implementation of javax.security.auth.spi.LoginModule and authenticates a user against Weblogic authentication provider.

- Example 3–17 is a sample SimpleLoginModule class that provides an implementation of javax.security.auth.spi.LoginModule, and is described for illustration purposes only.

- Example 3–18 is a sample CallbackHandler class that provides an implementation of javax.security.auth.callback.Callback interface.

Example 3–15   Sample Configuration File

```
/** Login Configuration for the Sample Application
JAAS authentication is performed in a pluggable fashion, so Java applications can
 remain independent from
underlying authentication technologies. Users can plugin there custom loginmodule
```

implementations which
can integrate with corresponding authentication provider.

The name for an entry in a login configuration file is the name that applications
 use to refer to the entry when
they instantiate a LoginContext. The specified LoginModules (described below) are
 used to control the authentication process.

As part of this sample we are using CustomLoginModule and SimpleLoginModule
1. CustomLoginModule - integrates with weblogic authentication provider
2. SimpleLoginModule - simply returns

You can also provide implementation of javax.security.auth.login.Configuration
 interface
Refer
http://download.oracle.com/javase/1.4.2/docs/api/javax/security/auth/login/Config
uration.html and
http://download.oracle.com/javase/1.4.2/docs/guide/security/jaas/tutorials/LoginC
onfigFile.html for more details.

**/

CustomLoginModule {
    sampleAssertion.loginmodule.CustomLoginModule required debug=true;
};

SimpleLoginModule {
  sampleAssertion.loginmodule.SimpleLoginModule required debug=true;
};

Example 3–16   Sample Implementation of javax.security.auth.spi.LoginModule

```
package sampleAssertion.loginmodule;
/**
 * <Description>
 * <p>
 * CustomLoginModule class implements the javax.security.auth.spi.LoginModule
 * interface. CustomLoginModule class is specified
 * by the login configuration file i.e Login.config file.
 * This class authenticates user against weblogic authentication provider.
 * If authentication is successful, the CustomLoginModule associate Principals and
 * Credentials with the authenticated Subject
 *
 * Users can create there own custom login modules
 * by implementing <code>javax.security.auth.spi.LoginModule</code> interface
 * </p>
 * </Description>
 *
 */
import java.security.AccessController;
import java.security.PrivilegedAction;
import java.util.Map;

import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;

import weblogic.security.services.Authentication;
```

```
public class CustomLoginModule implements LoginModule
{
    private Subject authenticatorSubject;
    private CallbackHandler callbackHandler;
    private boolean loginSucceeded;

    /**
     * Initialize this LoginModule.
     *
     * <p> This method is called by the <code>LoginContext</code> to initialize
     * the <code>LoginModule</code> with the relevant information.
     * <p>
     *
     * @param subject the - <code>javax.security.auth.Subject</code> to be
     * authenticated.
     *
     * @param callbackHandler - instance of
     * <code>javax.security.auth.callback.CallbackHandler</code>
     *    the CallbackHandler object is used by LoginModules to retrieve the
     * information set by the user
     *    e.g user name / password.
     *
     * @param sharedState state shared with other configured LoginModules. <p>
     *
     * @param options options specified in the login Configuration for this
     *  particular LoginModule.
     *
     **/
    @Override
    public void initialize(Subject subject, CallbackHandler callbackHandler,
Map<String, ?> sharedState, Map<String, ?> options)
    {
        this.authenticatorSubject = subject;
        this.callbackHandler = callbackHandler;
    }


    @Override
    /**
     * Method to authenticate a <code>Subject</code>.
     *
     * <p> The implementation of this method authenticates
     * a <code>Subject</code> using weblogic authentication provider
     * <p>
     *
     * @exception LoginException if the authentication fails
     *
     * @return true if the authentication succeeded
     */
    public boolean login() throws LoginException
    {
        /**authenticates user against weblogic authentication provider and sets
         * loginSucceeded
         * flag to true in case of successful authentication
         */
        this.authenticatorSubject = authenticate(this.callbackHandler);
        loginSucceeded = true;
        return loginSucceeded;
    }
```

```
/**
 * authenticates using weblogic authentication provider
 * @param CallbackHandler
 * @return authenticated Subject
 * @throws LoginException
 */
private Subject authenticate(CallbackHandler cbh) throws LoginException {
    try {
        return Authentication.login(cbh);
    } catch (LoginException e) {
        throw new LoginException("Authentication Failed"+e.getMessage());
    }
}


@Override
/**
 * Method to commit the authentication process.
 *
 * <p> This method is called if the LoginContext's
 * overall authentication succeeded
 * <p>
 *
 * @exception LoginException if the commit fails
 *
 * @return true if this method succeeded, or false if this
 *          <code>LoginModule</code> should be ignored.
 */
public boolean commit() throws LoginException {
    if (this.authenticatorSubject != null) {
        addToSubject(this.authenticatorSubject);
        return true;
    } else {
        return false;
    }
}
/**
 * Method to abort the authentication process.
 *
 * <p> This method is called if the LoginContext's
 * overall authentication failed.
 * <p>
 *
 * @exception LoginException if the abort fails
 *
 * @return true if this method succeeded, or false if this
 *          <code>LoginModule</code> should be ignored.
 */
public boolean abort() throws LoginException {
    return this.loginSucceeded && logout();
}


/**
 * Method which logs out a <code>Subject</code>.
 *
 * <p>An implementation of this method might remove/destroy a Subject's
 * Principals and Credentials.
 * <p>
 *
 * @exception LoginException if the logout fails
```

```
     *
     * @return true if this method succeeded, or false if this
     *            <code>LoginModule</code> should be ignored.
     */
    public boolean logout() throws LoginException
    {
        if (this.authenticatorSubject != null) {
            removeFromSubject(this.authenticatorSubject);
        }
        this.loginSucceeded = false;
        return true;
    }

    /**
     * associates relevant Principals and Credentials with the Subject located in
     * the LoginModule
     * @param Subject
     */
    protected void addToSubject(final Subject sub) {
        if (sub != null) {
            AccessController.doPrivileged(new PrivilegedAction<Object>() {
             public Object run() {

             authenticatorSubject.getPrincipals().addAll(sub.getPrincipals());
 authenticatorSubject.getPrivateCredentials().addAll(sub.getPrivateCredentials());
authenticatorSubject.getPublicCredentials().addAll(sub.getPublicCredentials());
                        return null;
                    }
                });
        }
    }

    /**
     * removes Principals and Credentials from the subject
     * @param Subject
     */
    private void removeFromSubject(final Subject sub) {
        if (sub != null) {
            AccessController.doPrivileged(new PrivilegedAction<Object>() {
             public Object run() {
 authenticatorSubject.getPrincipals().removeAll(sub.getPrincipals());
 authenticatorSubject.getPrivateCredentials().removeAll(sub.getPrivateCredentials(


));
authenticatorSubject.getPublicCredentials().removeAll(sub.getPublicCredentials());
                        return null;
                    }
                });
        }
    }
}
```

Example 3–17   Sample Implementation of javax.security.auth.spi.LoginModule

```
/**
 * <Description>
 * <p>
 * SimpleLoginModule class implements the LoginModule interface. SimpleLoginModule
 * class is specified
```

```
 * by the login configuration file i.e Loginconfig file.
 * This class simply returns true resulting in successful authentication.
 *
 * This class is shown for illustration purpose only, users can integrate it with
 * there custom authentication provider
 * </p>
 * </Description>
 *
 */
package sampleAssertion.loginmodule;

import java.util.Map;

import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;

public class SimpleLoginModule implements LoginModule {

    @Override
    public boolean abort() throws LoginException {
        return false;
    }

    @Override
    public boolean commit() throws LoginException {
        return true;
    }

    @Override
    public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map<String, ?> sharedState, Map<String, ?> options) {

    }

    @Override
    /**
     * This method simply returns true and results in successful authentication
     * Users can integrate it using there custom authentication provider
     */
    public boolean login() throws LoginException {
        System.out.println("Inside SimpleLoginModule");
        return true;
    }

    @Override
    public boolean logout() throws LoginException {
        return false;
    }


}
```

Example 3–18   Sample Callback Handler

```
package sampleAssertion;
/**
 * <Description>
 * <p>
```

```
 * MyCallbackHandler class implements the
 *  <code>javax.security.auth.callback.CallbackHandler </code> interface.
 * An application implements its owm implementation of CallbackHandler.
 * An instance of CallbackHandler is passed as an argument to the LoginContext
 * instantiation.
 * The LoginContext forwards the CallbackHandler directly to the underlying
 * LoginModules
 * so that they may interact with the application to retrieve specific
 * authentication data,
 * such as usernames and passwords.
 * </p>
 * </Description>
 *
 */
import java.io.IOException;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.UnsupportedCallbackException;

public class MyCallbackHandler implements CallbackHandler {

    private String userName = null;

    private String password = null;

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    /**
     * sets user name and password into callback
     */
    public void handle(Callback[] callbacks) throws IOException,
        UnsupportedCallbackException {

        for (int i = 0; i < callbacks.length; i++) {
            Callback c = callbacks[i];
            if (c instanceof NameCallback) {
                ((NameCallback) c).setName(this.userName);
            } else if (c instanceof PasswordCallback) {
                char[] password = this.password.toCharArray();
                ((PasswordCallback) c).setPassword(password);
            } else {
                throw new UnsupportedCallbackException(callbacks[i],
                    "Unrecognized Callback");
            }
        }

    }

}
```

3.4.2 Running the Sample

To use the authentication custom assertion sample, perform the following steps:

1. Create the custom assertion and custom assertion executor with the sample codes as described in Section 3.4.1, "Code Samples". These samples demonstrate the following key features:

    - Define the custom assertion implementation class:

        ```
            <orawsp:bindings>
        <orawsp:Implementation>sampleAssertion.CustomAuthExecutor
        </orawsp:Implementation>
            </orawsp:bindings>
        ```

        For more information, see Section A.2.4, "orawsp:Implementation".

    - View the execution stage in the custom assertion executor:

        ```
        oracle.wsm.common.sdk.IMessageContext.STAGE stage =
          ((oracle.wsm.common.sdk.IMessageContext) context).getStage();
        ```

        For more information, see Section 4.3, "Accessing Request, Response, and Fault Message Objects".

    - In order to authenticate a user, you first need a javax.security.auth.login.LoginContext. Obtain a LoginContext. Use the LoginModule implementation specified by the entry named "CustomLoginModule" in the JAAS login configuration file and use the specified CallbackHandler as follows:

        ```
        Subject subject = new Subject();
        LoginContext lc = new LoginContext("CustomLoginModule", subject,
        callbackhandler);
        ```

        Once the caller has instantiated a LoginContext, it invokes the login method to authenticate a subject. If authentication is successful, the CustomLoginModule populates the Subject with a Principal representing the user.

        For more information, see http://download.oracle.com/javase/1.5.0/docs/guide/security/jgss/tutorials/index.html.

        initializerCallbackHandler (Example 3–13) is used to check if incoming SOAP message contains UsernameToken, retrieve credentials from the incoming SOAP message and pass them to Login module class

    - In the event of success, set the result to success:

        ```
        result.setStatus(IResult.SUCCEEDED);
        ```

    - In the event of failure, set the fault that caused the request execution to fail:

        ```
        result.setStatus(IResult.FAILED);
                        throw new WSMException(
                            WSMException.FAULTCODE_QNAME_FAILED_AUTHENTICATION, e);
        ```

        For more information, see Section 4.8, "Handling Exceptions in Custom Assertions".

2. Create the JAR file as described in Section 2.2.4, "Step 4: Create the JAR File".

3. Add the custom policy to the policy store as described in Section 2.2.5, "Step 5: Add the Custom Policy to the Policy Store".

4. Update the classpath as described in Section 2.2.6, "Step 6: Deploy the Custom Assertion".

5. Attach the custom policy to the Web service by any one of the methods described in Section 2.2.7, "Step 7: Attach the Custom Policy to a Web Service".

6. Create a client for the Web service. This sample custom policy uses wss_username_token_client_policy on the client side. Attach the wss_username_token_client_policy to the client.

7. Create a configuration file and the class files that implement authentication as described in "Sample Authentication Files" on page 3-26. For more information on the configuration file, refer to the JAAS API at http://download.oracle.com/javase/1.5.0/docs/guide/security/jgss/tutorials/index.html.

8. Specify the Login configuration file created in Example 3–15. You can specify the file in any one of the following ways:

   - Edit the script you use to start WebLogic Server to add the following option after the java command and restart the server:

     -Djava.security.auth.login.config==<path of LoginConfig file>

   - The Java security properties file is located in the file named <JAVA_HOME>/lib/security/java.security, where <JAVA_HOME> refers to the directory where the JDK was installed. In this file, change the value of the "login.configuration.provider" security property to the fully qualified name of the Login configuration file.

9. Invoke request from the client. The user name and password set into client will be authenticated against configured login module which integrates with Weblogic authentication provider.

4

Implementing Advanced Features in Custom Assertions

The Java API Reference for Oracle Web Services Manager specifies packages, interfaces, and methods you use to implement advanced features in custom assertions. This chapter describes how to use the API to implement some common features. This chapter describes the use of APIs, and exception handling. It includes the following sections:

- Inputing Parameters to Custom Assertions

- Accessing Oracle WSM Context Properties

- Accessing Request, Response, and Fault Message Objects

- Accessing Parts of a Message Using XPath

- Accessing Certificates Used by Container for SSL

- Accessing Transport Properties

- Accessing CSF Keys

- Handling Exceptions in Custom Assertions

4.1 Inputing Parameters to Custom Assertions

You can input parameters to custom assertions using the following interfaces and methods:

- IAssertionBindings

- IConfig

- IPropertySet

- getBindings

- getConfigs

- getPropertySets

- getPropertyByName

- getValue

To input parameters to custom assertions, perform the following steps:

1. Specify parameters as properties inside your custom assertion. In this example, the orawsp:PropertySet with the name valid_ips defines a group of properties. The

orawsp:Property element defines a single property. orawsp:Value defines a list of valid values for the property.

```
<orawsp:PropertySet orawsp:name="valid_ips">
    <orawsp:Property orawsp:name="valid_ips" orawsp:type="string"
 orawsp:contentType="constant">
     <orawsp:Value>140.87.6.143,10.178.93.107</orawsp:Value>
     </orawsp:Property>
</orawsp:PropertySet>
```

2. Access the parameters inside the custom executor for the corresponding policy. For example, the following code in the execute method of custom assertion's executor class accesses the property valid_ips:

```
IAssertionBindings bindings =
 ((SimpleAssertion)(this.assertion)).getBindings();
IConfig config = bindings.getConfigs().get(0);
IPropertySet propertyset = config.getPropertySets().get(0);
String valid_ips = propertyset.getPropertyByName("valid_ips").getValue();
```

4.2 Accessing Oracle WSM Context Properties

Oracle WSM context properties are accessed using the IMessageContext interface.

You access parameters to custom assertions using the following interfaces and methods:

- IMessageContext
- getServiceURL
- getProperty
- getAllProperty

To access Oracle WSM context properties, perform the following steps:

1. To access Oracle WSM context properties inside the custom assertion executor, use the IMessageContext interface. For example:

```
IMessageContext messagecontext = (IMessageContext) context;
messagecontext.getServiceURL();
```

2. To access the value of a specific property inside the custom assertion executor, use the IMessageContext interface. For example:

```
messagecontext.getProperty("<property name>");
```

3. To access all the properties that are used during execution from inside the custom assertion executor, use the IMessageContext interface. For example:

```
msgContextProperties = messagecontext.getAllProperties();
```

4.3 Accessing Request, Response, and Fault Message Objects

The Oracle WSM custom security assertion has three stages: request, response, and fault.

- The request stage occurs when a client has made a request and that request is in the process of being delivered to its destination.
- The response stage occurs after the destination has processed the message and is in the process of returning a response.

- The fault stage occurs in the event of a fault.

The contextual information (such as stages and messages) is passed using context properties and can be obtained by the IMessageContext interface. The following interfaces and methods are used to access context properties:

- IMessageContext

- getStage

- getRequestMessage

- getResponseMessage

To access request, response and fault messages, and stages, perform the following:

1. To access the stage, use the following code within the custom assertion executor:

```
IMessageContext.STAGE stage = ((IMessageContext) iContext).getStage();
        if (stage == IMessageContext.STAGE.request) {
                //handle request
        }

        if (stage == IMessageContext.STAGE.response) {
              //handle response
        }

        if (stage == IMessageContext.STAGE.fault) {
            //handle fault conditions
        }
```

2. To retrieve the SOAP request message, use the same context oracle.wsm.common.sdk.IMessageContext, as shown in the following example:

```
oracle.wsm.common.sdk.SOAPBindingMessageContext soapMsgCtxt=
 (oracle.wsm.common.sdk.SOAPBindingMessageContext) context;
javax.xml.soap.SOAPMessage soapMessage = soapMsgCtxt.getRequestMessage();
```

3. To retrieve the SOAP response message, use the same context oracle.wsm.common.sdk.IMessageContext, as shown in the following example:

```
oracle.wsm.common.sdk.SOAPBindingMessageContext soapMsgCtxt=
 (oracle.wsm.common.sdk.SOAPBindingMessageContext) context;
javax.xml.soap.SOAPMessage soapMessage = soapMsgCtxt.getResponseMessage ();
```

4.4 Accessing Parts of a Message Using XPath

You can access parts of a SOAP message using XPath expression inside your custom policy executor.

In the following SOAP message example, the node arg0 has the value john:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
    <S:Body>
      <ns2:echo xmlns:ns2="http://project1/">
       <arg0>john</arg0>
      </ns2:echo>
    </S:Body>
</S:Envelope>
```

In XPath, there are seven types of nodes: element, attribute, text, namespace, processing-instruction, comment, and document nodes. XPath uses path expressions

to select nodes in an XML document. Table 4–1 describes some examples of XPath expressions.

Table 4–1   Examples of XPath Expressions

Expression	Description
/S:Envelope	Selects from the root element S:Envelope.
/S:Envelope/S:Body	Selects all S:Body elements that are children of S:Envelope
//S:Body	Selects all S:Body elements no matter where they are in a document

For example, to access the value of the node arg0, the XPath expression for arg0 in the above SOAP message is defined as:

```
//xpath expression that will be used to identify the node arg0
    String xpathStr = "/S:Envelope/S:Body/ns2:echo/arg0";
```

Define the namespaces for any namespace referenced by the XPath expression and add them to the namespace context. For example:

```
final DefaultNamespaceContext nsContext = new DefaultNamespaceContext();
nsContext.addEntry("S", "http://schemas.xmlsoap.org/soap/envelope/");
nsContext.addEntry("ns2", "http://project1/");

XPathFactory xpathFact = XPathFactory.newInstance();
XPath xpath = xpathFact.newXPath();
xpath.setNamespaceContext(nsContext);
```

Retrieve the value of node using the evaluate method. For example:

```
    //This will return node arg0 from SOAP message, here soapElement is
// org.w3c.dom.Elemet  representation of SOAP message
    org.w3c.dom.Node inputNode = (Node)xpath.evaluate(xpathStr, soapElement,
 XPathConstants.NODE);
```

4.5 Accessing Certificates Used by Container for SSL

To retrieve certificates for SSL, perform the following steps:

1. Retrieve SOAP response message using the same context oracle.wsm.common.sdk.IMessageContext, as shown in the following example:

    ```
    oracle.wsm.common.sdk.SOAPBindingMessageContext soapMsgCtxt=
     (oracle.wsm.common.sdk.SOAPBindingMessageContext) context;
    ```

2. Use the HttpServletRequest interface to retrieve the servlet request information.

    ```
    javax.servlet.http.HttpServletRequest request =
     (javax.servlet.http.HttpServletRequest)
     soapMsgCtxt.getTransportObject(ITransportContext.HTTP_SERVLETREQUEST_OBJ_
    NAME);
    ```

3. Access the attributes of an X.509 certificate. For example:

    ```
    X509Certificate[] certificates = (X509Certificate[])
     request.getAttribute(oracle.wsm.security.util.SecurityConstants.SSL_PEER_
    CERTIFICATES);
    ```

> **Note:** To use the javax.servlet.http.HttpServletRequest interface, the following JAR file must be included in the classpath: modules/javax.servlet_1.0.0.0_2-5.jar.

4.6 Accessing Transport Properties

To access transport objects such as HTTP requests and responses, perform the following:

1. Retrieve SOAP response message using the same context oracle.wsm.common.sdk.IMessageContext, as shown in the following example:

```
oracle.wsm.common.sdk.SOAPBindingMessageContext soapMsgCtxt=
 (oracle.wsm.common.sdk.SOAPBindingMessageContext) context;
```

2. Use the HttpServletRequest interface to retrieve servlet request information of the HTTP servlets, as shown in the following example:

```
//to retrieve http servlet request object
 HttpServletRequest request = (HttpServletRequest) soapMsgCtxt
         .getTransportObject(ITransportContext.HTTP_SERVLETREQUEST_OBJ_
NAME);
```

3. Use the HttpServletRequest interface to retrieve servlet response information of the HTTP servlets, as shown in the following example:

```
//to retrieve http servlet response object
 HttpServletResponse response = (HttpServletResponse) soapMsgCtxt
         .getTransportObject(ITransportContext.HTTP_SERVLETRESPONSE_OBJ_
NAME);
```

> **Note:** To use the javax.servlet.http.HttpServletRequest interface, the following JAR file must be included in the classpath: modules/javax.servlet_1.0.0.0_2-5.jar.

4.7 Accessing CSF Keys

You can use credential store framework (CSF) to manage the credentials securely, and store, retrieve, and maintain credentials.

To configure and use CSF, perform the following steps:

1. Configure CSF in jps-config.xml.

   For details on configuring credential store using WLST, see "Configuring the Credential Store Using WLST" in *Oracle Fusion Middleware Securing WebLogic Web Services for Oracle WebLogic Server*.

2. You can add, update, or retrieve CSF keys from CSF inside your custom assertion executor. Example 4–1 and Example 4–2 provide sample codes that can be used to access CSF keys from credential store.

Example 4–1   Accessing CSF Keys

```
final String mapName = "oracle.wsm.security";
final csfKey = "user.credentials";
final oracle.security.jps.service.credstore.PasswordCredential userCreds =
getCredentialsFromCSF(mapName, csfKey);
```

```
                     if (userCreds != null) {
                         System.out.println("name:"+ userCreds.getName());
                         System.out.println("name:"+ new String(userCreds.getPassword()));
                                    }
```

Example 4–2   getCredentialsFromCSF method

```
private static oracle.security.jps.service.credstore.PasswordCredential
getCredentialsFromCSF(
        final String mapName, final String csfKey) {
        oracle.security.jps.service.credstore.PasswordCredential
        passwordCredential = null;
        try {
            if (csfKey != null) {
            final oracle.security.jps.service.credstore.CredentialStore credStore
            = getCredStore();
            if (credStore != null) {
            passwordCredential =
(oracle.security.jps.service.credstore.PasswordCredential)
java.security.AccessController
            .doPrivileged(new
java.security.PrivilegedExceptionAction<oracle.security.jps.service.credstore.
Credential>() {
             public
 oracle.security.jps.service.credstore.Credential run() throws Exception {
                return (credStore .getCredential(mapName, csfKey));
                            }
                    });
                } else {
                    // failure obtaining csf credentials
                }
            }
        } catch (final java.security.PrivilegedActionException ex) {
            //handle excpetion
        } catch (final oracle.security.jps.JpsException jpse) {
            //handle excpetion
        }
        return passwordCredential;
    }


    private static oracle.security.jps.service.credstore.CredentialStore
 getCredStore() throws oracle.security.jps.JpsException {
        oracle.security.jps.service.credstore.CredentialStore csfStore;
        oracle.security.jps.service.credstore.CredentialStore appCsfStore = null;
        oracle.security.jps.service.credstore.CredentialStore systemCsfStore =
null;

        final oracle.security.jps.internal.api.runtime.ServerContextFactory
 factory = (oracle.security.jps.internal.api.runtime.ServerContextFactory)
 oracle.security.jps.JpsContextFactory
            .getContextFactory();

        final oracle.security.jps.JpsContext jpsCtxSystemDefault =

factory.getContext(oracle.security.jps.internal.api.runtime.ServerContextFactory.S
cope.SYSTEM);
```

```
        final oracle.security.jps.JpsContext jpsCtxAppDefault = factory

.getContext(oracle.security.jps.internal.api.runtime.ServerContextFactory.Scope.AP
PLICATION);

        appCsfStore = (jpsCtxAppDefault != null) ? jpsCtxAppDefault

.getServiceInstance(oracle.security.jps.service.credstore.CredentialStore.class) :
 null;

        if (appCsfStore == null) {
            systemCsfStore = jpsCtxSystemDefault

.getServiceInstance(oracle.security.jps.service.credstore.CredentialStore.class);
            csfStore = systemCsfStore;
        } else {
            //use Credential Store defined in app-level jps-config.xml
            csfStore = appCsfStore;
        }
        return csfStore;
    }
```

Note: The following JAR files must be included in the classpath:
oracle.jps_11.1.1/jps-api.jar, oracle.jps_11.1.1/jps-unsupported-api.jar.

You must provide the necessary permissons to the custom policy executor in the
system-jazn-data.xml file. For example,

```
<grant>
        <grantee>
            <codesource>
                <url>file:{location of custom policy executor jar}</url>
            </codesource>
        </grantee>
        <permissions>
            <permission>

<class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
                <name>context=SYSTEM,mapName=*,keyName=*</name>
                <actions>*</actions>
            </permission>
        </permissions>
    </grant>
```

4.8 Handling Exceptions in Custom Assertions

Any exceptions during the execution of custom assertions must be handled by the
WSMException in the custom assertion executor.

```
IResult execute(IContext mcontext) throws WSMException
```

This method must always return a non-null IResult object. The status field indicates
success or failure or other state. The IResult.getFault() method is used to return the
detailed cause for failure and returns null in case of success.

As shown in the following example, exceptions arising from within the execute
method of custom assertion executor should first be wrapped in WSMException, the

execution status should be set to IResult.FAILED, and the generateFault method throws the WSMException.

```
IResult execute(IContext mcontext) throws WSMException {
   IResult result = new Result();
   try {
      ....
      .....

   } catch (Exception e) {
      WSMException wsmException = new WSMException(e);
      result.setStatus(IResult.FAILED);
      generateFault(wsmException);
   }
}
```

A

Schema Reference for Custom Assertions

This appendix provides the XML schema for reference when creating a WS-Policy file that contains custom Web service assertions. Sections include:

- Graphical Representation
- Element Descriptions

A.1 Graphical Representation

The following graphic describes the element hierarchy of the custom assertions in the WS-Policy file.

Figure A–1   Element Hierarchy of Custom Assertion
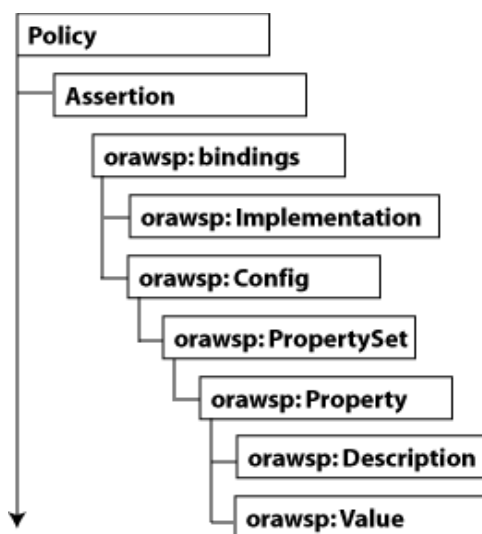
A.2 Element Descriptions

The following sections describe the elements in the custom assertion in more detail.

A.2.1 wsp:Policy

Groups nested policy assertions.

A.2.1.1 Attributes

The following table summarizes the Oracle extensions to the WS-Policy attributes.

Table A–1   Oracle Extensions to WS-Policy Attributes

Attribute	Description
attachTo	Policy subjects to which the policy can be attached. Valid values include:binding.client, binding.server, binding.any.
category	Category of the policy. Valid values include: security, mtom, wsrm, addressing, and management.
description	Description of the policy.
status	Status of the policy reference. Valid values include: enabled and disabled.

A.2.1.2 Example

```
<wsp:Policy xmlns="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:orasp="http://schemas.oracle.com/ws/2006/01/securitypolicy"
  orawsp:status="enabled"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-util
ity-1.0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  orawsp:category="security"
  orawsp:attachTo="binding.server"
  wsu:Id="ip_assertion_policy"
  xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  wsp:Name="oracle/ip_assertion_policy">
```

A.2.2 orasp:Assertion

Main element of the custom assertion.

A.2.2.1 Attributes

The following table summarizes the attributes of the orasp:Assertion element.

Table A–2   Attributes of <orasp:Assertion> Element

Attribute	Description
Optional	Flag that specifies whether the assertion is optional or required.
Silent	Flag that specifies whether the assertion is advertised. If set to true, the assertion is not advertised.
Enforced	Flag that specifies whether the assertion is currently enabled.
name	Name of the assertion.
description	Description of the assertion.
category	Category to which the assertion applies. Valid values include: security/authentication, security/msg-protection, security/authorization, security/logging, mtom, wsrm, addressing, and management.

A.2.2.2 Example

```
<orasp:ipAssertion orawsp:Silent="true" orawsp:Enforced="true"
orawsp:name="WSSecurity IpAssertion Validator"
```

```
orawsp:category="security/authentication">
...
</orasp:ipAssertion>
```

A.2.3 orawsp:bindings

The <oraswsp:bindings> element defines the bindings in the custom assertion.

A.2.3.1 Example

```
<orawsp:bindings>
...
</orawsp:bindings>
```

A.2.4 orawsp:Implementation

The <oraswsp:Implementation> element defines the custom assertion implementation class.

A.2.4.1 Example

```
<orawsp:Implementation>acme.security.wss.executor.WssUsernameTokenExecutor</orawsp
:Implementation>
```

A.2.5 orawsp:Config

The <oraswsp:Config> element defines the configuration for the custom assertion.

A.2.5.1 Attributes

The following table summarizes the attributes of the orawsp:Config element.

Table A–3   Attributes of <orawsp:Config> Element

Attribute	Description
name	Name of the configuration.
type	Category to which the configuration applies.
configType	Configuration type. Valid values include: declarative and programmatic. ■ declarative—Use deployment descriptors and configuration files to describe authentication and authorization requirements. ■ programmatic—Embed security enforcement within the application.

A.2.5.2 Example

```
<orawsp:Config orawsp:name="ipassertion" orawsp:configType="declarative">
```

A.2.6 orawsp:PropertySet

The <oraswsp:PropertySet> element groups nested properties.

A.2.6.1 Attributes

The following table summarizes the attributes of the orawsp:PropertySet element.

Table A–4   Attributes of <orawsp:PropertySet> Element

Attribute	Description
name	Name of the property set.

A.2.6.2 Example

```
<orawsp:PropertySet orawsp:name="valid_ips">
```

A.2.7 orawsp:Property

The <oraswsp:Property> element defines a single property.

A.2.7.1 Attributes

The following table summarizes the attributes of the orawsp:Property element.

Table A–5   Attributes of <orawsp:Property> Element

Attribute	Description
name	Name of the property.
type	Type of the property. For example, string.
contentType	Specifies whether the property is required and can be overridden. Valid values include:
	■ constant—Property is a constant value and cannot be overridden.
	■ required—Property is required and can be overridden.
	■ optional—Property is optional and can be overridden.
	For information about overriding policies, see "Attaching Client Policies Permitting Overrides" in Security and Administrator's Guide for Web Services.

A.2.7.2 Example

```
<orawsp:Property orawsp:name="valid_ips" orawsp:type="string"
 orawsp:contentType="constant">
```

A.2.8 orawsp:Description

The <oraswsp:Description> element provides a description of the property.

A.2.8.1 Example

```
<orawsp:Description>Valid IP Values</orawsp:Description>
```

A.2.9 orawsp:Value

The <oraswsp:Value> element provides a list of valid values for the property.

A.2.9.1 Example

```
<orawsp:Value>140.87.6.143,10.178.93.107</orawsp:Value>
```