BEA AquaLogic Enterprise Security™

Programming Security for Web Services

Version 2.5
Revised: December 2006

Copyright

Restricted Rights Legend

Third-Party Software License Agreement

Sun Microsystems, Inc.’s XACML implementation v2.0

This product includes Sun Microsystems, Inc.’s XACML implementation v2.0, which is governed by the following terms:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Sun Microsystems, Inc. or the names of contributors maybe used to endorse or promote products derived from this software without specific prior written permission.

This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You acknowledge that this software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

For all third-party software license agreements, see the 3rd_party_licenses.txt file, which is placed in the \ales25-admin directory when you install the AquaLogic Enterprise Security Administration Server.

Trademarks and Service Marks

1. Introduction and Roadmap

2. Introduction to the Web Services Security Service Module

3. Using the Web Services Client Authorization Cache

4. Using Failover with a Web Services Client

5. Web Services Interfaces

Index

CHAPTER 1

Introduction and Roadmap

The following sections describe the content and organization of this document:

Scope

This document provides an overview of the Web Services Security Service Module and the related standards and features it supports.

Documentation Audience

It is assumed that the reader understands Web Services technologies and has a general understanding of the Microsoft Windows or UNIX operating systems being used. This document is intended for the following audiences:

- Web Services Developers—Developers who are programmers who focus on developing Web services applications and who work with other engineering, quality assurance (QA), and database teams to implement security features. Web services developers have in-depth working knowledge of Web Services programming.

- Application Developers—Developers who focus on designing applications and work with other engineers, quality assurance (QA) technicians, and database teams to implement applications.
- Security Administrators—Administrators who are responsible for installing and configuring the AquaLogic Enterprise Security products and designing policy. Security administrators work with other administrators to implement and maintain security configurations, authentication and authorization schemes, and to set up and maintain access to deployed application resources. Security administrators have a general knowledge of security concepts and the AquaLogic Enterprise Security architecture.

Prerequisites for this Document

Prior to reading this guide, you should read the *Introduction to BEA AquaLogic Enterprise Security*. This document describes how the product works and provides conceptual information that is helpful to understanding the necessary installation components.

Additionally, BEA AquaLogic Enterprise Security includes many unique terms and concepts that you need to understand. These terms and concepts—which you will encounter throughout the documentation—are defined in the *Glossary*.

Guide to this Document

This document is organized as follows:

- Chapter 2, "Introduction to the Web Services Security Service Module,"provides an overview of the Web Services Security Service Module and the related standards and features it supports.
- Chapter 3, "Optimizing Web Services Performance with Caches," describes how to use the Web Services Client Authorization Cache to allow an application using the Web Services SSM to take advantage of in-process caching to achieve performance improvements when making authorization calls.
- Chapter 4, "Using Failover with a Web Services Client," describes how to develop a Web Services client that uses failover to connect to alternate Web Services SSM if the primary SSM is unreachable.
- Chapter 5, "Web Services Interfaces," describes the Web Services interfaces and the methods that each interface supports.

Related Information

The BEA corporate web site provides all documentation for BEA AquaLogic Enterprise Security. Other BEA AquaLogic Enterprise Security documents that may be of interest to the reader include:

- *WSDL Documentation for the Web Service Interfaces*—This document provides reference documentation for the Web Services Interfaces that are provided with and supported by this release of BEA AquaLogic Enterprise Security.
- *Administration and Deployment Guide*—This document provides step-by-step instructions for performing various administrative tasks.
- *Integrating ALES with Application Environments*—This document describes important tasks associated with integrating AquaLogic Enterprise Security into application environments.
- *Policy Managers Guide*—This document how to write access control policies for BEA AquaLogic Enterprise Security, and describes how to import and export policy data.
- *Installing Security Services Modules*—This document describes how to install ALES Security Services Modules, including the Web Services Security Service Module.
- *Developing Security Providers* —This document provides security vendors and security and application developers with the information needed to develop custom security providers.
- *Javadocs for Security Service Provider Interfaces*—This document provides reference documentation for the Security Service Provider Interfaces that are provided with and supported by this release of BEA AquaLogic Enterprise Security.
- *Programming Security for Java Applications*—The document describes how to implement security in Java applications. It include descriptions of the Security Service Application Programming Interfaces and programming instructions for implementing security in Java applications.
- *Javadocs for Java API*—This document provides reference documentation for the Java Application Programming Interfaces that are provided with and supported by this release of BEA AquaLogic Enterprise Security.

CHAPTER 2

Introduction to the Web Services Security Service Module

This section provides an introduction to AquaLogic Enterprise Security Web Services Security Service Module and describes interfaces clients use to interact with it.

This section covers the following topics:

Overview of Web Services

Web services are a special type of service that can be shared by and used as components of distributed Web-based applications. Web services interface with existing back-end applications, such as customer relationship management systems, order-processing systems, and so on.

Traditionally, software application architecture tended to fall into two categories: huge monolithic systems running on mainframes or client-server applications running on desktops. Although these architectures work well for the purpose the applications were built to address, they are closed and cannot be easily accessed by the diverse users of the Web. Thus the software industry has evolved toward loosely coupled service-oriented applications that interact dynamically over the Web. The applications break down the larger software system into smaller modular components, or shared services. These services can reside on different computers and can be implemented by vastly different technologies, but they are packaged and transported using

standard Web protocols, such as Extensible Markup Language (XML) and Hyper Text Transfer Protocol (HTTP), thus making them easily accessible by any user on the Web.

This concept of services is not new. Remote Method Invocation (RMI), Component Object Model (COM), and Common Object Request Broker Architecture (CORBA) are all service-oriented technologies. However, applications based on these technologies must be written using that particular technology, often as implemented by a particular vendor. This requirement typically hinders widespread acceptance of such services on the Web. To solve this problem, web services are defined to share the following properties that make them easily accessible from heterogeneous environments:

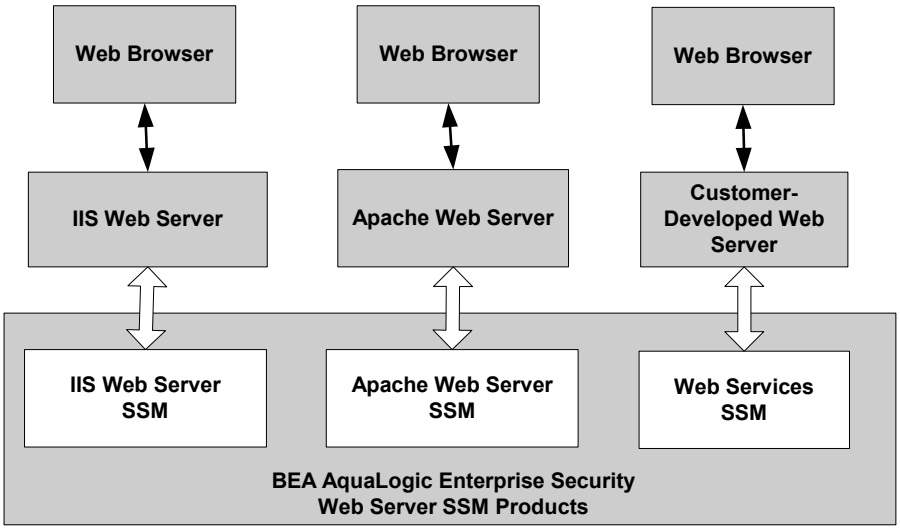
- Web services are accessed over the Web.
- Web services describe themselves using an XML-based description language.
- Web services communicate with clients (both end-user applications or other web services) through XML messages that are transmitted by standard Internet protocols, such as HTTP.

Product Overview

The BEA AquaLogic Enterprise Security provides three Web Server Security Service Modules (SSMs): the IIS Web Server SSM, the Apache Web Server SSM, and the Web Services SSM (see

Figure 2-1). This document only describes the Web Services SSM and the security service application programming interfaces (APIs) that it supports.

Figure 2-1 Web Server and Web Services SSMs



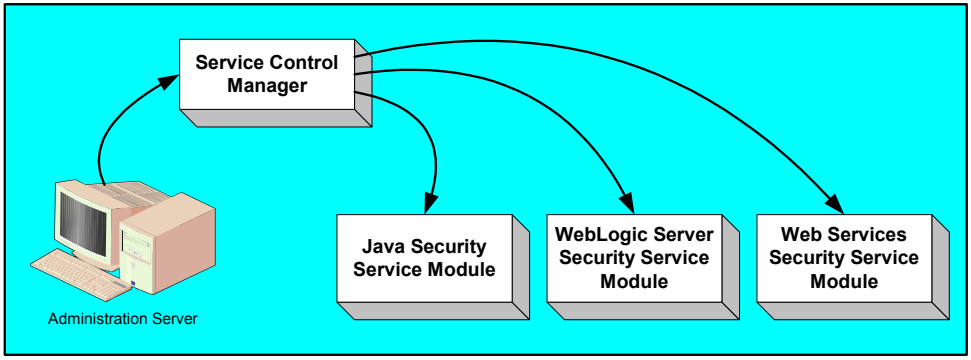
The following topics provide more information on the these products:

- "Web Services Environment" on page 2-3
- "Web Services Security Service Module" on page 2-6
- "Client Trust Model" on page 2-7
- "Deployment Model" on page 2-8
- "Usage Model" on page 2-9

Web Services Environment

Figure 2-2 shows the major components that make up the BEA AquaLogic Enterprise Security product environment.

Figure 2-2 AquaLogic Enterprise Security Product Environment



- Administration Server

  The Administration Server allows you to configure, deploy, and manage multiple Security Service Modules in a distributed environment. While the modules consume configuration data and then service security requests accordingly, the Administration Server allows you to configure and deploy security configuration information to the modules and modify that information as needed.

- Service Control Manager

  The Service Control Manager (SCM) is an essential component of the configuration provisioning mechanism and a key component of a fully-distributed security enforcement architecture.

  Note: In this release of AquaLogic Enterprise Security it is possible to deploy an SSM without the SCM. You can use the PolicyIX tool, described in P olicyIX in the *Administration Reference*, to communicate directly with the BLM and retrieve configuration data. The PolicyIX tool allows you to export configuration data (configured either through the ALES Administration Console or directly via the BLM API) for a given SSM to an XML file, and use it with the configured SSMs when the SCM is not available.

  A SCM is a machine agent that exposes a provisioning, or deployment, interface to the Administration Server to facilitate the management of a potentially large number of distributed Security Service Modules (SSMs). The SCM can receive and store meta-data updates, both full and incremental, initiated by the Administration Server.

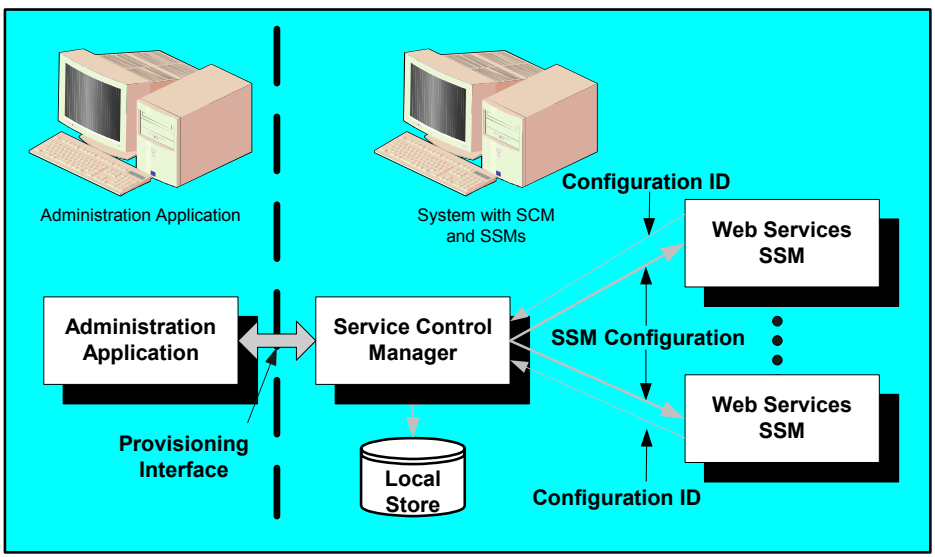
  The Administration Server uses this provisioning mechanism to distribute configuration data to each created instance of a SSM where it is consumed locally (see Figure 2-3). Each

instance of an SSM is assigned a unique configuration ID, which is registered with the SCM when the SSM is enrolled. The SCM uses the configuration ID when distributing and updating configuration data to each SSM instance to ensure that the correct data is distributed.

- Web Services Security Service Module

  The Web Services SSM provides an application programming interface (API) that allows security developers to write custom applications that invoke AquaLogic Enterprise Security services through SOAP. These interfaces support the most commonly required security functions and are organized into services that are logically grouped by functionality. You define and deploy the security configuration using the Administration Server. You can configure more than one instance of the Web Services SSM on a single machine; however, each instance must run in separate process. The number of machines in your network on which you can configure a Web Services SSM is unlimited (see Figure 2-3). After you deploy the initial security configuration to a Web Server SSM, it does not require any additional communication with the Service Control Manager (SCM) to perform runtime security functions. However, the SCM does maintain communication with each Web Services SSM instance so that it can distribute, or deploy, full and incremental security configuration and updates and updates to access control policies.

Figure 2-3 Deploying Security Configuration and Policy Data



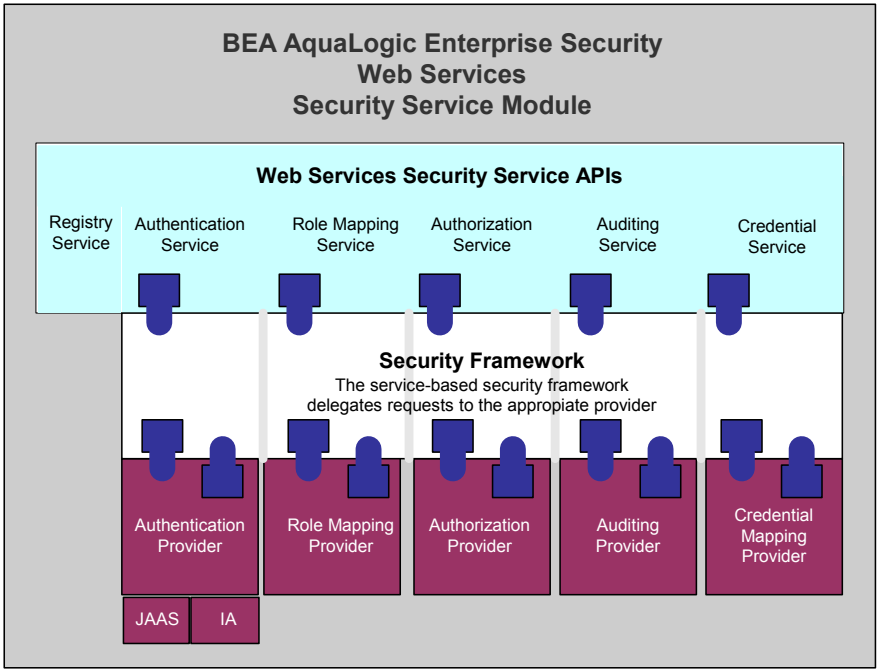
Web Services Security Service Module

The Web Services Security Service Module (SSM) provides five security service APIs: Authentication, Authorization, Auditing, Role Mapping, and Credential Mapping (see Figure 2-4). These APIs can be used to developed web services clients to access the AquaLogic Enterprise Security infrastructure and use it to make access control decisions for users attempting to access web server application resources. Once the web services client is implemented, it uses the Web Services SSM (which incorporates the Security Services APIs, the Security Framework, and the configured security providers) to make access control decisions for the web server to which it is connected. Then you can use the AquaLogic Enterprise Security Administration Server to configured and deploy a security configuration to protect the web server application resources. Thus, the Web Services SSM enables security administrators and web developers to perform security tasks for applications running on a web server. Additionally, you can use the

Web Services SSM to add information provided by the Security Framework (such as roles and response attributes) to the HTTP requests handled by the protected web server applications.

Figure 2-4 shows the components of the Web Services SSM.

Figure 2-4 Web Services SSM Architecture



For a description of the Web Services Security Service APIs, see "Web Services Interfaces" on page 5-1

Client Trust Model

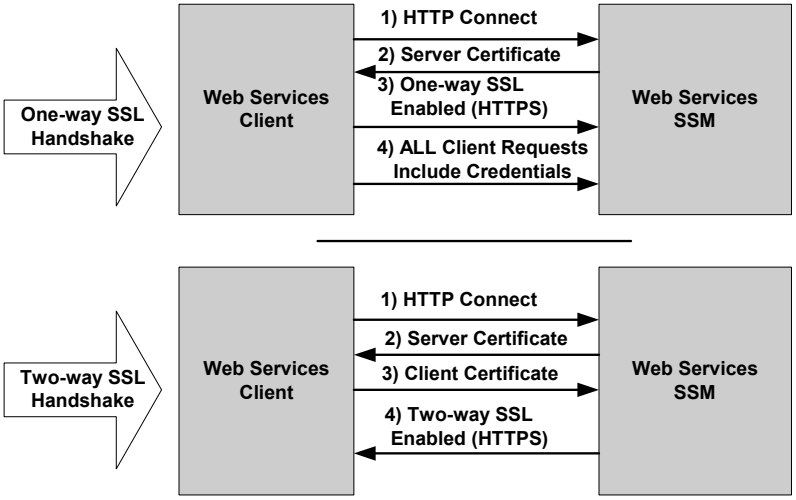
To protect messages in transit between the client and the Web Services SSM, a channel security protocol (SSL 3.0 or TLS 1.0) is used for all communication between the two. The Web Services SSM supports both one-way and two-way SSL (see Figure 2-5). To establish an SSL connection with one-way SSL, only the server is required to present a digital certificate. With two-way SSL authentication, both the client and server must present digital certificates before the SSL connection is enabled between the two. Thus, with two-way SSL, the Web Services SSM not only

authenticates itself to the client to complete the SSL handshake (which is the minimum requirement for certificate authentication), but it also requires authentication from the requesting client. The client first authenticates the server's certificate, and then the server authenticates the client's certificate. The client certificate must be signed by a recognized certificate authority (CA).

As mentioned, Web services clients can communicate with the Web Services SSM over one-way or two-way SSL connections. However, when the client communicates over a one-way SSL, each client request be accompanied by a valid client name/password pair or an ALES cookie. Figure 2-5 shows the Web Services SSM client trust model.

For more information on authentication, see "Authentication Service Interface" on page 5-4.

Figure 2-5 Client Trust Model

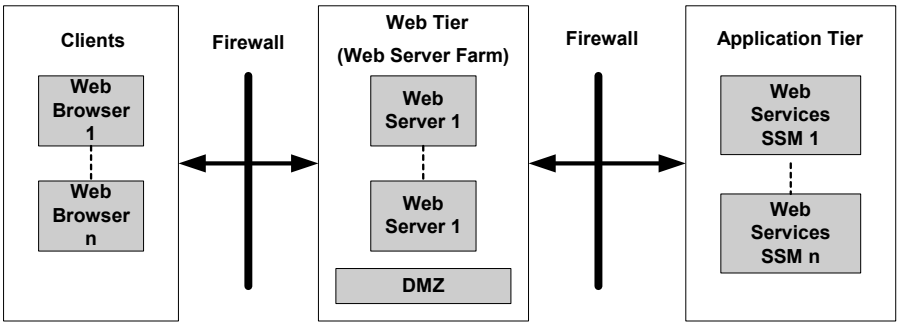


Deployment Model

Figure 2-6 shows how you typically deploy instances of Web Services Security Service Modules (SSM) to protect application resources. The web servers (the web services clients) are located in the web tier, which is in the Demilitarized Zone (DMZ), and are protected from unwanted traffic on the Internet by a firewall. The DMZ is created by using two firewalls. If the web servers and Web Services SSMs are running on different machines, the Web Services SSMs are located in the application tier behind the second firewall for added security. The Web Services SSM

supports Secure Sockets Layer (SSL) communication so all traffic between the web servers and the SSM is encrypted.

Figure 2-6 Typical Web Services Deployment Model

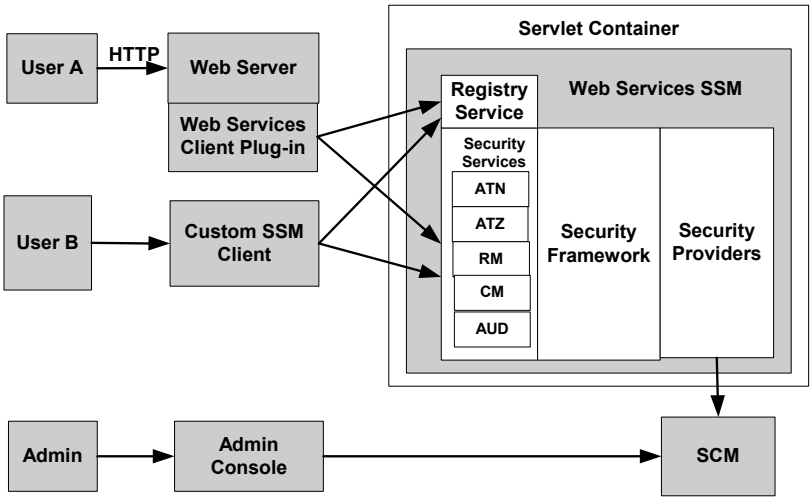


Usage Model

Once the Web Services client has established an SSL connection with the Web Services SSM, it can issue requests on behalf of users for access to the web resources protected by the Web Services SSM (see Figure 2-7).

- User A uses web servers with web services client plug-ins to access the Registry Service and locate specific instances of SSMs, and a specific security service on that SSM instance. The Web servers use that information to make requests for access to application resources on behalf of users. BEA provides two web server products that have built-in web services clients: the IIS Web Server SSM and the Apache Web Server SSM.
- User B uses a custom SSM client, which has a user-written web services client, to access the Registry Service and in turn the specific SSM security service. Because communication between the custom SSM client and the Web Services SSMs is placed on the network, the custom SSM clients can be located any where on the network. Their location is not restricted to the local machine.
- The Admin user uses the administration console to the distribute the initial security configuration and all updates over the network. If the Administration Server is disconnected from the Web Services SSMs, the SSMs continue to function.

Figure 2-7 Usage Model



The interaction between the client and the Web Services SSM is as follows (see Figure 2-7):

1. The client issues a user request that includes the following:
   - A user identity credential (a username/password pair, a signed SAML 1.1 assertion, or ALES cookie)
   - The resource to which the user wants access and the action to be performed
   - The Authentication Service
2. The client queries the local Registry Service to retrieve the fully qualified URL for the endpoint of the authentication service. If the authentication service exists on the local machine, the SSM returns the URL to the client. If it does not exist, the request fails. If the Authentication Service is omitted from the client request, the URL for the default authentication service on the local machine is returned.
3. The web services client calls the Authentication Service, uses the user credentials to authenticate the user. If authentication succeeds, the service returns an identity assertion token that is an internal representation of the user's identity. If the authentication fails, the service returns an AuthenticationFaliure SOAP Fault.
4. The client queries the local Registry Service to retrieve the fully qualified URL for the endpoint of the Authorization Service.

5. The Authorization Service and the associated Role Mapping Service determine whether the user has been granted the privileges required to access to the specified resource and perform the requested action. The service answers the question "Is this user allowed to perform the particular action on specified resource?" To make the access decision, the service compares the roles granted to the user to the policy written to protect the requested resource. If none of the user's roles match the roles allowed to access the requested resource, or if the resource does not have any policy to protect it, access is denied.
6. If access is allowed, the user is granted access to the resource and the request is serviced.

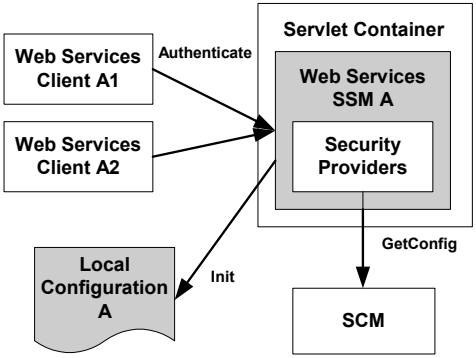
Product Features

The Web Services Security Service Module (SSM) has the following features:

- **Standalone Deployment**

  Each instance of a Web Services SSM is made accessible to clients via a separate SOAP endpoint with a unique URL. Further, each security service is deployed as a separate component inside the hosting process, with each service using disparate configuration entry to identify and initialize itself. Figure 2-8 shows how each Web Services SSM is deployed and initialized in an environment where multiple Web Services SSMs are deployed.

Figure 2-8 Multiple Web Services SSM Deployments



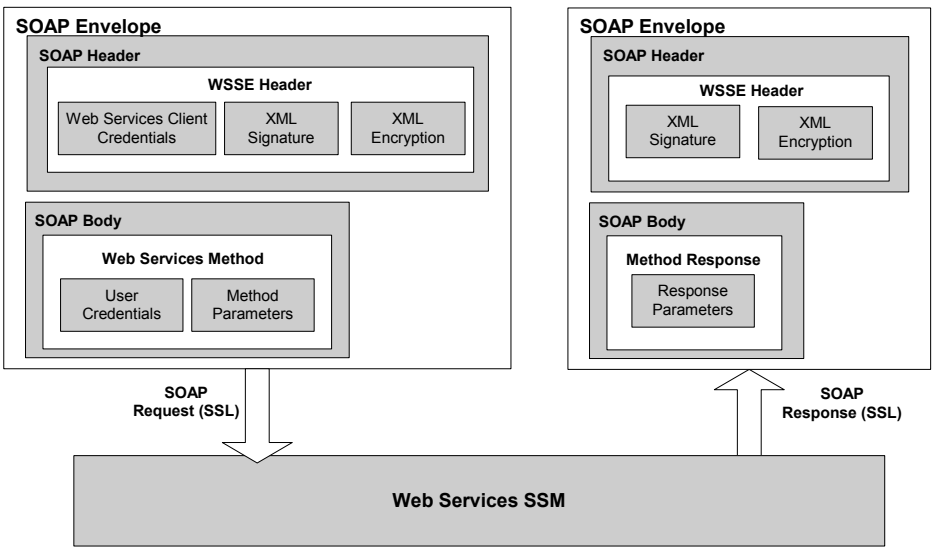
- **Credentials Protection**

  Identity assertions must be signed by a trusted entity. An assertion that is not signed or signed by an unknown authority is rejected and the processing stopped. The digital signature is attached to the identity assertion and covers the entire assertion.

- **XML Structures Follow Industry Standards**

  The XML structures defined by the OASIS WS-Security Standard v1.0 for passing username/password during client authentication with one-way SSL. See Figure 2-9 for the general format of the SOAP messages exchanged between the web services client and the Web Services SSM.

Figure 2-9 SOAP Message Format



- **Secure Web Services security service APIs**

  Access to the Web Services SSM security service APIs is restricted to clients that have been authenticated.

- **Channel and message protection**

  In order to protect messages in transit, the Web Services SSM supports a channel security protocol (SSL 3.0 or TLS 1.0) for all communication that takes place between the SSM and its clients.

- **Support for one-way and two-way SSL client connections**

  The Web Services SSM always authenticates itself to its client using its X.509 site certificate.

A client presents its certificate as part of a two-way SSL handshake with the Web Services SSM servlet container. The client's identity, contained in the SSL certificate, is subsequently used for client authentication.

Note: For two-way SSL authentication, any certificate authority (CA) used to generate the client certificate must be manually added to the Web Services SSM `peer.jks` keystore.

- **Automatic restart of security services**

  The Web Services SSM uses the AquaLogic Enterprise Security process monitoring and management mechanism to automatically restarted a security service if it is found to be unresponsive. Upon restart, the service is initialized using the latest configuration and automatically resumes its normal operation.

- **Event auditing capabilities**

  The Web Services SSM relies on the AquaLogic Enterprise Security auditing capabilities to provide a file-based, audit logging facility with configurable audit log filename. Any service or request-related failures produce an audit trail. Additionally, the following Web Service SSM instance lifecycle events produce audit trails:

  – Initialization complete, ready to serve requests

  – Shutdown initiated

  – Restarted after a process crash

- **Configuration stored locally**

  The Web Services SSM relies only on the local configuration for its operation. The local configuration includes all necessary information to start-up Web Services SSM process, identify its instance, and set up a two-way SSL connection to the local Service Control Manager (SCM) process.

- **WSDL 1.1**

  The Web Services Description Language (WSDL) is an XML-based specification that describes a web service. A WSDL document describes web service operations, input and output parameters, and how a client application connects to the web service.

  For more information, see http://www.w3.org/TR/wsdl.

CHAPTER 3

Optimizing Web Services Performance with Caches

You can improve ALES's authorization performance in Web Services application using one or more caches. The following caches are available:

- authorization caching on the client side
- identity caching on the server side
- authorization caching on the server side

The client-side authorization cache and the server-side identity cache are described in the following sections:

- "Using the Web Services Client Authorization Cache" on page 3-1
- "Using the Web Services SSM Identity Cache" on page 3-7

The server-side authorization cache is described in Authorization Caching in *Integrating ALES with Application Environments*.

Using the Web Services Client Authorization Cache

You can use a client-side Authorization cache to allow an application using the Web Services SSM to take advantage of in-process caching to achieve performance improvements when making authorization calls.

The Web Services Authorization cache has been implemented as an Axis handler. The handler implementation allows you to add and remove the Authorization cache without affecting existing

code. The Authorization cache can be configured through a Java API. If you do not use the configuration API to configure the cache, the default values for the cache will be used.

Installing the Web Services Authorization Cache

The Authorization cache is provided as a separate Java Library (JAR file), `ssmwsClientCache.jar`. This JAR file, as well as the `cacheclientconfig.wsdd` file, must both be available to your Web Services client application in order for the cache to function. The `ssmwsClientCache.jar` is installed by default in:

BEA_HOME`/ales25-ssm/webservice-ssm/lib`

To enable the cache in your Web Services client application:

1. Add `ssmwsClientCache.jar` to the classpath of the Web Services client application.
2. Add the following Java option to the Web Service client application's startup commands:

   `-Daxis.ClientConfigFile=<`*file location for cacheclientconfig.wsdd*`>`

As an alternative, you can enable the cache programmatically by setting the Axis properties for the `cacheclientconfig.wsdd` at the beginning of the Web Services client program before any Axis call. For example:

```
AxisProperties.setProperty("axis.ClientConfigFile",
D:/ssmws/axis/config/cacheclientconfig.wsdd");
```

Client Configuration File

Listing 3-1 gives an example of the `cacheclientconfig.wsdd` file.

Listing 3-1 Cache Client WSDD File

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultClientConfig"
            xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <globalConfiguration>
    <parameter name="disablePrettyXML" value="true"/>
    <requestFlow>
      <handler
           type="java:com.bea.security.ssmws.client.CacheRequestHandler"/>
```

```
    </requestFlow>
<responseFlow>
      <handler
        type="java:com.bea.security.ssmws.client.CacheResponseHandler"/>
    </responseFlow>
  </globalConfiguration>
  <transport name="http"
        pivot="java:org.apache.axis.transport.http.HTTPSender"/>
  <transport name="local"
        pivot="java:org.apache.axis.transport.local.LocalSender"/>
  <transport name="java"
        pivot="java:org.apache.axis.transport.java.JavaSender"/>
</deployment>
```

Third Party Dependencies

The Web Services client Authorization cache requires a third party product, Apache Axis 1.2.1, which is a Java implementation of SOAP. The following elements of Axis are required:

- `axis.jar`
- `axis-ant.jar`
- `commons-discovery-0.2.jar`
- `commons-logging-1.0.4.jar`
- `jaxrpc.jar`
- `saaj.jar`
- `wsdl4j-1.5.1.jar`

For more information about downloading and using Axis, see the Apache Software Foundation's Axis site.

Authorization Cache Operation

The authorization method, `isAccessAllowed()`, has five parameters:

- IdentityAssertion
- Resource
- Action

- Context
- Request Credential Type
- Direction

The first four parameters and Direction act as the keys in the Authorization cache's entries. The keys in the Authorization cache are constructed using the following formula:

> Map (key = `IdentityAssertion` + Resource + Action + Contexts + Direction, value = Result + ResponseAttributes + Roles)

Note that the context object is actually an array of context record elements. Two arrays are considered equal if they contain the same elements in the same order. Consequently, two context objects are considered unequal if the context record elements they consist of were added in a different order.

If the Request Credential Type parameter is set, the return result may have a new `IdentityAssertion`. In this case, the cache will be bypassed, and the cache is updated accordingly if a new `IdentityAssertion` is returned.

The value in the cache holds all the contents of the response, including access decision, response attributes (single values and lists), and roles, if any.

As described in "Config" on page 3-5, the Authorization cache's behavior is configurable, using the `Config` class. The Authorization cache will be updated if one of following criteria is met:

- `SessionExpirationDuration` expires. The expired entry will be removed. The client will call `isAccessAllowed` and establish a new session for this `IdentityAssertion`. SessionExpirationDuration is a system attribute that you can configure using the `Config` class. Set this attribute to a time that is short enough that the policy changes will not affect the correctness of the caching results.
- A new `IdentityAssertion` is returned when calling `isAccessAllowed`. The client will remove all cached evaluation decisions on this `IdentityAssertion`.
- The number of user sessions exceeds the maximum capacity (`Config.sessionEvictionCapacity`). A configurable percentage (`Config.sessionEvictionPercentage`) of least-recently-used sessions will be removed.
- The number of cached decisions exceeds the maximum capacity per user session (`Config.userEvictionCapacity`). A configurable percentage (`Config.userEvictionPercentage`) of least-recently-used cached values will be removed.
- When the API method `CacheManager.flush()` is called, the whole cache will be flushed.

There will be only one Authorization cache in any one Web Services client application. The cache is used only for the authorization call to one Web Services SSM.

Configuring the Client Authorization Cache

In addition to the public API provided by the Web Services SSM (see *Web Services Interface* in *Programming Security for Web Services*), the Web Services Client Authorization Cache provides a configuration API so that you can customize the Authorization cache implementation. The following classes are packaged in `com.bea.security.ssmws.client`:

- Config
- CacheManager

The optimal values to use in configuring the cache depend on your application's particular needs and traffic. The maximum number of entries in the cache is the product of the `SessionEvictionCapacity` and the `UserEvictionCapacity`. Make sure that the maximum size of the cache does not exceed the amount of memory that you can allocate for the cache. In production, monitoring the CacheManager's `HitRatio` will indicate how effectively the cache is being used.

Please note that configuration parameters of the server-side authorization cache (described in Authorization Caching in *Integrating ALES with Application Environments*) have no effect on the client-side authorization cache. These caches are completely independent and do not share any configuration.

Config

The `com.bea.security.ssmws.client.Config` class provides these methods for configuring the Authorization cache:

setSessionExpirationDuration

```
public void setSessionExpirationDuration(int
sessionExpirationDuration)
```

Set the duration for which to cache session data, in seconds. Default value is 60.

setSessionEvictionCapacity

```
public void setSessionEvictionCapacity(int sessionEvictionCapacity)
```

Set the number of authorization sessions to maintain in the cache. Once the limit is reached, old sessions are dropped and automatically re-established when needed. Default value is 1000.

setSessionEvictionPercentage

```
public void setSessionEvictionPercentage(float
sessionEvictionPercentage)
```

Set the percentage of authorization sessions to drop when the eviction capacity is reached. Default value is 10.

setUserEvictionCapacity

```
public void setUserEvictionCapacity(int UserEvictionCapacity)
```

Set the maximum number of users who can have sessions in cache simultaneously. Once exceeded, old cached values are overwritten. Default value is 500.

setUserEvictionPercentage

```
public void setUserEvictionPercentage(float UserEvictionPercentage)
```

Set the percentage of users whose cached sessions will be dropped when the eviction capacity is reached. Default value is 10.

CacheManager

The `com.bea.security.ssmws.client.CacheManager` class provides these methods for configuring and operating the Authorization cache:

initialize

```
public static void initialize(Config config)
```

Initializes the configuration for the Authorization cache with the given `Config` object. This method can be called one time. If any other call occur before this, an implicit initialization is done. Once initialized, subsequent calls to initialize are ignored.

instance

```
public CacheManager instance ()
```

Returns the one global instance of `CacheManager`.

flush

```
public void flush()
```

Flushes all the entries in the cache.

getVersion

```
public String getVersion ()
```

Return a version string. Currently, “1.0” will be returned.

getSize

```
public void getSize()
```

Return the total number of cache entries at the time of this call.

getHitRatio

```
public void getHitRatio()
```

Return the hit ratio at the time of this call.

Using the Web Services SSM Identity Cache

You can improve the performance of Web Services applilcations by enabling and configuring the Web Services Security Service Module's server-side identity cache.

Every `isAccessAllowed` operation called by the client requires a valid identity (subject, roles, custom profile data, etc). However, the client stores only a token that can be used to create a valid identity on the server side. Identity creation is a relatively expensive operation that can consume substantial amount of resources and time. To avoid this overhead, you can configure the Web Services SSM to cache the identities; thus, when the server gets a request, it will try to find the identity object in the cache instead of creating a new one.

You configure the Web Services SSM identity cache in the `<cache>` element of `<`*`INSTANCE_HOME`*`>/apps/ssmws-asi/SAR-INF/config.xml`, where `<`*`INSTANCE_HOME`*`>` is the installation directory of the Web Services SSM instance. Listing 3-2 shows a fragment from this configuration file.

Listing 3-2 Configuration for Server-Side Identity Cache

```
<cache>
   <!-- This setting enables/disables server-side identity caching. Values:
true/false. Optional -->
      <enabled>true</enabled>
<!-- TTLs for server-side caching of user identities, seconds. Names: any
acceptable credential type. Optional -->
   <tokenTTLs>
      <SAML.Assertion>300</SAML.Assertion>
      <ALESIdentityAssertion>300</ALESIdentityAssertion>
   </tokenTTLs>
</cache>
```

The `<cache>` element contains two parameters that configure the identity cache:

- enabled - specifies whether the identity cache is enabled (default value is true).
- tokenTTLs - Time to live in the cache for different credential types. Every nested element specifies the time to live in seconds for entries in the corresponding identity cache. The default time to live is 300 seconds. Once the configured time to live has expired, the the identity object is removed from the cache and a subsequent call will require re-authentication of the user.

You should configure the identity cache based on your requirements, available hardware, application usage patterns, etc. However, if the Web Services SSM is used to secure an application that has a concept of session timeout itself, for example, a web application or a portal application, then you should set the timeout value of the identity cache to a value similar to the one that is used in the application. In that case, the identity object will be removed as soon as the application session has been expired, and, on the other hand, no additional re-authentication will be required while the session is active.

CHAPTER 4

Using Failover with a Web Services Client

AquaLogic Enterprise Security supports the ability of Axis-based Web Services clients to fail over to another Web Services SSM when the client encounters a connection problem. The following sections describe this failover functionality:

Note: The Web Services client failover feature is supported for clients developed with Axis 1.2 or later. For more information, see the Apache Axis site: http://ws.apache.org/axis/ .

Configuring a Web Services Client for Failover

To configure a Web Services client for failover:

1. In the client's classpath, include `/webservice-ssm/lib/ssmwsClientFailover.jar` and `Log4j.jar`.
2. In the client's JAVA_OPTIONS, include:

   ```
   -Daxis.ClientConfigFile=<SSMWS_INSTANCE_HOME>/config/failover-client-config.wsdd
   ```

3. In the client's code, include failover initialization code similar to the code in Listing 4-1:

Listing 4-1 Sample Failover Initialization

```
// Populate the SSMInstanceList with the list of WS SSM instances. For
example:

ArrayList SSMInstanceList = new ArrayList();
SSMInstanceList.add("http://12.10.1.4:9090");
SSMInstanceList.add("http://12.10.1.2:9090");

// Set the Http Connection Time Out and number of retry attempts
// that will be made on the SSM instance

int noOfRetry = 3;
int httpConTimeOut = 60000; // The time is in milliseconds.
FailOverManager.getInstance(SSMInstanceList, noOfRetry,httpConTimeOut);

//     <<<<<<< Original Client Code >>>>>>
ServiceRegistryBindingStub serviceRegistry =
new ServiceRegistryBindingStub(new java.net.URL(serviceRegistryURL),null);

SsmIdType ssmIdType =
new SsmIdType(ServiceTypeEnum.ALES_AUTHENTICATION, ssmId);
ComplexAnyURI authURL;

authURL = serviceRegistry.locateService(ssmIdType);

System.out.println("Authentication URL: " +
authURL.getLocateServiceResponse().toString());
/**
* Authentication Code
*/
// get the URL for the Authentication Service should be
// http://<hostname>:<webservice-ssm-port>/Authentication
authenticationService = new  AuthenticationBindingStub(
new java.net.URL(authURL.getLocateServiceResponse().toString()),null);
```

Client Configuration File

Listing 4-2 gives an example of the `failover-client-config.wsdd` file.

Listing 4-2 Cache Client WSDD File

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultClientConfig"
            xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <globalConfiguration>
    <parameter name="disablePrettyXML" value="true"/>
    <parameter name="enableNamespacePrefixOptimization" value="false"/>
  </globalConfiguration>
  <transport name="http"
  pivot="java:com.bea.security.ssmws.client.failover.FailOverHTTPSender"/>
  <transport name="local"
  pivot="java:org.apache.axis.transport.local.LocalSender"/>
  <transport name="java"
pivot="java:org.apache.axis.transport.java.JavaSender"/>
</deployment>
```

FailOverManager API

The `FailOverManager` class stores and manages the failover endpoints for Web Services clients. This class is a singleton; it stores a list of endpoints and the values for the number of retries on connection failure and HTTP connection timeout. It includes the following public methods:

getInstance()

Returns an instance of the `FailOverManager`. This method makes sure that there is only one instance of this class in the JVM. This method is invoked to get an instance of the `FailOverManager` configured with its default properties. The `FailOverManager` has the following configuration properties:

failOverEndPoints
: A list of endpoints; on failover, the Web Services client attempts to connect to the listed endpoints in the order they are provided. If no endpoinnts are specified, the only connection URL that will be tried is the first one that is used when setting up a Web Service call via the `BindingStub`. Default is empty.

noOfRetries
: The number of times each endpoint is tried before failing over to the next endpoint. Default is 3.

httpConTimeOut
: Time, in milliseconds, to wait before trying again to connect to the current endpoint. Default is 0, indicating immediate retry.

If you need to change any of these values, then use the `getInstance(ArrayList failOverEndPoints,int noOfRetries, int httpConTimeOut)` method, which takes these as arguments.

getInstance(ArrayList failOverEndPoints,int noOfRetries, int httpConTimeOut)

Returns an instance of the `FailOverManager`. You can supply values for the failover endpoint list, number of retries, and HTTP connection timeout as arguments.

setEndPointList (ArrayList endpoints)

Replaces the existing list of failover endpoints with the provided list of endpoints. The `currentEndPoint` variable of the `FailOverManager` points to the first endpoint in the list. On failover, the Web Services client attempts to connect to the listed endpoints in the order they are provided. Each endpoint is tried the number of times specified by the `noOfRetries` property of the `FailOverManager` before failing over to the next endpoint. If all the endpoints fail, AXIS will throw the regular connection exception to the client.

This method is thread-safe; you can call this method during runtime to refresh the list of endpoints.

getEndPointList()

Returns the list of end points.

CHAPTER 5

Web Services Interfaces

To develop Web Services clients, you use the Web Services Security Service Module (SSM) application programming interfaces (APIs) developed by BEA Systems.

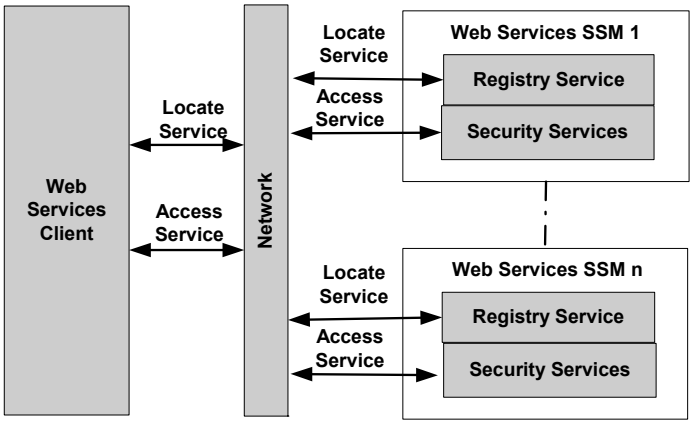
These interfaces are described in the following sections:

Registry Service Interface

To map security service type and SSM configuration and SOAP endpoints, each Web Services SSM process has its own separate Registry Service. The address of the Registry Service should be known to the web services client from its configuration. This service maps the SSM Configuration ID and the service type to the Web Services SSM and SOAP endpoints for the security services. The Configuration ID is defined in the `default.properties` file for the SSM

instance. The Registry Service makes it possible to distinguish between the supported service types (auditing, authentication, authorization, credential mapping, and role mapping). Registry Service operations on a particular machine are limited to the local machine (see Figure 5-1). The Web Services client learns about the Registry Service from its configuration file.

Figure 5-1 Registry Service Function



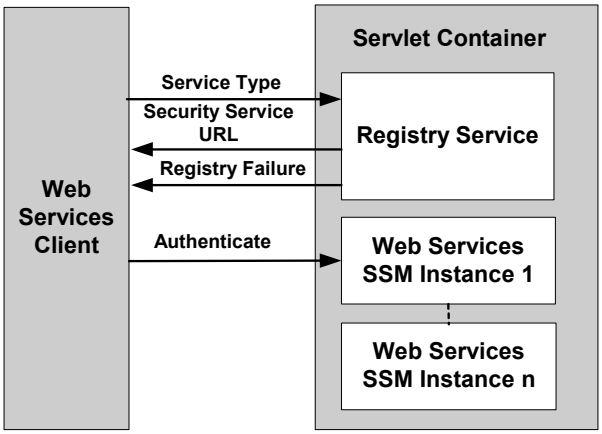
The following topics provide or more information on the Registry Service.

- "How the Registry Service Works" on page 5-2
- "Registry Service Methods" on page 5-3

How the Registry Service Works

The Registry Service works as follows (Figure 5-2):

Figure 5-2 How the Registry Service Works



1. If a web services client asks the Registry Service about the existence of a particular security service type, the Registry Service responds with a true or false answer.
2. If a web services client asks for the URL of a valid security service type for the local host, the Registry Service returns the fully qualified URL of the endpoint for the requested service type that is provided by the SSM configuration. The Configuration ID is defined in the `default.properties` file.
3. If a web services client asks the registry for a URL for a security service type that has an invalid security service type for Web Services SSM identified by the Configuration ID, the Registry Service returns a RegistryFailure SOAP Fault.

Note: The Registry Service does not provide lifecycle management functions for the Web Services.

For more information on the Registry Service interface and the methods it supports, see *WSDLdocs for Web Services Interfaces*.

Registry Service Methods

The Registry Service supports two methods: `locateService()` and `doesServiceExist()`. Both methods accept the requested service type of the Web Server SSM or Web Services SSM that provides the service. The supported service types are: `ALES_AUDIT`, `ALES_AUTHENTICATION`, `ALES_AUTHORIZATION`, `ALES_CREDENTIAL`, and `ALES_ROLE`. The type for all service types is `string`.

For the `locateService()` method, the Registry Service returns the fully qualified URL for the endpoint of the requested service. For the `doesServiceExist()` method, the service returns a Boolean value (`true` or `false`) that indicates whether the service exists and can be requested.

Methods Common to All Web Services Interfaces

The following methods are supported by all Web Services interfaces, except for the Registry Service interface.

- `getServiceType()`—This method takes an empty request and returns a structure that contains the service. The Web Services SSM supports these security service types: `ALES_AUDIT`, `ALES_AUTHENTICATION`, `ALES_AUTHORIZATION`, `ALES_CREDENTIAL`, and `ALES_ROLE`. The type for all service types is `string`.
- `getServiceVersion()`—This method takes an empty request and returns a structure that contains the version of the service. The Web Service SSM supports these version types: `MajorVersion` (type `int`), `MinorVersion` (type `int`), `PatchLevel` (type `string`), and `Version` (type `long`).
- `isCompatible()`—This method accepts service version information and returns compatibility information. The method accept these values: `MajorVersion` (type `int`), `MinorVersion` (type `int`), `PatchLevel` (type `string`), and `Version` (type `long`). It returns these compatibility responses: `ALES_NOT_COMPATIBLE`, `ALES_COMPATIBLE`, `ALES_COMPATIBLE_DEPRECATED`, `ALES_COMPATIBLE_UNKNOWN`. All compatibility responses are returned as strings. You use this method to determine whether the version of the service interface specified in the web services client is compatible with the current version of the service interface in the instance of the Web Services SSM.

Authentication Service Interface

The Authentication Service provides security functions to an application so as to establish, verify, and transfer a person or process identity. Thus, the Authentication Service provides two main security functions: authentication and identity assertion.

The Authentication Service accepts user credentials and/or identity assertion tokens and verifies that they match the user identity stored in the existing user profile. The following types of identity assertion tokens are supported:

- Username/password
- Signed Security Assertion Markup Language (SAML) 1.1 assertions

- ALES cookie

The Extensible Markup Language (XML) structures used by the Authentication Service for transmitting identity assertion tokens and other credential types are defined the WSDL interface.

To ensure secure handling of credentials, all credentials presented to the Web Services Security Service Module must satisfy the following requirements:

- All SAML 1.1 identity assertions must be signed by a trusted entity, as determined by the SAML IdentityAsserter. The signature must be attached to the identity assertion and cover the entire assertion. An identity assertion that is not signed or signed by an unknown authority is rejected and the processing is stopped.
- The caller should verify the validity of server credentials prior to invoking the Web Service. In particular, the caller should verify its validity by examining its expiration date, certification path, and revocation status.

The Web Services SSM only accepts clear-text passwords. However, the credentials presented are always be protected by SSL, either one-way or two-way. The Web Services SSM returns credentials to clients over SSL as well.

The SOAP authentication interface enables the Web Services SSM to return challenges to clients if they fail to provides the information necessary to complete the authentication process. In such cases, the client can respond with the requested information. In order to avoid expensive round trips, however, the web services client should pass in all available credentials information with the initial SOAP request.

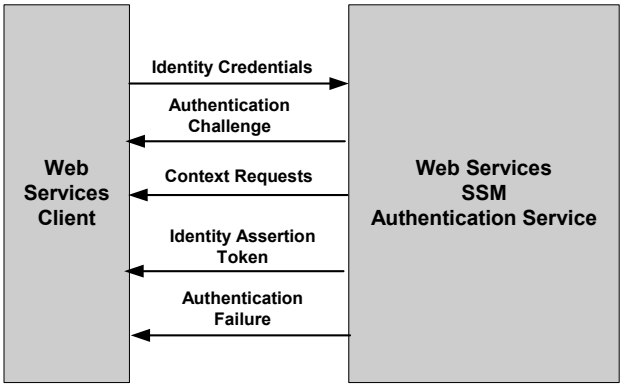
The following topics provide more information on the Authentication Service interface:

- “Authentication Process” on page 5-5
- “Authentication Service Methods” on page 5-7

Authentication Process

The authentication process is as follows (see Figure 5-3):

Figure 5-3 Authentication Service Process



1. The web services client connects to the Web Services SSM over HTTP and the Web Services SSM responds by presenting a digital certificate to prove its identity to the client. The Web Services SSM authenticates itself to the web services client using its server X.509 certificate. If the Web Services SSM presents a certificate that is not valid (for instance, the certificate has expired or has a subject name mismatch), the client should break the connection.
2. The web services client verifies the SSMs digital certificate and submits a certificate signed by a recognized certificate authority (CA) to the Web Services SSM.
3. The Web Services SSM verifies the clients certificate and establishes an SSL connection.
4. The web services client submits credentials to Web Services SSM to login. Web services clients present user credentials with each login request. Use credential can be username/password, a SAML assertion, or an ALES cookie.
5. The Authentication Service verifies the client credentials.
6. If the credentials authenticate, the Authentication Service returns an identity assertion token to the web services client.
7. If the credentials do not authenticate, the Authentication Service submits the credentials for checking. The Authentication Service does one of two things:
   - Validates the user credentials, grants the login, and returns a success message to the Web Services SSM.
   - Invalidates the user credentials, blocks the login, and returns a failure message to the Web Services SSM.

8. If an authentication decision cannot be made with the parameters included in the initial client request, the request fails.
9. If authentication is successful, the Web Services SSM returns the default identity assertion token representing the user. The type of the default identify assertion token is configurable. If the type is not specified, SAML assertions are used.
10. If authentication fails, the Web Services SSM returns an `AuthenticationFailure` SOAP Fault to the caller.

For more information on the Authentication Service Interface and the methods it supports, see *WSDLdocs for Web Services Interfaces*.

Authentication Service Methods

To support authentication functions the authentication provides the following methods:

- "authenticate() Method" on page 5-7
- "assertIdentity() Method" on page 5-8
- "isAssertionTokenSupported() Method" on page 5-9
- "validateIdentity() Method" on page 5-10

authenticate() Method

This method accepts any credential type supported by the authentication provider or a response to an earlier authentication challenge, and, optionally, the type of requested identity assertion that represents the identity and application context of the authenticated user. In response, it returns either the requested identity assertion token and status information or an authentication challenge.

Note: In addition to the identity assertion types, the Web Services SSM supports user credentials in form of usernames with passwords.

Table 5-1 describes the `authenticate()` method parameters.

Table 5-1 authenticate() Method Parameters

Parameter	Description	Supported Values/Types
Input Parameters		
`Identity Credential`	Specifies the identity credential that is to be used to authenticate the caller.	`ssm:IdentityCredential Type`

Table 5-1 authenticate() Method Parameters (Continued)

Parameter	Description	Supported Values/Types
`RquestCredential Type`	Specifies the identity credential type.	`ssm:CredentialTypeType`
`AppContext`	Specifies the application context in which authentication is being requested.	`ssm:"ContextType"` Where `ssm:ContextType` can be one of the following: `StringValue/string`, `BoolValue/boolean`, `DateTimeValue/dateTime`, `TimeValue/time`, `IntValue/integer` `IpValue/ssm:IpType`
`Return Parameters`		
IdentityAssertion	Returns the identity assertion token.	`type="ssm:Identity AssertionType"`
Challenge	Returns an authentication challenge to the caller.	`type="ssm:ChallengeType"`
StatusInfo	Returns status information.	type="xsd:string"

assertIdentity() Method

This method accepts any supported identity assertion type or a response to an earlier authentication challenge, and, optionally, the type of requested identity assertion that represents the identity and application context of the authenticated user. In response, it returns either the requested identity assertion token and status information or an authentication challenge.

Table 5-2 describes the `assertIdentity()` method parameters.

Table 5-2 assertIdentity() Method Parameters

Parameter	Description	Supported Values/Types
Input Parameters		
`IdentityAssertion`	An identity assertion token that represents the authenticated identity of the user.	`type="ssm:IdentityAssertion Type"`
`Requested CredentialType`	Specifies the type of credential being submitted for authentication.	`type="ssm:CredentialTypeType"`

`AppContext`	Specifies the application context in which authentication is being requested.	`ssm:ContextType`
		Where `ssm:ContextType` can be one of the following:
		`StringValue/string`,
		`BoolValue/boolean`,
		`DateTimeValue/dateTime`,
		`TimeValue/time`,
		`IntValue/integer`
		`IpValue/ssm:IpType`

`Return Parameters`		
IdentityAssertion	Returns the identity assertion token.	An acceptable user's identity assertion token. `type="ssm:Identity AssertionType"`
Challenge	Returns an authentication challenge to the caller.	`type="ssm:ChallengeType"`
StatusInfo	Returns status information.	type="xsd:string"

isAssertionTokenSupported() Method

This method accepts an identity assertion credential type that represents the authenticated user's identity. It returns a Boolean value (`true` or `false`) to indicate whether this token type is supported by this instance of the Web Services SSM.

Table 5-3 describes the `isAssertionTokenSupported()` method parameters.

Table 5-3 isAssertionTokenSupported() Method Parameters

Parameter	Description	Supported Values/Types
Input Parameter		
`Assertion CredentialType`	Specifies the identity assertion credential type. The type specified can be any type supported by the ALES Credential Mapping provider. The Authentication Service may also return an authentication challenge to the caller requesting other assertion token types. If the caller fails to specify an assertion token type supported by the Authentication Service even after challenges, the Authentication Service returns `CredentialMappingFailure` to the caller.	`AssertionCredentialType type="ssm:CredentialTypeType"`
Return Parameter		
isAssertionToken SupportedResponse	Specifies whether the assertion token is supported.	`true/false type="xsd:boolean"`

validateIdentity() Method

This method accepts any supported identity assertion type that represents the identity of the authenticated user. It returns a structure with a Boolean value (`true` or `false`) that indicates whether the token is valid.

Table 5-4 describes the `validateIdentity()` method parameter.

Table 5-4 validateIdentity() Method Parameters

Parameter	Description	Supported Values/Types
Input Parameter		
`IdentityAssertion`	An identity assertion token that represents the authenticated identity of the user.	`IdentityAssertion type="ssm:IdentityAssertionType"`
Return Parameter		
`validateIdentityResponse`	Specifies whether the identity is valid.	`true/false type="xsd:boolean"`

Authorization Service Interface

The Authorization Service is a service that allows an application to determine if a specific identity is permitted to access a specific resource. This decision may then be enforced in the application directly at the policy enforcement point.

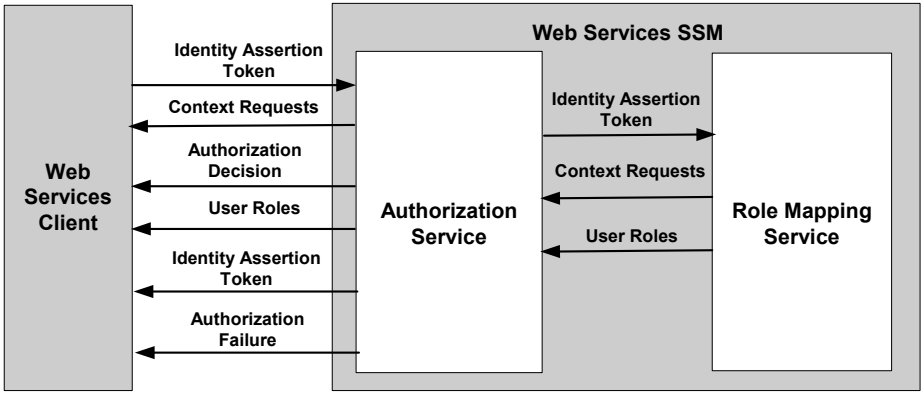
The Authorization Service is primarily based on a single method: `isAccessAllowed()`. This method accepts a supported type of user credential or an identity token, a runtime resource, and a runtime action. Optionally, this method can accept the type of the requested identity assertion token that represents the identity of the authenticated user, the application context, and direction parameters. The `isAccessAllowed()` method requires that a valid, authenticated identity or a null identity token (representing an anonymous identity) be present when requesting an access decision. The following topics provide more information on the Authorization Service.

- "Authorization Process" on page 5-11
- "Authorization Service Methods" on page 5-13

Authorization Process

The authorization process is as follows (see Figure 5-4):

Figure 5-4 Authorization and Role Mapping Process



1. The authorization process begins when the web services client calls the Web Services SSM to answer the question "Is this user allowed to perform this particular action on the specified resource?" Identity assertion token types supported include ALES cookies and signed SAML 1.1 assertions.
2. If the token matches a credential in the cache, the SSM sets the user identity to the internal identity representation, which includes the user identity and the list of roles the user has been granted.
3. If the token does not match any of the credentials in the cache, the SSM calls the Authorization Service and the Role Mapping Service to determine whether the user is authorized to access resources and to get the list of roles the user has been granted. If the user is authorized, the user identity is set to the internal identity representation, which includes the user identity and the list of roles the user has been granted.
4. The Authorization Service then compares the user identity to the resource security policy to determine whether the user is in a role that has been granted the access privilege that is necessary to perform the requested action on the particular resource.
5. If the authorization process fails due to parameter-related problems, an `AuthorizationFailure` SOAP Fault is returned to the caller.
6. In the absence of an error, the authorization decision that is returned to the web services client contains a clear and unambiguous true or false statement that allows or disallows access to the resource in question. A negative authorization decision is a valid result and does not result in a SOAP Fault.

For more information about the Authorization Service interface, see the *WSDLdocs for Web Services Interfaces*.

Authorization Service Methods

The Authorization Service supports the following methods:

- "isAccessAllowed()" on page 5-13
- "isAuthenticationRequired() Method" on page 5-15

isAccessAllowed()

This method accepts a supported type of an identity assertion token and runtime resource and action structures. Optionally, it can accept the requested identity assertion credential type, application context, and authorization direction parameters. In response, this method returns the authorization decision and authorization data, or, if required by the authorization provider, additional context requests.

Table 5-5 describes the `isAccessAllowed()` method parameters.

Table 5-5 isAccessAllowed() Method Parameters

Parameter	Description	Supported Values/Types
Input Parameters		
`IdentityAssertion`	An identity assertion token that represents the authenticated identity of the user. The token type must be supported by the ALES Credential Mapping provider; otherwise, the service returns `CredentialMappingFailure` to the caller.	`IdentityAssertion/ ssm:IdentityAssertionType`
`RuntimeResource`	Specifies the runtime resource that the caller is requesting.	`ResourceString, AuthorityName` `type="ssm:Runtime ResourceType"`

Table 5-5 isAccessAllowed() Method Parameters (Continued)

Parameter	Description	Supported Values/Types
`RuntimeAction`	Specifies the runtime action that the caller is requesting.	`ActionString, AuthorityName` `type="ssm:Runtime ActionType"`
`Request CredentialType`	Specifies the identity assertion credential type. The type specified can be any type supported by the ALES Credential Mapping provider. The Authentication Service may also return an authentication challenge to the caller requesting other assertion token types. If the caller fails to specify an assertion token type supported by the Authentication Service even after challenges, the Authentication Service returns `CredentialMappingFailure` to the caller.	`AssertionCredentialType/ ssm:CredentialTypeType`
`AppContext`	Specifies the application context in which access to the resource is being requested.	`ssm:"ContextRecordType"` Where `ssm:ContextRecordType` is one of the following: `StringValue/string, BoolValue/boolean, DateTimeValue/dateTime, TimeValue/time, IntValue/integer IpValue/ssm:IpType`
`AtzDirection`	Specifies authorization direction parameters.	`ALES_ONCE, ALES_POST, ALES_PRIOR` type="ssm:AtzDirectionEnum"
Return Parameters		
`isAccessAllowed Response`	Indicates whether access is allowed.	`true/false` `type="xsd:boolean"`

Table 5-5 isAccessAllowed() Method Parameters (Continued)

Parameter	Description	Supported Values/Types
`AtzDecisionData`	Information about the access decision.	`type="ssm:AtzDecisionDataType"`
`"ContextRequests"`	Requests additional context if required by the ASI Authorization provider.	type="ssm:ContextRequestsType"

isAuthenticationRequired() Method

The Authorization Service interface also supports the `isAuthenticationRequired()` method. This method accepts a runtime resource and a runtime action. It returns a Boolean value (`true` or `false`) that indicates whether authentication is require to access this resource. The web services client uses this method to test whether privileges are required to access a particular resource.

Table 5-6 describes the `isAuthenticationRequired()` method parameters.

Table 5-6 isAuthenticationRequired() Method Parameters

Parameter	Description	Parameter Values/Types
Input Parameters		
`RuntimeResource`	Specifies the runtime resource that the caller is requesting.	`ResourceString, AuthorityName` `type="ssm:RuntimeResourceType"/`
`RuntimeAction`	Specifies the runtime action that the caller is requesting.	`ActionString, AuthorityName` `type="ssm:RuntimeActionType"`
Return Parameter		
isAuthentication RequiredResponse	Indicates whether authentication is required.	`true/false` `type="xsd:boolean"`

Authorization Via XACML

This version of AquaLogic Enterprise Security allows external applications to ask authorization questions using the XACML protocol. This capability is supported only in the Web Services SSM.

The XACML service is implemented as an extension to the existing Authorization Service in the Web Service SSM, and uses the same configuration and administration scripts of the Web Service SSM. The XACML service is silently installed together with the Web Service SSM. The XACML service is not viewable or configurable through the Administration Console.

Note: The XACML service supports XACML 2.0 only. It does not support a XACML 1.0 context. In addition, the SAML 2.0 profile for XACML 2.0 is not supported in this release.

Two-Way SSL Recommended

As described in "Client Trust Model" on page 2-7, the Web Services SSM supports both one-way and two-way SSL (see Figure 2-5). However, the connection between the PEP and the XACML service should be over two-way SSL.

XACML Service Use Case

This section describes a typical use case for the XACML service.

The following new terms are used in this section: **PEP** and **PDP**. The policy enforcement point (PEP) is the system entity that performs access control by making decision requests and enforcing authorization decisions. The policy decision point (PDP) is the system entity that evaluates applicable policy and renders an authorization decision. The XACML service implements a PDP: it provides a XACML-based Authorization service (inside the Web Service SSM) to your client PEP.

The basic use case for the XACML service is similar to the following:

1. The PEP receives an access request from some user (subject).
2. The PEP makes sure that the user is authenticated, either by authentication or some form of identity of assertion.
3. The PEP constructs the XACML context request using the user's identity assertion, the action to be performed, the resource the user is requesting, and the environmental values if applicable.

4. The PEP sends the request via SOAP to the XACML service (PDP).
5. The XACML service converts the request to a format understood by the AquaLogic Enterprise Server and calls the Authorization Service's isAccessAllowed method to authorize the user.
6. Depending on the authorization result, the XACML service constructs a XACML context response and returns it to the PEP with status, and obligations if any.
7. The PEP takes actions according to the response.

Sample XACML Client Application is Provided

This version of AquaLogic Enterprise Security includes a simple XACML client (PEP) application that does authorization to the Web Service SSM using the XACML protocol. The example demonstrates the following concepts:

- Authorization with ALESIdentityAssertion as the subject.
- Authorization with extra attributes in the subject and/or environment in XACML context.
- Get authorization result, response attributes and roles from the XACML response.

The example is located in
`BEA_HOME\ales25-ssm\webservice-ssm\examples\XACMLClient`.

Overview of XACML Context

XACML is an OASIS standard language that specifies schemas for authorization policies and for authorization decision requests and responses. It also specifies how to evaluate policies against requests to compute a response. In this release, AquaLogic Enterprise Server implements the authorization process using XACML context. The XACML context is used to convey an authorization decision request and response.

The XACML standard (http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf) specifies how the XACML context should be used for authorization, and the behaviors of PEPs and PDPs.

The XACML context is the protocol used by the PEP to ask the PDP for an authorization decision. It defines a `<xacml-context:Request>` element to convey the request and a `<xacml-context:Response>` for the response. The request contains four elements for authorization, including:

- Subject

- Action
- Resource
- Environment

The PEP sends the authorization request to the PDP using a XACML context, which contains the subject, action, resource, and environment. The PEP is asking the PDP the following question: "Given such a subject, is it allowed to perform the specified action on the specified resource in the specified environment?".

The response contains:

- Decision
- Status
- Obligations

The XACML service takes the XACML request, converts the request to a format understood by the AquaLogic Enterprise Server, and calls the Java API (specifically, the Authorization Service's isAccessAllowed method) to authorize the user.

When a decision has been made, the XACML service wraps the AquaLogic Enterprise Server returned values into the XACML response. The XACML service returns a SOAP fault if the PEP sends a malformed request that does not follow the XACML context schema. Otherwise, the XACML service will always return a decision of "Permit", "Deny", "Indeterminate" or "NotApplicable".

The PEP then interprets the result and enforces the decision. For the "Permit" decision, the PEP allows the request, and for the "Deny" decision, the PEP denies the request. For results of "Indeterminate" or "NotApplicable," it is up to the PEP to interpret them. For "Permit-biased" PEP, the access will be allowed; for "Deny-biased" PEP, access will be denied.

Authentication and Valid Token Types

The policy enforcement point (PEP) that sends the authorization request to the XACML service is responsible for authenticating the user and supplying a valid AquaLogic Enterprise Server token as the subject.

The following types of identity assertion tokens are supported:

- Username/password
- Signed Security Assertion Markup Language (SAML) 1.1 assertions

- ALES cookie

The Extensible Markup Language (XML) structures used by the Authentication Service for transmitting identity assertion tokens and other credential types are defined the WSDL interface.

The XACML service accepts the token supplied by the PEP and tries to assert that token (assertion) in AquaLogic Enterprise Server.

SOAP Binding of XACML Context

As described in "Overview of XACML Context" on page 5-17, the request contains an `<xacml-context:Request>` and the response contains an `<xacml-context:Response>` element. These two elements are directly enclosed in the SOAP body.

One, and only one, request or response should be in the SOAP body.

There are currently two defined XACML namespaces. Policies are defined using the identifier `urn:oasis:names:tc:xacml:2.0:policy:schema:os`, and request and response contexts are defined using the identifier `urn:oasis:names:tc:xacml:2.0:context:schema:os`.

This implementation uses the following identifier to represent the XACML service:`http://security.bea.com/ssmws/ssm-ws-1.0.wsdl`.

Listing 5-1 shows a sample SOAP request.

Listing 5-1 Sample Soap Request

```
POST /XACMLAuthorization HTTP/1.0
Host: pdp-01
Connection: Keep-Alive
Content-Type: application/soap+xml; charset=utf-8
SOAPAction: ssmws:xacml:authorization
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <xacml-context:Request
xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os">
      <xacml-context:Subject>…</xacml-context:Subject>
```

```
      <xacml-context:Resource>…</xacml-context:Resource>
      <xacml-context:Action>…</xacml-context:Action>
      <xacml-context:Environment>…</xacml-context:Environment>
    </xacml-context:Request>
  </soap:Body>
</soap:Envelope>
```

If a malformed SOAP request is sent to the server (for example, a non-XML request or an XML request that does not follow the schema definition), a SOAP fault is returned to the client (PEP).

How the XACML Request Element is Interpreted in AquaLogic Enterprise Security

The request element defined by the XACML standard (http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf) is as follows:

```
<xs:element name="Request" type="xacml-context:RequestType"/>
<xs:complexType name="RequestType">
    <xs:sequence>
        <xs:element ref="xacml-context:Subject" maxOccurs="unbounded"/>
        <xs:element ref="xacml-context:Resource" maxOccurs="unbounded"/>
        <xs:element ref="xacml-context:Action"/>
        <xs:element ref="xacml-context:Environment"/>
    </xs:sequence>
</xs:complexType>
```

As detailed in the standard, the `<Request>` element contains `<Subject>`, `<Resource>`, `<Action>` and `<Environment>` elements. There may be multiple `<Subject>` elements and, under some conditions, multiple `<Resource>` elements. Each child element contains a sequence of `<xacml-context:Attribute>` elements associated with the subject, resource, action, and environment, respectively.

The sections that follow describe how the <Subject>, <Resource>, <Action> and <Environment> elements are interpreted.

Attribute

XACML subject, resource, action, and environment elements usually contain XACML context attributes, <xacml-context:Attribute>.

The context attributes contain metadata (such as AttributeId, DataType), and one or more <AttributeValue> element. As described in the XACML standard (http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf), the schema of Attribute is as follows:

```
<xs:element name="Attribute" type="xacml-context:AttributeType"/>

<xs:complexType name="AttributeType">

    <xs:sequence>

        <xs:element ref="xacml-context:AttributeValue"
maxOccurs="unbounded"/>

    </xs:sequence>

    <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>

    <xs:attribute name="DataType" type="xs:anyURI" use="required"/>

    <xs:attribute name="Issuer" type="xs:string" use="optional"/>

</xs:complexType>
```

Different attributes in different elements will have different AttributeId and DataType.

The <AttributeValue>, by definition, can carry anything. It contains any element under any namespace, as well as any attributes. Any valid XML data can be put in. However, it must be meaningful for its parent <Attribute>'s AttributeId and DataType.

There can be multiple <AttributeValue> elements in an <Attribute>; however, the XACML service allows only one of them. If there is more than one, an indeterminate result is returned and a processing-error status is reported.

An <Attribute> without an <AttributeValue> results in a SOAP fault because it does not follow the schema.

If there is an Issuer attribute in <Attribute>, it is currently ignored by the XACML service.

The DataType of the <Attribute> is usually ignored, because the content of the <AttributeValue> is regarded only as a string type.

Subject

The <Subject> consists of a sequence of <xacml-context:Attribute> elements associated with the subject.

```
<xs:element name="Subject" type="xacml-context:SubjectType"/>

<xs:complexType name="SubjectType">

    <xs:sequence>

        <xs:element ref="xacml-context:Attribute" minOccurs="0"
maxOccurs="unbounded"/>

    </xs:sequence>

    <xs:attribute name="SubjectCategory" type="xs:anyURI"

default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"/>

</xs:complexType>
```

The <Subject> element has an optional attribute, SubjectCategory. This represents the subject's role of this request. For example, 'access-subject', 'requesting-machine', 'code-base'.

Inside a <Subject>, there could be an <Attribute> with AttributeId of "urn:oasis:names:tc:xacml:1.0:subject-category", and the <AttributeValue> of this <Attribute> also represents the subject category. This is equal to the "SubjectCategory" attribute of the <Subject>.

For example, the following two <Subject> elements are functionally equal:

```
<Subject>

        <Attribute
AttributeId="urn:oasis:names:tc:xacml:2.0:subject-category"
DataType="xs:anyURI">
<AttributeValue>urn:oasis:names:tc:xacml:1.0:subject-category:access-subje
ct</AttributeValue>

        </Attribute>

    </Subject>
```

```
<Subject
SubjectCategory=”urn:oasis:names:tc:xacml:1.0:subject-category:access-subj
ect”>
    </Subject>
```

Multiple subjects

There can be multiple subjects in a `<Request>` element.

The XACML context specifies that these subjects should be "disjunctive," which is defined in the standard as a sequence of predicates combined using the logical ‘OR’ operation.

For example, one subject may represents the human user who initiates the request, another may represent the code that is doing the request. They are specified by subject categories.

The rules of handling multiple subjects are as follows:

- If the two subjects have the same subject categories, the elements inside them are treated as if they are in one `<Subject>` element.
- For one request, there must be at least one `<Subject>` having the “`access-subject`” category, which represents the direct accessing subject. For the `<Attribute>` elements of this category, there should be no more than one `<Attribute>` element with the following `AttributeId`, which represents the requester’s identity: `urn:oasis:names:tc:xacml:1.0:subject:subject-id`.

  The subject-id `<Attribute>` is described in “The subject id <Attribute>” on page 5-23.
- For other `<Subject>` elements or other `<Attribute>` elements that are not a subject-id, they are converted as an AquaLogic Enterprise Security authorization context (AppContext).

The subject id <Attribute>

The `DataType` attribute of the subject-id `<Attribute>` element indicates the subject type. The type is of “`anyURI`” type, but in order to be understood, the `DataType` should be prefixed with the identifier that represents the XACML service: `http://security.bea.com/ssmws/ssm-ws-1.0.wsdl`.

In addition, a fragment should be suffixed after this URL indicating the subject type that could be asserted in AquaLogic Enterprise Security; that is, the first parameter of the `assertIdentity()` method in the Authentication service API.

For example, if the subject contains ALESIdentityAssertion, it should have the DataType `http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#ALESIdentityAssertion`, and the `<AttributeValue>` of this `<Attribute>` is taken as the subject.

However, if a valid `<AttributeValue>` is provided and correctly recognized by the Web Service SSM, the DataType may not be checked.

If the `<Subject>` (which is of 'access-subject' category) does not contain an `<Attribute>` with "subject-id", this subject will be considered as an anonymous user. The service will construct an anonymous identity and pass it to the authorization service.

Also note that this implementation reuses the Web Service SSM's identity cache for better performance, as described in "Using the Web Services Client Authorization Cache" on page 3-1.

Mapping Other Subject and Attribute Elements to AquaLogic Enterprise Security Identity

For other `<Subject>` elements and <Attribute> elements that are not in the 'access-subject' category, or are in the 'access-subject' category but without a "subject-id" id, those `<Attribute>` elements are converted as authorization context (AppContext) of AquaLogic Enterprise Security.

If there are multiple `<Attribute>` elements with the same id, a later one will override the previous one.

Also, only text values in `<AttributeValue>` are acceptable; child XML elements are ignored.

The rule for converting an attribute to an AquaLogic Enterprise Security context key-value pair is as follows:

1. If the AttributeId is a URN (for example, "`urn:oasis:names:tc:xacml:1.0:subject:authn-locality:authentication-method`"), the last field of the URN, that is, the string after the last colon, is taken as the context key, and the AttributeValue is the context value. For example, if the previous URN has an AttributeValue of "username/password", the resulting ALES context will be "authentication-method" : "username/password".

2. If the AttributeId is a URL (for example, "http://security.bea.com/ssmws/ssm-ws-1.0.wsdl/schema#Cache-Control"), the fragment (the string after the '#' character) of the URL is taken as the context key. The example URL is converted as "Cache-Control".

   Note that this is a XACML extension: the XACML standard does not have this type of URL as AttributeId, but the PEP could use this method to pass some context to AquaLogic Enterprise Server system.

3. If none of the above matches, the full URI is used as the context key. The value is always the AttributeValue in this Attribute.

Mapping Resources and the AquaLogic Enterprise Security Resource

At least one of the resource attributes should have the following AttributeId: `urn:oasis:names:tc:xacml:2.0:resource:resource-id`.

The AttributeValue of this resource attribute should be the resource that can be understood by AquaLogic Enterprise Security; that is, the resource under the binding point of the SSM. The example queries the user for the resource name, and suggests a value of "store/book."

The DataType of this attribute should always be the XML schema standard “string” type: `http://www.w3.org/2001/XMLSchema#string`.

Other attributes in <Resource>, except for the one associated with the naming authority, are converted as authorization context attributes for authorization evaluation in AquaLogic Enterprise Security, using the same rule as <Subject> attributes.

Mapping Actions and the AquaLogic Enterprise Security Action

At least one of the attributes in the <Action> element must have the following AttributeId: `urn:oasis:names:tc:xacml:1.0:action:action-id`.

The attribute value of this attribute is the action performed on the resource. For example: GET, POST. The example queries the user for the action name.

The DataType of this attribute should be the XML schema standard string `http://www.w3.org/2001/XMLSchema#string`.

Other attributes in `<Action>` are converted as context attributes for authorization evaluation in AquaLogic Enterprise Security, using the same rule as `<Subject>` attributes.

How the Environment Element is Interpreted in AquaLogic Enterprise Security

The XACML service converts environment attributes to AquaLogic Enterprise Security application context. The rule to convert environment elements to AquaLogic Enterprise Security contexts is the same as for the subject. All attributes in `<Environment>` elements are converted.

For example, a sample environment element is as follows:

```
<Environment>
```

```
        <Attribute
AttributeId="http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#DOB"
DataType="http://www.w3.org/2001/XMLSchema#dateTime">
            <AttributeValue>1979-03-29</AttributeValue>
        </Attribute>
    </Environment>
```

The fragment (the string after the '#' character) of the URL is taken as the context key and the AttributeValue is the context value, which results in an AquaLogic Enterprise Security context of `DOB:1979-03-29`.

Sample Request

Listing 5-2 shows a full sample of a XACML request, without the SOAP envelope.

Listing 5-2 XACML Request

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
    <Subject>
        <Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="
http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#ALESIdentityAssertion">
            <AttributeValue>
               <ALESIdentityAssertion xmlns="…">……</ALESIdentityAssertion>
            </AttributeValue>
        </Attribute>
    </Subject>
    <Resource>
        <Attribute
AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string">
            <AttributeValue>/cgi-bin/j_security_check</AttributeValue>
```

```
        </Attribute>
    </Resource>
    <Action>
        <Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string">
            <AttributeValue>POST</AttributeValue>
        </Attribute>
    </Action>
    <Environment/>
</Request>
```

How the AquaLogic Enterprise Security XACML Response is Generated

The `<Response>` element is the root element of a response. It contains one or more `<Result>` elements, which contain authorization decision results. The XACML service supports only one <Result> element.

The <Result> element is defined as follows:

```
<xs:element name="Result" type="xacml-context:ResultType"/>
<xs:complexType name="ResultType">
    <xs:sequence>
        <xs:element ref="xacml-context:Decision"/>
        <xs:element ref="xacml-context:Status" minOccurs="0"/>
        <xs:element ref="xacml:Obligations" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="ResourceId" type="xs:string" use="optional"/>
</xs:complexType>
```

The <Result> element has a ResourceId attribute that indicates the identifier of the request resource element. This implementation does not use this value.

Mapping Decision.

Decision is the evaluation result. It contains a string value of "Permit", "Deny", "Indeterminate", or "NotApplicable". "Permit" or "Deny" are typically returned. For example: <Decision>Permit</Decision>.

If some required attribute is required, "Indeterminate" is returned, with the "missing-attribute" status.

Mapping Status

The <Status> element is an optional element of <Result>. It contains the following elements:

- <StatusCode>

  <StatusCode> has an attribute named "Value," which can be one of the following values:

  – urn:oasis:names:tc:xacml:1.0:status:ok

  – urn:oasis:names:tc:xacml:1.0:status:missing-attribute

  – urn:oasis:names:tc:xacml:1.0:status:syntax-error

  – urn:oasis:names:tc:xacml:1.0:status:processing-error

  If any exception occurred during the authorization process, processing-error is returned.

  If some attribute is required for evaluating the policy, missing-attribute is returned.

  Otherwise, "ok" is returned.

  "syntax-error" is never returned.

- <StatusMessage>(Optional)

  <StatusMessage> is simply a string to show the message accompanying the status. If an exception occurs, the exception will be in the <StatusMessage> for information purpose.

- <StatusDetail>(Optional)

  <StatusDetail> is an extension point and it can contain anything. This implementation puts a <MissingAttributeDetail> value in it if the status is "missing-attribute". In that case, the AttributeId is the missing attribute key, prefixed with the XACML service namespace (`http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#`) and there is no `<AttributeValue>` in the `<MissingAttributeDetail>` element.

Mapping Obligations

A XACML obligation is an operation specified in a policy or policy set that should be performed by the PEP in conjunction with the enforcement of an authorization decision.

This implementation places the AquaLogic Enterprise Server evaluation returned values, like roles or the report_as values, in an obligation. Obligations are used only when the authorization result is “Permit.”

```
<xs:element name="Obligations" type="xacml:ObligationsType"/>

    <xs:complexType name="ObligationsType">

        <xs:sequence>

            <xs:element ref="xacml:Obligation" maxOccurs="unbounded"/>

        </xs:sequence>

    </xs:complexType>

<xs:element name="Obligation" type="xacml:ObligationType"/>

    <xs:complexType name="ObligationType">

        <xs:sequence>

            <xs:element ref="xacml:AttributeAssignment" minOccurs="0"
maxOccurs="unbounded"/>

        </xs:sequence>

       <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>

        <xs:attribute name="FulfillOn" type="xacml:EffectType"
use="required"/>

    </xs:complexType>


    <xs:element name="AttributeAssignment"
type="xacml:AttributeAssignmentType"/>

    <xs:complexType name="AttributeAssignmentType" mixed="true">

        <xs:complexContent mixed="true">

            <xs:extension base="xacml:AttributeValueType">
```

```
                <xs:attribute name="AttributeId" type="xs:anyURI"
use="required"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```

Obligations consist of one or more <Obligation> elements. An <Obligation> has the following attributes:

- FulfillOn: could be “Permit” or “Deny.” This obligation should be fulfilled if the effect (result) is permit or deny. This implementation always puts “Permit” in this attribute.
- ObligationId: the identifier of this obligation. Two types of obligation id are available in this implementation:
  - `http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#ResponseAttributes` represents a set of response attributes.
  - `http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#Roles` represents the roles when performing authorization.

The `<Obligation>` element has zero or more `<AttributeAssignment>` elements. `<AttributeAssignment>` has an AttributeId and text content. The AttributeId is the returned attribute key, prefixed with the namespace http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#, and the content is the attribute value.

For example, the response attribute “currency” : ”USD” is returned as:

```
<AttributeAssignment
AttributeId="http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#currency"
DataType="http://www.w3.org/2001/XMLSchema#string">USD
</AttributeAssignment>
```

A role looks as follows:

```
<AttributeAssignment
AttributeId="http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#role"
DataType="http://www.w3.org/2001/XMLSchema#string">Admin
</AttributeAssignment>
```

Sample Response

Listing 5-3 shows a complete sample response, without the SOAP envelope.

Listing 5-3 XACML Response

```
<?xml version="1.0" encoding="UTF-8"?>
<Response xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
   <Result>
      <Decision>Deny</Decision>
      <Status>
          <StatusCode
Value="urn:oasis:names:tc:1.0:status:processing-error"/>
          <StatusMessage>Authentication required</StatusMessage>
      </Status>
      <Obligations>
        <Obligation FulfillOn="Permit"
ObligationId="http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#Roles">
           <AttributeAssignment
DataType="http://www.w3.org/2001/XMLSchema#string"

AttributeId="http://security.bea.com/ssmws/ssm-ws-1.0.wsdl#role">
              Admin
           </AttributeAssignment>
          </Obligation>
      </Obligations>
   </Result>
</Response>
```

WSDL Definition of the XACML Service

Listing 5-4 shows the WSDL definition of the XACML Service.

Listing 5-4 The WSDL File for the XACML Service

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
    xmlns:ssm="http://security.bea.com/ssmws/ssm-soap-types-1.0.xsd"
    xmlns:tns="http://security.bea.com/ssmws/ssm-ws-1.0.wsdl"
    targetNamespace="http://security.bea.com/ssmws/ssm-ws-1.0.wsdl">
<types>
    <!-- note this policy schema is not a full one. it is stripped for the
XACML runtime only -->
    <xs:import namespace="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
schemaLocation="access_control-xacml-2.0-policy-schema-os.xsd"/>
    <xs:import namespace="urn:oasis:names:tc:xacml:2.0:context:schema:os"
schemaLocation="access_control-xacml-2.0-context-schema-os.xsd"/>
    <!-- for fault types -->
    <xs:import
namespace="http://security.bea.com/ssmws/ssm-soap-types-1.0.xsd"
schemaLocation="ssm-soap-types.xsd"/>
  </types>
<message name="XACMLAtzDecisionRequest">
    <part name="param" element="xacml-context:Request"/>
  </message>
```

```
  <message name="XACMLAtzDecisionResponse">
    <part name="param" element="xacml-context:Response"/>
  </message>
  <message name="XACMLFault">
    <part name="fault" element="ssm:xacmlFailure"/>
  </message>
  <message name="serviceFault">
    <part name="fault" element="ssm:serviceFailure"/>
  </message>
<portType name="XACMLPort">
    <operation name="authorize">
      <input message="tns:XACMLAtzDecisionRequest"/>
      <output message="tns:XACMLAtzDecisionResponse"/>
      <fault name="serviceFault" message="tns:serviceFault" />
      <fault name="xacmlFault" message="tns:XACMLFault"/>
    </operation>
  </portType>
<binding name="XACMLBinding" type="tns:XACMLPort">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
   <!-- Axis has problem support this wrapped style, use document instead.
-->
    <operation name="authorize">
      <soap:operation soapAction="ssmws:xacml:authorization"
style="document"/>
      <input>
        <soap:body use="literal"/>
      </input>
```

```
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="serviceFault">
        <soap:fault use="literal"/>
      </fault>
      <fault name="xacmlFault">
        <soap:fault use="literal"/>
      </fault>
    </operation>
  </binding>
<service name="XACMLService">
    <port name="XACMLAuthorization" binding="tns:XACMLBinding">
      <soap:address location="http://localhost/XACMLAuthorization"/>
    </port>
  </service>
</definitions>
```

Auditing Service Interface

The Auditing Service logs events based on activity related to enterprise security. The Web Services Security Service Module (SSM) runtime uses the Auditing Service to log appropriate data when events occur. The Auditing Service is based on an event model. When something of note occurs, an auditing event is automatically logged. A user or a application that wants notification when a particular event occurs can derive a new class from the `AuditRecord` class.

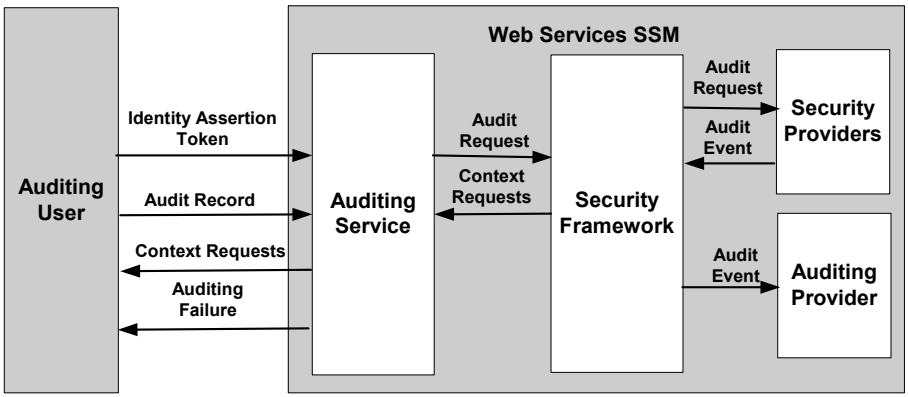
The following topics provide more information on the Auditing Service.

- "Auditing Process" on page 5-35
- "Auditing Service Method" on page 5-36

Auditing Process

The auditing process is as follows (see Fig):

Figure 5-5 Auditing Process



1. During the auditing process, the Web Services SSM audit logging function supports automated, centralized logging of audit messages. To capture a particular event for notification purposes an auditing user can use the `recordEvent()` method to specify the name of the audit record to be captured, and, optionally, the identity assertion token of the auditing user, and the application context.
2. If the auditing provider requires application context that is not included in the `recordEvent()` method, the Auditing Service returns requests for additional application context.
3. If parameter-related audit logging failures occur, including passing in an invalid identity assertion token for the user, an `AuditingFailure` SOAP Fault is returned to the caller.
4. If no errors occur during the auditing process, an empty response is returned to the auditing user as an event notification and the audit event is logged by the auditing provider.

For more information on the Auditing Service interface and the methods it supports, see *WSDLdocs for Web Services Interfaces*.

Auditing Service Method

The Auditing Service passes the audit event to the Web Services SSM runtime. Based on its configuration, the SSM runtime routes the event to the proper auditing providers so that it can be recorded.

The Auditing Service supports a single method, `recordEvent()`. This method accepts an audit record, and, optionally, an identity assertion token, representing the auditing user, and an application context.

Table 5-7 describes the `recordEvent()` method parameters.

Table 5-7 recordEvent() Method Parameters

Parameter	Description	Supported Values/Types
Input Parameters		
`AuditRecord`	Specifies the type of the audit record that encapsulates the logging information.	`type="ssm:AuditRecord Type"`
`IdentityAssertion`	An identity assertion token that represents the authenticated identity of the auditing user.	`type="ssm:Identity AssertionType"`
`AppContext`	Specifies the application context in which request for an auditing record is being made.	`ssm:ContextType` Where `ssm:ContextType` can be one of the following: `StringValue/string`, `BoolValue/boolean`, `DateTimeValue/dateTime`, `TimeValue/time`, `IntValue/integer` `IpValue/ssm:IpType`

Role Mapping Service Interface

The Role Mapping Service allows an application to extract role information about specific identities and resources within the context of the application. These roles can then be used for customizing an interface or for other purposes.

Note: Do not use roles by themselves for authorization, because many policies, allowing or disallowing access to a resource, may be written against a role. Use the Authorization Service to determine actual access rights.

The Role Mapping Service evaluates an interaction of an identity with a resource within an application context and returns a list of role names associated with the configuration of that identity. These roles can change with every resource or be static for the identity across all resources. The roles assigned to an identity are determined by the security policy.

The Web Services SSM is capable of retrieving the roles that a user may have for the given resource and action combination. The user identity is passed as an identity assertion token, instead of a Java object. To obtain roles, authenticated users must be authorized to obtain their roles for the given resource/action combination.

The Role Mapping Service requires that the application pass in a valid identity, a valid resource, and a valid action. The application context is optional and may be set to null if no context is passed in.

The following topics provide more information on the Role Mapping Service.

- "Role Mapping Process" on page 5-37
- "Role Mapping Service Method" on page 5-38

Role Mapping Process

The role mapping process is as follows (see Figure 5-4):

1. During the role mapping process, the Web Services SSM provides a mechanism for optionally specifying the application context to support role mapping.
2. If parameter-related role mapping failures occur, including passing in an invalid identity assertion token for the user, a `RoleMappingFailure` SOAP Fault is returned to the caller.
3. If no errors occur during the role mapping process, a list of zero or more roles, configured in the policy for the provided user/resource/action combination, is returned to the caller. An empty list is a valid response and does not result in a SOAP Fault.

For more information on the Role Mapping Service interface and the methods it supports, see *WSDLdocs for the Web Services Interfaces*.

Role Mapping Service Method

The Role Mapping Service interface supports one method, `getRoles()`. This method gets the roles for an authenticated user identity in reference to a `RuntimeResource`, `RuntimeAction`, and an optional `Context`.

The `getRoles()` method accepts a supported type of an identity token, and, optionally, runtime resource and action structures, and an application context. It returns either a list of user roles associated for the identity and a time-to-live parameter for user roles.

Table 5-8 describes the `getRoles()` method parameters.

Table 5-8 getRoles Method Parameters

Parameter	Description	Supported Values/Types
Input Parameters		
`IdentityAssertion`	An identity assertion token that represents the authenticated identity of the user. The token type must be supported by the Role Mapping provider; otherwise, the service returns `RoleMappingFailure` to the caller.	`IdentityAssertion` `type="ssm:IdentityAssertion Type"`
`RuntimeResource` resource	Specifies the runtime resource that the caller is requesting.	`ResourceString, AuthorityName` `type="ssm:RuntimeResource Type"`
`RuntimeAction action`	Specifies the runtime action that the caller is requesting.	`ActionString,` `AuthorityName` `type="ssm:Runtime ActionType"`

Table 5-8 getRoles Method Parameters (Continued)

Parameter	Description	Supported Values/Types
`AppContext`	Specifies the application context in which access to the resource is being requested.	`type="ssm:ContextType"` Where `ssm:ContextType` can be one of the following: `StringValue/string`, `BoolValue/boolean`, `DateTimeValue/dateTime`, `TimeValue/time`, `IntValue/integer` `IpValue/ssm:IpType`
Return Parameters		
Roles	Specifies a list roles granted to the user.	type="ssm:IdentityRoleType"
RolesTtlAdvice	Specifies time to live for user roles.	type="xsd:positiveInteger

Credential Mapping Service Interface

The Credential Mapping Service allows an client application to fetch credentials of certain types that are associated with a specific identity for a specific resource. These credentials can then be used on behalf of that identity to execute some privileged function, such as logging into a database or sending e-mail.

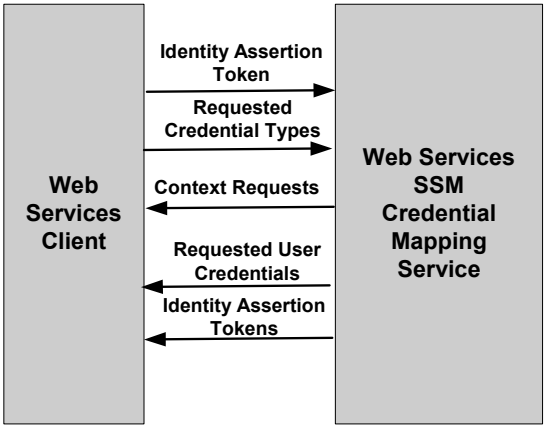
The following topics provide more information on the Credential Mapping Service.

- "Credential Mapping Process" on page 5-39
- "Credential Mapping Method" on page 5-41

Credential Mapping Process

The credential mapping process is as follows (see Figure 5-6):

Figure 5-6 Credential Mapping Process



1. The client presents a get credentials request to the Web Services SSM. The request includes a supported type of an identity assertion token and a list of requested credential types. Optionally, the request can include a runtime resource, runtime action, and an application context.
2. In response, the Web Services SSM returns a list of requested user credentials and identity assertion tokens or, if required by the Credential Mapping provider, additional context requests.
3. The Web Services SSM returns the following types of credentials in response to the client's request:
   - Username/password pairs
   - Signed SAML 1.1 assertions
   - ALES cookies
4. If parameter-related credential mapping failures occur, the Web Service returns a `CredentialMappingFailure` SOAP Fault to the caller.
5. If no errors occur during the credential mapping process, the Web Services SSM returns a list of zero or more credentials that are configured in the security policy for the specified user/resource/action combination. An empty list is a valid response and does not result in a SOAP Fault. Also, if some of the requested credential types are not available for specified user/resource/action combination, a SOAP Fault does not result.

For more information on the Credential Mapping Service interface and the methods it supports, see WSDLdocs *for Web Services Interfaces*.

Credential Mapping Method

The Credential Mapping Service supports one method: `getCredentials()`. This method accepts a supported type of an identity assertion token and a list of requested credential types. Optionally, this method can accept an identity assertion token that represents the identity of a different user and a runtime resource structure, which includes the requested resource and action and the application context. In response, the `getCredentials()` method returns either a list of requested user credentials, identity assertion tokens, and a list of any missing credential types.

Note: Since password credentials need to be returned in clear text to the caller in order to be usable for authentication in external systems, you should pay particular attention to providing channel and message security to protect messages in transit between clients and the Web Service. At a minimum, you must use a channel security protocol, such as SSL or TLS, for all communication.

Authenticated users are always authorized to obtain their own credentials for the given resource/action combination. However, authenticated user cannot requests credentials on behalf of another user.

Credential mapping process involves issuing new types of credentials to the specified combination of user, resource, and action. The user identity is passed as an identity assertion token, instead of a Java object.

Table 5-9 describes the `getCredentials()` method parameters.

Table 5-9 getCredentials Method Parameters

Parameter	Description	Supported Values/Types
Input Parameters		
`IdentityAssertion`	An identity assertion token that represents the authenticated identity of the user. The type used must be supported by the ALES Credential Mapping provider; otherwise, the service returns `CredentialMappingFailure` to the caller.	`IdentityAssertion` `ssm:"IdentityAssertion Type"`

Table 5-9 getCredentials Method Parameters (Continued)

Parameter	Description	Supported Values/Types
`Requested CredentialTypes`	Specifies a list of the types of credentials being requested. The Web Services SSM supports the following types of credentials: • Username/password pairs • Signed SAML 1.1 assertions • ALES proprietary cookie	`type="ssm:CredentialTypes Type"` Any non-empty string consisting of any number of alphanumeric characters and these separators: period (“.”), comma (“,”), and underscore (“_”).
`RuntimeResource`	Specifies the runtime resource that the caller is requesting.	`ResourceString, AuthorityName` `type="ssm:RuntimeResource Type"`
`RuntimeAction`	Specifies the runtime action that the caller is requesting.	`ActionString, AuthorityName` `type="ssm:Runtime ActionType"`
`AppContext`	Specifies the application context in which access to the resource is being requested. Specified as a name/value pair.	`ssm:"ContextType"` Where `ssm:ContextType` can be one of the following: `StringValue/string, BoolValue/boolean, DateTimeValue/dateTime, TimeValue/time, IntValue/integer IpValue/ssm:IpType`
Return Parameters		
`MissingTypes`	List of missing credential types.	type="xsd:string"
`Identity Credential`	List of user credentials.	type="ssm:IdentityCredentialType"
`IdentityAssertion`	Identity assertion tokens that represent the authenticated identity of the user.	`IdentityAssertion` type="ssm:IdentityAssertionType"

Index