Oracle® Fusion Middleware

Developer's Guide for Oracle Access Manager and Oracle Security Token Service

11g Release 1 (11.1.1)

E12491-03

June 2011

ORACLE®

Oracle Fusion Middleware Developer's Guide for Oracle Access Manager and Oracle Security Token Service
11g Release 1 (11.1.1)

E12491-03

Primary Author: Trish Fuzesy

Contributing Author: Toby Close, Peter Povenic, Umesh Waghode, Svetlana Klomeyskaya, Resha Chhada, Gail Flanegin

Content

Preface

This guide explains how to write custom applications and plug-ins to functions programmatically, to create custom AccessGates that protect non-Web-based resources.

This Preface covers the following topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

This document is intended for administrators who are familiar with Oracle Access Manager and Oracle Security Token Service.

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

Related Documents

For more information, see the following documents in the Oracle Fusion Middleware 11g Release 1 (11.1.1) documentation set:

- *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*
- *Oracle Security Token Service Java API Reference*
- *Oracle Access Manager Access SDK Java API Reference*
- *Oracle Access Manager Extensibility Java API Reference*
- *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*

Conventions

The following text conventions are used in this document:

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

What's New in Oracle Access Manager?

This section describes new features of Oracle Access Manager 11g.

The following sections describe the new features in Oracle Access Manager that are reflected in this book:

- Product and Component Name Changes
- Oracle Access Manager 11g Software Developer Kit

Product and Component Name Changes

Many Oracle Access Manager component names remain the same. However, there are several important changes that you should know about, as shown in the following table:

Area	Oracle Access Manager 10g	Oracle Access Manager 11g
Deployment	Stand alone server	Deployed in a container
Component Names	Access Server Policy Manager Identity Server WebPass	OAM Server OAM Administration Console N/A N/A
Agents	WebGate AccessGate	OAM Agent OAM Agent
Console Names	Policy Manager Identity System Console Access System Console	OAM Administration Console N/A N/A
Directory Profiles	Directory Profiles	User-Identity Stores
Identity Administration	Identity Server	Identity agnostic (Oracle Identity Manager 11g is used by default)
Administrators	Master Administrator Master Identity Administrator Master Access Administrator Delegated Administrators	OAM Administrator N/A N/A N/A
Agent and partner application registration	N/A	OAM Administration Console Remote registration tool provides automated Agent registration and application domain creation with default security policies

Area	Oracle Access Manager 10g	Oracle Access Manager 11g
Automated creation of Oracle Access Manager 10g form-based authentication scheme, policy domain, access policies, and WebGate profile for the Identity Asserter for single sign-on	OAMCfgTool Platform-agnostic tool and scripts	N/A
Configuration Store	LDAP	XML file
Policy Store	LDAP	RDBMS
Policy Model	Open (default allow)	Closed (default = deny access)
Policy Domain	Policy Domain	Application Domain
Session management	Stateless, stored in a cookie	Stateful, stored on the server
Authentication to LDAP	LDAP defined system wide	LDAP defined in an authentication scheme
Resource Types	Resource Type	Resource Type
Resources	Resource	Resource
Host Identifiers	Host Identifiers	Host Identifiers
Authentication	Authentication Authentication Scheme Authentication Plug-ins N/A Authentication Rule	Authentication Authentication Scheme Authentication Plug-ins Authentication Modules Authentication Policy
Authorization	Authorization Authorization Rule Authorization Expression	Authorization Constraint Authorization Policy
Actions	Actions	Responses
Software Developer Kit	Access SDK	Access SDK
Access Protocol	NetPoint Access Protocol (NAP)	Oracle Access Protocol (OAP)
Access Protocol port number	6021	5575 (assigned by the Internet Assigned Numbers Authority (IANA))

Oracle Access Manager 11g Software Developer Kit

Oracle Access Manager 11g provides a pure Java software developer kit (SDK) for the creation of custom AccessGates and extensions of authentication and authorization functionality. Oracle Access Manager 11g also provides compatibility with the Oracle Access Manager 10g JNI SDK, which can be migrated to use the Oracle Access Manager 11g.

1 Introduction to this Book

This chapter provides the following sections to introduce this book:

- Chapter 2: Introduction to the Access SDK and API
- Chapter 3: Creating Custom Authentication Plug-ins
- Chapter 4: Writing Oracle Security Token Service Module Classes
- Introduction to Java API References

1.1 Chapter 2: Introduction to the Access SDK and API

Oracle Access Manager 11g provides a pure Java software developer kit (SDK) for the creation of custom Access Clients and extensions of Oracle Access Manager 11g authentication and authorization functionality and custom tokens. Oracle Access Manager 11g also provides backward compatibility with the Oracle Access Manager 10g JNI SDK, which can be migrated and used with Oracle Access Manager 11g.

Chapter 2 provides the following sections:

- Section 2.1, "Introduction to the Access SDK"
- Section 2.2, "Locating Access SDK Packages and Resources"
- Section 2.3, "Uses, Functionality, and New Features"
- Section 2.4, "Messages, Exceptions and Logging"
- Section 2.5, "Configuring and Deploying Access Clients"
- Section 2.6, "Developing Access Clients"
- Section 2.7, "Building and Deploying an Access Client Program"
- Section 2.8, "Compatibility: 11g versus 10g Access SDK and APIs"
- Section 2.9, "Migrating Earlier Applications or Converting Your Code"
- Section 2.10, "Best Practices"

1.2 Chapter 3: Creating Custom Authentication Plug-ins

The OAM Server uses both authentication and authorization controls to limit access to the resources that it protects. Authentication is governed by specific authenticating schemes, which rely on one or more plug-ins that test the credentials provided by a user when he or she tries to access a resource. The plug-ins can be taken from a standard set provided with OAM Server installation, or custom plug-ins created by your own Java developers.

Chapter 3 provides the following sections:

- Section 3.1, "Introduction to Authentication Plug-ins"
- Section 3.2, "Introduction to Plug-in Interfaces"
- Section 3.3, "Sample Code: Custom Database User Authentication Plug-in"
- Section 3.4, "Developing an Authentication Plug-in"
- Section 3.5, "Adding Custom Plug-ins"
- Section 3.6, "Creating a Custom Authentication Module for Custom Plug-ins"
- Section 3.7, "Creating Authentication Schemes with Custom Authentication Modules"
- Section 3.8, "Configuring Logging for Custom Plug-ins"

1.3 Chapter 4: Writing Oracle Security Token Service Module Classes

When Oracle Security Token Service does not support the token that you want to validate or issue out-of-the-box, you can write your own validation and issuance module classes. One of the following two (validation or issuance class) is required for custom tokens:

- Custom validation class uses Oracle Security Token Service to validate a custom token
- Custom issuance class enables Oracle Security Token Service to issue a custom token

Chapter 4 discusses Oracle Access Manager 11g and Oracle Security Token Service custom token options. It includes the following sections:

- Section 4.1, "Introduction to Oracle Security Token Service Custom Token Module Classes"
- Section 4.2, "Writing a TokenValidatorModule Class"
- Section 4.3, "Writing a TokenIssuanceModule Class"
- Section 4.4, "Making Custom Classes Available"
- Section 4.5, "Managing a Custom Oracle Security Token Service Configuration"

1.4 Introduction to Java API References

Specific Java API reference details for this release are provided the following publications:

- *Oracle Access Manager Access SDK Java API Reference*
- *Oracle Access Manager Extensibility Java API Reference*
- *Oracle Security Token Service Java API Reference*

2

Introduction to the Access SDK and API

This chapter provides the following sections:

2.1 Introduction to the Access SDK

The Oracle Access Manager 11g Access SDK is a platform independent package that Oracle has certified on a variety of enterprise platforms (using both 32-bit and 64-bit modes) and hardware combinations. It is provided on JDK versions that are supported across Oracle Fusion Middleware applications.

The `oracle.security.am.asdk` package provides the Oracle Access Manager 11g version of the Application Programming Interface (API). The 11g version is very similar to the Oracle Access Manager 10g API, with enhancements for use with the OAM 11g Server. The 11g Access SDK provides backwards compatibility by supporting `com.oblix.access` interfaces.

See Also: *Oracle Security Token Service Java API Reference*

The Oracle Access Manager 10g (10.1.4.3) `com.oblix.access` package and classes are deprecated. A deprecated API is not recommended for use, generally due to improvements, and a replacement API is usually given. Deprecated APIs may be removed in future implementations.

Note: Oracle strongly recommends that developers use the Oracle Access Manager 11g Access SDK for all new development.

2.2 Locating Access SDK Packages and Resources

Once the Access SDK is installed, do *not* change the relative locations of the subdirectories and files. Doing so may prevent an accurate build and proper operation of the API.

Table 2–1 identifies the Access SDK packages and resources and where you can find each one.

Table 2–1 Locations: Access SDK Resources

Resources and Locations
Supported Versions and Platforms: Oracle Technology Network
`http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html`
Download Packages: Oracle Technology Network
`http://www.oracle.com/technetwork/middleware/downloads/oid-11g-161194.html`
Oracle Access Manager 11g Access SDK Packages:
■ **oracle.security.am.asdk**: A new authentication and authorization API that provides enhancements to take advantage of Oracle Access Manager 11g Server functionality. The Oracle Access Manager 11g version of the Access SDK API can be used with either Oracle Access Manager 10gR3 (10.1.4.3) or Oracle Access Manager 11gR1 (11.1.1.5+) version of the Server.
■ **com.oblix.access**: Available for compatibility with programs written with an Oracle Access Manager 10g JNI ASDK, this is the 10g version of the authentication and authorization API with some enhancements for Oracle Access Manager 11g.
Java Docs:
■ *Oracle Access Manager Access SDK Java API Reference*
■ *Oracle Access Manager Extensibility Java API Reference*
■ *Oracle Security Token Service Java API Reference*
Each method includes the following details:
■ Comprehensive description of a method
■ Parameters of the method
■ Return values
■ Exceptions the method may throw
■ Other relevant details

2.3 Uses, Functionality, and New Features

The Oracle Access Manager 11g Access SDK is intended for Java application developers and the development of tightly coupled, performant integrations.

From a functional perspective, the Oracle Access Manager 11g Access SDK maintains parity with the 10*g* (10.1.4.3) Java Access SDK to ensure that you can re-write existing custom code using the new API layer.

The Oracle Access Manager 11g Access SDK includes authentication and authorization functionality. However, it does not include Administrative APIs (for instance, there is no 11g Policy Manager API) and does not use Oracle Access Manager 11g cookies.

The most common use of the Access SDK is to enable the development of a custom integration between Oracle Access Manager and other applications (Oracle or third party). Usage examples include:

- Accessing session information that may be stored as part of the Oracle Access Manager authentication process.
- Verifying the validity of the Oracle Access Manager session cookie rather than trusting an HTTP header for the principle user.

Another use for the Access SDK is the development a custom Access Client for a Web server or an application server for which Oracle does not provide an out-of-the-box integration.

Table 2–2 describes the primary features of the Oracle Access Manager 11g Access SDK.

Table 2–2 Oracle Access Manager 11g Access SDK Features

Feature	Description
Installation	**Client Package**: Is comprised of a single jar file. Supporting files (for signing and TLS negotiations) are not included and should be generated separately. **Server Related Code**: Is included as part of the core Oracle Access Manager server installation. Note: Access Clients and plug-ins developed with Oracle Access Manager 10*g* (10.1.4.3) can be used with Oracle Access Manager 11g. Oracle Access Manager 10*g* (10.1.4.3) bundle patches are used to distribute Java SDK code enhancements for use with Oracle Access Manager 11g.
Built In Versioning	Enables you to: ■ Determine the Access SDK version that is installed. ■ Validate compatible versions it can operate with (Oracle Access Manager 10*g* (10.1.4.3) and Oracle Access Manager 11g). If there is a mismatch, Access SDK functions halt and an informative message is logged and presented.
Logging	The Access SDK logging mechanism enables you to specify the level (informational, warning, and error level) of detail you want to see in a local file. Messages provide enough detail for you to resolve an issue. For example, if an incompatible Access SDK package is used, the log message includes details about a version mismatch and what version criteria should be followed. If the SDK generates large amounts of logs within a given period of time, you can configure a rollover of the logs based on a file limit or a time period. For example, if a file limit has been reached (or a certain amount of time has passed), the log file is copied to an archive directory and a new log file is started
New Calls	The Access SDK incorporates new calls to determine additional information about the session based on the new Oracle Access Manager 11g architecture. Note: Access Clients and plug-ins developed with the Oracle Access Manager 10g com.oblix.access package can be migrated to operate with the OAM 11g Server.

The Access SDK enables you to develop custom integrations with Oracle Access Manager for the purpose of controlling access to protected resources such as authentication, authorization, and auditing. This access control is generally accomplished by developing and deploying custom Access Clients, which are applications or plug-ins that invoke the Access Client API to interface with the Access SDK runtime.

Access Client-side caching is used internally within the Access SDK runtime to further minimize the processing overhead. The Access SDK runtime, together with the Oracle Access Manager server, transparently performs dynamic configuration management, whereby any Access Client configuration changes made using Oracle Access Manager administration console are automatically reflected in the affected Access SDK runtimes.

You can develop different types of custom Access Clients, depending on their desired function, by utilizing all, or a subset of, the Access Client API. The API is generally agnostic about the type of protected resources and network protocols used to communicate with the users. For example, the specifics of HTTP protocol and any use of HTTP cookies are outside of the scope of Access SDK. You can develop Access Clients to protect non-HTTP resources as easily as agents protecting HTTP resources.

The typical functions that a custom Access Client can perform, individually or in combination with other Access Clients, are as follows:

- Authenticate users by validating their credentials against Oracle Access Manager and its configured user repositories.
- Authenticate users and check for authorization to access a resource.
- Authenticate users and create unique Oracle Access Manager sessions represented by session tokens.
- Validate session tokens presented by users, and authorize their access to protected resources.
- Terminate Oracle Access Manager sessions given a session token or a named session identifier.
- Enumerate Oracle Access Manager sessions of a given user by specifying named user identifier.
- Save or retrieve custom Oracle Access Manager session attributes.

Some Access Client operations are restricted for use by the designated Access Client instances. For example, see `OperationNotPermitted` in *Oracle Access Manager Access SDK Java API Reference*.

An Oracle Access Manager administrator can use the Oracle Access Manager administration console to control the privileges of individual Access Clients. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.

2.4 Messages, Exceptions and Logging

This section describes the messages and exceptions used by the Access SDK to indicate status or errors.

The execution log generated by the Access SDK is also described. The execution log provides information about operations performed. For example, operation status, any errors or exceptions that occur, and any general information that is helpful for troubleshooting.

The following topics are discussed in this section:

- Messages
- Exceptions
- Logging

2.4.1 Messages

The Access SDK provides support for localized messages that indicate status or error conditions. Error messages, which are provided to the application as exceptions, are also localized. These localized error messages are logged in the Access SDK log file.

2.4.2 Exceptions

The following types of exceptions are used to indicate error conditions to an application:

- **OperationNotPermittedException**

  The Oracle Access Manager 11g Access SDK introduces a new set of session management APIs. Only privileged Access Clients can perform these session management operations.

  If the Access client is not allowed to perform these operations, the Oracle Access Manager 11g server returns an error. When the server returns an error, the Access SDK will throw this exception.

- **AccessException**

  The Oracle Access Manager Access SDK API throws an `AccessException` whenever an unexpected, unrecoverable error occurs during the performance of any operation.

2.4.3 Logging

The Access SDK uses Java logging APIs for producing logs. Specifically, the `oracle.security.am.asdk` package contains the `AccessLogger` class, which produces the Access SDK log.

To generate the Access SDK log, you must provide a logging configuration file when you start the application. Provide this log configuration file as a Java property while running the application, where the Java property `-Djava.util.logging.config.file` is the path to `logging.properties`.

For example:

```
java -Djava.util.logging.config.file=JRE_DIRECTORY/lib/logging.properties
```

The `logging.properties` file defines the number of Loggers, Handlers, Formatters, and Filters that are constructed and ready to go shortly after the VM has loaded. Depending on the situation, you can also configure the necessary logging level.

You must provide the log file path against the `java.util.logging.FileHandler.pattern` property in the `logging.properties` file. If you provide only the file name, the file will be created under the current directory.

The following is an example `logging.properties` file:

```
# "handlers" specifies a comma separated list of log Handler
# classes.  These handlers will be installed during VM startup.
# Note that these classes must be on the system classpath.
# By default we only configure a ConsoleHandler, which will only
# show messages at the INFO and above levels.
# Add handlers to the root logger.
# These are inherited by all other loggers.
handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler

# Set the logging level of the root logger.
# Levels from lowest to highest are
# FINEST, FINER, FINE, CONFIG, INFO, WARNING and SEVERE.
# The default level for all loggers and handlers is INFO.
.level= ALL

# Configure the ConsoleHandler.
# ConsoleHandler uses java.util.logging.SimpleFormatter by default.
# Even though the root logger has the same level as this,
# the next line is still needed because we're configuring a handler,
# not a logger, and handlers don't inherit properties from the root logger.
```

```
java.util.logging.ConsoleHandler.level =INFO
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFormatter

# The following special tokens can be used in the pattern property
# which specifies the location and name of the log file.
#   / - standard path separator
#   %t - system temporary directory
#   %h - value of the user.home system property
#   %g - generation number for rotating logs
#   %u - unique number to avoid conflicts
# FileHandler writes to %h/demo0.log by default.
java.util.logging.FileHandler.pattern=%h/asdk%u.log


# Configure the FileHandler.
# FileHandler uses java.util.logging.XMLFormatter by default.
#java.util.logging.FileHandler.limit = 50000
#java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter
java.util.logging.FileHandler.level=ALL
```

The following is a sample of the log output:

```
Apr 19, 2011 5:20:39 AM AccessClient createClient
FINER: ENTRY
Apr 19, 2011 5:20:39 AM ObAAAServiceClient setHostPort
FINER: ENTRY
Apr 19, 2011 5:20:39 AM ObAAAServiceClient setHostPort
FINER: RETURN
Apr 19, 2011 5:20:39 AM ObAAAServiceClient setHostPort
FINER: ENTRY
Apr 19, 2011 5:20:39 AM ObAAAServiceClient setHostPort
FINER: RETURN
Apr 19, 2011 5:20:39 AM AccessClient createClient
FINER: RETURN
Apr 19, 2011 5:20:39 AM AccessClient initialize
FINER: read config from server, re-init if needed
Apr 19, 2011 5:20:39 AM AccessClient updateConfig
FINER: ENTRY
Apr 19, 2011 5:20:39 AM AccessClient readConfigFromServer
FINER: ENTRY
Apr 19, 2011 5:20:39 AM ObAAAServiceClient getClientConfigInfo
FINER: ENTRY
Apr 19, 2011 5:20:39 AM ObAAAServiceClient sendMessage
FINER: ENTRY
Apr 19, 2011 5:20:39 AM oracle.security.am.common.nap.util.NAPLogger log
FINER: Getting object using poolid primary_object_pool_factory
Apr 19, 2011 5:20:39 AM oracle.security.am.common.nap.util.pool.PoolLogger
logEntry
FINER: PoolLogger : main entered: KeyBasedObjectPool.acquireObject
Apr 19, 2011 5:20:39 AM oracle.security.am.common.nap.util.NAPLogger log
FINEST: Creating pool with id = primary_object_pool_factory
Apr 19, 2011 5:20:39 AM oracle.security.am.common.nap.util.pool.PoolLogger log
FINER: PoolLogger:main : Maximum Objects = 1Minimum Objects1
Apr 19, 2011 5:20:39 AM oracle.security.am.common.nap.util.pool.PoolLogger
logEntry
FINER: PoolLogger : main entered: constructObject
Apr 19, 2011 5:20:39 AM oracle.security.am.common.nap.ObMessageChannelImpl <init>
```

2.5 Configuring and Deploying Access Clients

This section describes the configuration steps required before an Access Client developed using the Access SDK can be deployed. For more information, see Section 2.6.1.2, "Access Client Architecture".

After development, the Access Client must be deployed in a live Oracle Access Manager 11g environment before you can test and use it. The Access Client deployment process is similar to that of other Oracle Access Manager Agents.

The following overview outlines the tasks that must be performed by a user with Oracle Access Manager administrator credentials.

> **See Also:** *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*

Task overview: Deploying Access Client Code

It is assumed that the Access Client program is already developed and compiled.

1. Retrieve the Access SDK jar file and copy this to the computer you will use to build the Access Client.
2. Copy the Access Client to the computer hosting the application to be protected.
3. Configure the Access Client.
4. Verify you have the required Java environment available.

   If your Access Client is in a standalone environment, you can use Java Development Kit (JDK) or Java Runtime Environment (JRE). If your Access Client is a servlet application, you can use Java EE or the Java environment available with your Java EE container.
5. Verify that the Access SDK jar file is in the class path.

2.5.1 Configuration Requirements

An Access SDK configuration consists of the following files:

- **Configuration File (ObAccessClient.xml)**

  The configuration file holds various details such as Oracle Access Manager server host, port, and other configuration items that decide behavior of the Access Client. For example, idle session time. Name of this file is ObAccessClient.xml.

- **SSL Certification and Key File**

  This file is required only if the transport security mode is Simple or Cert. Both the Oracle Access Manager 10g Server and Oracle Access Manager 11g Server supports transport security modes Open, Simple and Cert to communicate with agents. An Access Client developed using Access SDK is called an *agent*. Depending on the mode in which Oracle Access Manager server is configured, Access Client will have to be configured to communicate in the same mode.

  For Simple or Cert transport security mode, the following is required:

  – Certificate for the Access Client

  – Private key for the Access Client

  – CA certificate to trust OAM Server's certificate

- **password.xml File**

This file is required only if the transport security mode is Simple or Cert. This file contains a password in encrypted form. This password is the one using which SSL key file is protected.

- **Log Configuration**

  Is required in order to generate a log file.

2.5.2 Generating the Required Configuration Files

The ObAccessClient.xml configuration file can be obtained by registering an Access Client as an OAM 10g Agent with the OAM 11g Server, using the Oracle Access Manager 11g administration console or a remote registration tool. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.

If the transport security mode is given as Simple or Cert mode during registration, the Oracle Access Manager administration console will create an SSL certificate and key file in PEM format. The certificate and key file can then be imported in the oamclient-keystore.jks file. The CA certificate used to issue the certificate and key should be imported into oamclient-truststore.jks. For more information, see Section 2.5.3, "SSL Certificate and Key Files".

The Oracle Access Manager administration console will also create a password.xml file.

An Access Client application developed with the `oracle.security.am.asdk` API can specify the location to obtain the configuration file and other required files. This is done by initializing the Access SDK and providing the directory location where the configuration files exist.

For information about options available to specify location of the configuration files to the Access SDK, see *Oracle Access Manager Access SDK Java API Reference*.

2.5.3 SSL Certificate and Key Files

Oracle Access Manager 11g Access SDK uses SSL certificates and key files from a database commonly known as trust stores or key stores. It requires these stores to be in JKS (Java Key Standard) format.

2.5.3.1 Simple Transport Security Mode

Importing the CA Certificate

The CA certificate must be imported to the trust store. Oracle Access Manager 10g JNI ASDK provides a self-signed CA certificate that can be used in Simple mode, and is used for issuing certificates to the Access Client. OAM 11g Server also provides a self-signed CA certificate.

In Oracle Access Manager 10g JNI ASDK, the CA certificate is found in the following directory and is named cacert.pem: ASDK_INSTALL_
DIR/oblix/tools/openssl/simpleCA.

In OAM 11g Server, the CA certificate is found in the following directory and is named cacert.der: $MIDDLEWARE_HOME/user_projects/domains/base_
domain/config/fmwconfig.

Execute the following command to import the PEM or DER format CA certificate into trust store:

1. Edit ca_cert.pem or cacert.der using a text editor to remove all data except what is contained within the CERTIFICATE blocks, and save the file. For example:

   ```
   -----BEGIN CERTIFICATE-----
   Content to retain
   -----END CERTIFICATE-----
   ```

2. Execute the following command, modifying as needed for your environment:

   ```
   keytool -importcert -file <<ca cert file cacert.pem or
   cacert.der>> -trustcacerts  -keystore
   oamclient-truststore.jks -storetype JKS
   ```

3. Enter keystore password when prompted. This must be same as the global pass phrase used in the OAM Server.

Setting Up The Keystore

The Access Client's SSL certificate and private key file must be added to the keystore. The SSL certificate and private key file must be generated in Simple mode so the Access Client can communicate with OAM Server.

Oracle Access Manager 10g JNI ASDK provides for generating a certificate and key file for the Access Client. These certificates are in PEM format.

OAM 11g Server provides a tool called Remote Registration and Administration Console for generating a certificate and key file for the Access Client. These certificates are also in PEM format. The names of these files are aaa_cert.pem and aaa_key.pem.

Execute the following commands in order to import the certificate and key file into keystore oamclient-keystore.jks.

1. Edit aaa_cert.pem using any text editor to remove all data except that which is contained within the CERTIFICATE blocks, and save the file. For example:

   ```
   -----BEGIN CERTIFICATE-----
   Content to retain
   -----END CERTIFICATE-----
   ```

2. Execute the following command, modifying as needed for your environment:

   ```
   openssl pkcs8 -topk8 -nocrypt -in aaa_key.pem -inform PEM
   -out aaa_key.der -outform DER
   ```

   This command will prompt for a password. The password must be the global pass phrase.

3. Execute the following command, modifying as needed for your environment:

   ```
   openssl x509 -in aaa_cert.pem -inform PEM -out aaa_cert.der
   -outform DER
   ```

4. Execute the following command, modifying as needed for your environment:

   ```
   java -cp importcert.jar
   oracle.security.am.common.tools.importcerts.CertificateImport
   -keystore oamclient-keystore.jks -privatekeyfile aaa_key.der
   -signedcertfile aaa_cert.der -storetype jks -genkeystore yes
   ```

   In this command, `aaa_key.der` and `aaa_cert.der` are the private key and certificate pair in DER format.

5. Enter the keystore password when prompted. This must be same as global pass phrase.

2.5.3.2 Cert Transport Security Mode

In Cert transport security mode, the certificates for the server and agent should be requested from a certifying authority. Optionally, the Simple mode self-signed certificates can also be used as a certifying authority, for purposes of issuing Cert mode certificates.

Follow these steps to prepare for Cert mode:

1. Import a CA certificate of the certifying authority using the certificate and key pair issued for Access Client and OAM Server. Follow the steps in "Importing the CA Certificate" on page 2-8. Instead of cacert.pem or cacert.der, substitute the CA certificate file of the issuing authority.
2. If Oracle Access Manager 10g JNI ASDK install is available, it provides a way to generate certificate and key file for the Access Client. These certificates will be in PEM format.

   For more information about how to generate a certificate using an imported CA certificate, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.

   To import this certificate, key pair in the oamclient-keystore.jks in PEM format, follow instructions in "Setting Up The Keystore" on page 2-9.

2.6 Developing Access Clients

The following topics are discussed in this section:

- Introduction to Access Clients
- Structure of an Access Client

2.6.1 Introduction to Access Clients

Access Clients process user requests for access to resources within the LDAP domain protected by the OAM Server. Typically, you embed custom Access Client code in a servlet (plug-in) or a standalone application that receives resource requests. This code uses Access Manager API libraries to perform authentication and authorization services on the OAM Server.

If a resource is not protected, the Access Client grants the user free access to the requested resource. If the resource is protected and the user is authorized to provide certain credentials to gain access, the Access Client attempts to retrieve those user credentials so that the OAM Server can validate them. If authentication of the user and authorization for the resource succeeds, the Access Client makes the resource available to the user.

Access Clients can differ according to a variety of factors, as described in Table 2–3.

Table 2–3 Access Client Variations

Variation	Description
Type of application	Standalone application versus server plug-ins.
Development Language	Each development language provides a choice of interfaces to the underlying functionality of the API. For Oracle Access Manager 11g, Java is the only development language for custom Access Clients.
Resource Type	Protect both HTTP and non-HTTP resources.

Table 2–3 (Cont.) Access Client Variations

Variation	Description
Credential Retrieval	Enable HTTP FORM-based input, the use of session tokens, and command-line input, among other methods.

2.6.1.1 When to Create a Custom Access Client

Typically, you deploy a custom Access Client instead of a standard WebGate when you need to control access to a resource for which Oracle Access Manager does not already supply an out-of-the-box solution. This might include:

- Protection for non-HTTP resources.
- Protection for a custom web server developed to implement a special feature (for example, a reverse proxy).
- Implementation of single sign-on (SSO) to protect a combination of HTTP and non-HTTP resources.

  For example, you can create an Access Client that facilitates SSO within an enterprise environment that includes an Oracle WebLogic Server cluster as well as non-Oracle WebLogic Server resources.

2.6.1.2 Access Client Architecture

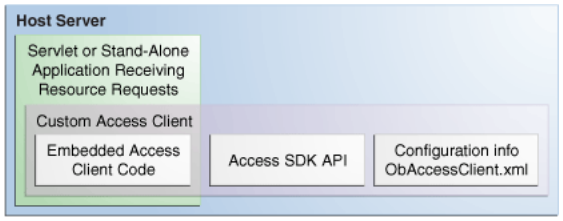
Each Access Client is built from three types of resources, as described in Table 2–4.

Table 2–4 Resources to Build Access Clients

Resource	Description
Custom Access Client code	Built into a servlet or standalone application. For Oracle Access Manager 11g, you write Access Client code using the Java language platform.
Configuration information	■ Primary configuration file required for Access SDK is ObAccessClient.xml, which contains configuration information that constitutes an Access Client profile. ■ Additional configuration artifacts are required depending on the transport security mode (Open, Simple or Cert) in which an Access Client is configured to interact with OAM Server. If security mode is Simple or Cert, then the following files are required. ■ oamclient-truststore.jks – JKS format trust store file which should contain CA certificate of the certificate issuing authority. ■ oamclient-keystore.jks – JKS format key store file which should contain certificate and private key file issued for the Access Client. ■ password.xml – An XML file that holds the value of global pass phrase. Same password is also used to unprotect private key file.
Access Manager API libraries	Facilitate Access Client interaction with the OAM Server.

Figure 2–1 shows Access Client components installed on a host server:

Figure 2–1 Architectural Detail of an Access Client


Host Server
Servlet or Stand-Alone Application Receiving Resource Requests
Custom Access Client
Embedded Access Client Code
Access SDK API
Configuration info ObAccessClient.xml

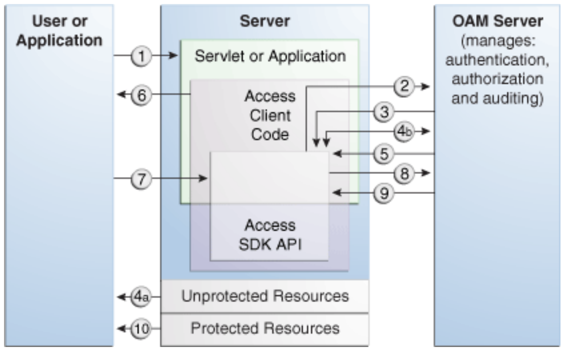
2.6.1.3 Overview of Access Client Request Processing

Regardless of the variability introduced by the types of resources discussed in Section 2.6.1.2, "Access Client Architecture", most Access Clients follow the same basic steps to process user requests.

When a user or application submits a resource request to a servlet or application running on the server where the Access Client is installed, the Access Client code embedded in that servlet or application initiates the basic process shown in the following diagram.

Figure 2–2 illustrates the process of handling a resource request.

Figure 2–2 Process Overview: Handling a Resource Request



Process Overview: Handling a resource request

1. The application or servlet containing the Access Client code receives a user request for a resource.
2. The Access Client constructs an `ResourceRequest` structure, which the Access Client code uses when it asks the OAM Server whether the requested resource is protected.
3. The OAM Server responds.
4. Depending upon the situation, one of the following occurs:
   - If the resource is not protected, the Access Client grants the user access to the resource.
   - If the resource is protected, the Access Client constructs an `AuthenticationScheme` structure, which it uses to ask the OAM Server what credentials the user needs to supply. This step is only necessary if the Access Client supports the use of different authentication schemes for different resources.

5. The OAM Server responds.
6. The application uses a form or some other means to ask for user credentials. In some cases, the user credentials may already have been submitted as part of:
   - A valid session token
   - Input from a web browser
   - Arguments to the command-line script or keyboard input that launched the Access Client application
7. The user responds to the application.
8. The Access Client constructs an `UserSession` structure, which presents the user credentials to the OAM Server, which maps them to a user profile in the Oracle Access Manager user directory.
9. If the credentials prove valid, the Access Client creates a session token for the user, then it sends a request for authorization to the OAM Server. This request contains the user identity, the name of the target resource, and the requested operation.
10. The Access Client grants the user access to the resource, providing that the user is authorized for the requested operation on the particular resource.
11. (Not pictured). A well-behaved Access Client deallocates the memory used by the objects it has created, then shuts down the Access Manager API.

The steps detailed in "Process Overview: Handling a resource request" on page 2-12 represent only the main path of the authorization process. Typically, additional code sections within the servlet or application handle branch situations where:

- The requested resource is not protected.
- The authentication challenge method associated with the protected resource is not supported by the application.
- The user has a valid single sign-on cookie (`ObSSOCookie`), which enables the user to access to the resource without again presenting her credentials for as long as the session token embedded in the cookie remains valid. For details about `ObSSOCookies` and single sign-on, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.
- The user fails to supply valid credentials under the specified conditions.
- Some other error condition arises.
- The developer has built additional custom code into the Access Client to handle special situations or functionality.

2.6.2 Structure of an Access Client

The structure of a typical Access Client application roughly mirrors the sequence of events required to set up an Access Client session.

Access Client Application Structure Sections

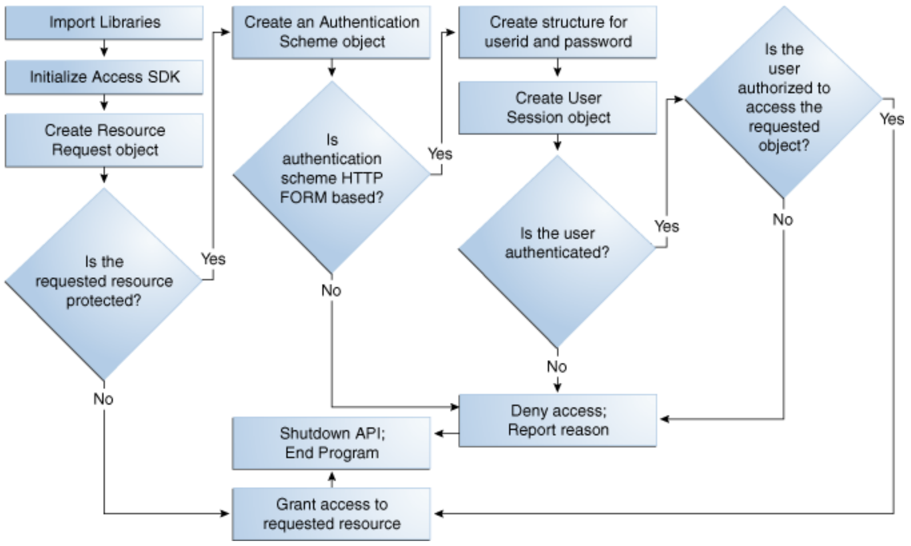
1. Include or import requisite libraries.
2. Get resource.
3. Get authentication scheme.
4. Gather user credentials required by authentication scheme.
5. Create user session.

6. Check user authorization for resource.
7. Clean up (Java uses automatic garbage collection).
8. Shut down.

2.6.2.1 Typical Access Client Execution Flow

All HTTP FORM-based Access Client applications and plug-ins follow the same basic pattern, as illustrated by the following figure. Figure 2–3 shows a process flow for form-based applications:

Figure 2–3 Process Flow for Form-based Applications


Import Libraries
Initialize Access SDK
Create Resource Request object
Is the requested resource protected?
Yes
No
Create an Authentication Scheme object
Is authentication scheme HTTP FORM based?
Yes
No
Create structure for userid and password
Create User Session object
Is the user authenticated?
Yes
No
Is the user authorized to access the requested object?
Yes
No
Deny access; Report reason
Shutdown API; End Program
Grant access to requested resource

Process overview: Access Client Execution for Form-based Applications

1. Import libraries.
2. Initialize the SDK.
3. Create `ResourceRequest` object.
4. Determine if the requested resource is protected.

   Resource Not Protected: Grant access, shut down the API, and end program.
5. **Requested Resource is Protected**: Create an `AuthenticationScheme` object
6. **Authentication Scheme HTTP FORM-based**: Create a structure for user ID and password, create `UserSession` object, determine if the user is authenticated
7. **Authentication Scheme Not HTTP FORM-based**: Deny access and report reason, shut down the API and end program.
8. **User is Authenticated**: Determine if the user is authorized (Step 10).
9. **User is Not Authenticated**: Deny access and report reason, shut down the API and end program.
10. **User is Authorized**: Grant access, shut down the API, and end program.
11. **User Not Authorized**: Deny access and report reason, shut down the API and end program.

Note: To run this test application, or any of the other examples, you must make sure that your Access System is installed and set up correctly. Specifically, check that it has been configured to protect resources that match exactly the URLs and authentication schemes expected by the sample programs. For details on creating application domains and protecting resources with application domains, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.

2.6.2.2 Example of a Simple Access Client: JAccess Client.java

This example is a simple Access Client program. It illustrates how to implement the bare minimum tasks required for a working Access Client, as described here.

Simple Access Client Processing: JAccess Client.java

- Connect to the OAM Server
- Log in using an authentication scheme employing the HTTP FORM challenge method
- Check authorization for a certain resource using an HTTP GET request
- Catch and report Access SDK API exceptions

Typically, this calling sequence is quite similar among Access Clients using the FORM challenge method. FORM-method Access Clients differ principally in the credentials they require for authentication and the type of resources they protect.

A complete listing for `JAccess Client.java` appears in Example 2–1. You can copy this code verbatim into the text file `JAccess Client.java` and execute it on the computer where your Access Manager SDK is installed.

Section 2.6.2.2.1, "Annotated Code: JAccess Client.java" provides annotated code line-by-line to help you become familiar with Java Access Manager API calls.

Example 2–1 JAccess Client.java

```
import java.util.Hashtable;
import oracle.security.am.asdk.*;

public class JAccessClient {
   public static final String ms_resource = "//Example.com:80/secrets/
         index.html";
   public static final String ms_protocol = "http";
   public static final String ms_method = "GET";
   public static final String ms_login = "jsmith";
   public static final String ms_passwd = "j5m1th";
   public String m_configLocation = "/myfolder";
   public static void main(String argv[]) {
  AccessClient ac = null;
       try {
       ac = AccessClient.createDefaultInstance(m_configLocation,
 AccessClient.CompatibilityMode.OAM_10G);

         ResourceRequest rrq = new ResourceRequest(ms_protocol, ms_resource,
               ms_method);
         if (rrq.isProtected()) {
            System.out.println("Resource is protected.");
            AuthenticationScheme authnScheme = new AuthenticationScheme(rrq);
            if (authnScheme.isForm()) {
               System.out.println("Form Authentication Scheme.");
               Hashtable creds = new Hashtable();
               creds.put("userid", ms_login);
               creds.put("password", ms_passwd);
               UserSession session = new UserSession(rrq, creds);
               if (session.getStatus() == UserSession.LOGGEDIN) {
                  if (session.isAuthorized(rrq)) {
                     System.out.println("User is logged in and authorized for the"
                           +"request at level " + session.getLevel());
                  } else {
                     System.out.println("User is logged in but NOT authorized");
                  }
//user can be loggedout by calling logoff method on the session object
               } else {
                  System.out.println("User is NOT logged in");
               }
            } else {
               System.out.println("non-Form Authentication Scheme.");
            }
         } else {
            System.out.println("Resource is NOT protected.");
         }
      }
      catch (AccessException ae) {
         System.out.println("Access Exception: " + ae.getMessage());
      }
      ac.shutdown();
   }
}
```

2.6.2.2.1 Annotated Code: JAccess Client.java Import standard Java library class Hashtable to hold credentials.

```
import java.io.Hashtable;
```

Import the library containing the Java implementation of the Access SDK API classes.:

```
import oracle.security.am.asdk.*;
```

This application is named `JAccessClient`.

```
public class JAccessClient {
```

Since this is the simplest of example applications, we are declaring global constants to represent the parameters associated with a user request for access to a resource.

Typically, a real-world application receives this set of parameters as an array of strings passed from a requesting application, HTTP FORM-based input, or command-line input. For example:

```
public static final String ms_resource = "//Example.com:80/secrets/index.html";
   public static final String ms_protocol = "http";
   public static final String ms_method = "GET";
   public static final String ms_login = "jsmith";
   public static final String ms_passwd = "j5m1th";
```

Launch the main method on the Java interpreter. An array of strings named `argv` is passed to the main method. In this particular case, the user `jsmith`, whose password is `j5m1th`, has requested the HTTP resource

`//Example.com:80/secrets/index.html`. GET is the specific HTTP operation that will be performed against the requested resource. For details about supported HTTP operations and protecting resources with application domains, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.

```
 public static void main(String argv[]) {
```

Place all relevant program statements in the main method within a large try block so that any exceptions are caught by the catch block at the end of the program.

```
AccessClient ac = null;

 try {
```

Initialize the Access SDK by creating AccessClient instance by providing directory location of configuration file ObAccessClient.xml. There are multiple ways to provide configuration location to initialize the Access SDK. For more information refer to *Oracle Access Manager Access SDK Java API Reference*.

You only need to create an instance of AccessClient and it initializes Access SDK API. `AccessClient.CompatibilityMode.OAM_10G` indicates that Access SDK will be initialized to work in a mode which is compatible with both the 10g and 11g releases of Oracle Access Manager.

```
 ac = AccessClient.createDefaultInstance(m_configLocation ,
 AccessClient.CompatibilityMode.OAM_10G);
```

Create a new resource request object named `rrq` using the `ResourceRequest` constructor with the following three parameters:

- **ms_protocol**, which represents the type of resource being requested. When left unspecified, the default value is HTTP. EJB is another possible value, although this particular example does not cover such a case. You can also create custom types, as described in the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.
- **ms_resource**, which is the name of the resource. Since the requested resource type for this particular example is HTTP, it is legal to prepend a host name and port number to the resource name, as in the following:

  ```
  //Example.com:80/secrets/index.html
  ```

- **ms_method**, which is the type of operation to be performed against the resource. When the resource type is HTTP, the possible operations are GET and POST. For EJB-type resources, the operation must be EXECUTE. For custom resource types, you define the permitted operations when you set up the resource type. For more information on defining resource types and protecting resources with application domains, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.

```
ResourceRequest rrq = new ResourceRequest(ms_protocol,
   ms_resource, ms_method);
```

Determine whether the requested resource `rrq` is protected by an authentication scheme.

```
 if (rrq.isProtected()) {
```

If the resource is protected, report that fact.

```
 System.out.println("Resource is protected.");
```

Use the `AuthenticationScheme` constructor to create an authorization scheme object named `authnScheme`. Specify the resource request `rrq` so that `AuthenticationScheme` checks for the specific authorization scheme associated with that particular resource.

```
 AuthenticationScheme authnScheme =new AuthenticationScheme(rrq);
```

Determine if the authorization scheme is FORM-based.

```
 if (authnScheme.isForm()) {
```

If the authorization scheme does use HTTP FORM as the challenge method, report that fact, then create a hashtable named `creds` to hold the name:value pairs representing the user name (`userid`) and the user password (`password`). Read the values for `ms_login` and `ms_passwd` into the hashtable.

```
System.out.println("Form Authentication Scheme.");
Hashtable creds = new Hashtable();
creds.put("userid", ms_login);
creds.put("password", ms_passwd);
```

Using the `UserSession` constructor, create a user session object named session. Specify the resource request as `rrq` and the authentication scheme as `creds` so that `UserSession` can return the new structure with state information as to whether the authentication attempt has succeeded.

```
UserSession session = new UserSession(rrq, creds);
```

Invoke the `getStatus` method on the `UserSession` state information to determine if the user is now successfully logged in (authenticated).

```
if (session.getStatus() == UserSession.LOGGEDIN) {
```

If the user is authenticated, determine if the user is authorized to access the resource specified through the resource request structure `rrq`.

```
if (session.isAuthorized(rrq)) {
  System.out.println(
    "User is logged in " +
    "and authorized for the request " +
```

Determine the authorization level returned by the `getLevel` method for the user session named `session`.

```
 "at level " + session.getLevel());
```

If the user is not authorized for the resource specified in `rrq`, then report that the user is authenticated but not authorized to access the requested resource.

```
} else {
  System.out.println("User is logged in but NOT authorized");
```

If the user is not authenticated, report that fact. (A real world application might give the user additional chances to authenticate).

```
} else {
  System.out.println("User is NOT logged in");
```

If the authentication scheme does not use an HTTP FORM-based challenge method, report that fact. At this point, a real-world application might branch to facilitate whatever other challenge method the authorization scheme specifies, such as `basic` (which requires only `userid` and `password`), `certificate` (SSL or TLS over HTTPS), or `secure` (HTTPS through a redirection URL). For more information about challenge Methods and configuring user authentication, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.

```
} else {
  System.out.println("non-Form Authentication Scheme.");
}
```

If the resource is not protected, report that fact. (By implication, the user gains access to the requested resource, because the Access Client makes no further attempt to protect the resource).

```
} else {
  System.out.println("Resource is NOT protected.");
}
}
```

If an error occurs anywhere within the preceding try block, get the associated text message from object `ae` and report it.

```
catch (AccessException ae) {
  System.out.println(
  "Access Exception: " + ae.getMessage());
}
```

If the application need to logout user, then it can invoke logoff method on the object of `UserSession` class.

Now that the program is finished calling the OAM Server, shut down the API, thus releasing any memory the API might have maintained between calls.

```
  ac.shutdown();
}
}
```

Exit the program. You don't have to deallocate the memory used by the structures created by this application because Java Garbage Collection automatically cleans up unused structures when it determines that they are no longer needed.

2.6.2.3 Example: Java Login Servlet

This example follows the basic pattern of API calls that define an Access Client, as described in Section 2.6.2.2, "Example of a Simple Access Client: JAccess Client.java". However, this example is implemented as a Java servlet running within a Web server, or even an application server. In this environment, the Access Client servlet has an opportunity to play an even more important role for the user of a Web application. By storing a session token in the user's HTTP session, the servlet can facilitate single sign-on for the user. In other words, the authenticated OAM Server session information that the first request establishes is not discarded after one authorization check. Instead, the stored session token is made available to server-side application components such as beans and other servlets, so that they do not need to interrupt the user again and again to request the same credentials. For a detailed discussion of session tokens, `ObSSOCookies`, and configuring single sign-on, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.

This sample login servlet accepts userid/password parameters from a form on a custom login page, and attempts to log the user in to Oracle Access Manager. On successful login, the servlet stores a session token in the `UserSession` object. This enables subsequent requests in the same HTTP session to bypass the authentication step (providing the subsequent requests use the same authentication scheme as the original request), thereby achieving single sign-on.

A complete listing for the Java login servlet is shown in Example 2–2. This code can provide the basis for a plug-in to a web server or application server.

Section 2.6.2.3.1, "Annotated Code: Java Login Servlet" is an annotated version of this code.

Example 2–2 Java Login Servlet Example

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import oracle.security.am.asdk.*;

public class LoginServlet extends HttpServlet {

   public void init(ServletConfig config) throws ServletException {
      try {

       AccessClient ac = AccessClient.createDefaultInstance("/myfolder" ,
 AccessClient.CompatibilityMode.OAM_10G);
      } catch (AccessException ae) {
         ae.printStackTrace();
      }
   }

   public void service(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
      AuthenticationScheme authnScheme = null;
      UserSession user = null;
      ResourceRequest resource = null;
       response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HTML>");
      out.println("<HEAD><TITLE>LoginServlet: Error Page</TITLE></HEAD>");
      out.println("<BODY>");
      HttpSession session = request.getSession( false);
      String requestedPage = request.getParameter("request");
      String reqMethod = request.getMethod();
      Hashtable cred = new Hashtable();
      try {
         if (requestedPage == null || requestedPage.length()==0) {
            out.println("<p>REQUESTED PAGE NOT SPECIFIED\n");
            out.println("</BODY></HTML>");
            return;
         }
         resource = new ResourceRequest("http", requestedPage, "GET");
         if (resource.isProtected()) {
            authnScheme = new AuthenticationScheme(resource);
            if (authnScheme.isBasic()) {
               if (session == null) {
                  String sUserName = request.getParameter("userid");
                  String sPassword = request.getParameter("password");
                  if (sUserName != null) {
                     cred.put("userid", sUserName);
                     cred.put("password", sPassword);
                     user = new UserSession(resource, cred);
                     if (user.getStatus() == UserSession.LOGGEDIN) {
                        if (user.isAuthorized(resource)) {
                           session = request.getSession( true);
                           session.putValue( "user", user);
                           response.sendRedirect( requestedPage );
                        } else {
                           out.println("<p>User " + sUserName + " not" +
                                " authorized for " + requestedPage + "\n");
                        }
                     } else {
                        out.println("<p>User" + sUserName + "NOT LOGGED IN\n");
                     }
                  } else {
                     out.println("<p>USERNAME PARAM REQUIRED\n");
                  }
               } else {
                  user = (UserSession)session.getValue("user");
                  if (user.getStatus() == UserSession.LOGGEDIN) {
                     out.println("<p>User " + user.getUserIdentity() + " already"+
                           "LOGGEDIN\n");
                  }
               }
            } else {
               out.println("<p>Resource Page" + requestedPage + " is not"+
                    " protected with BASIC\n");
            }
         } else {
            out.println("<p>Page " + requestedPage + " is not protected\n");
         }
      } catch (AccessException ex) {
      out.println(ex);
      }
   out.println("</BODY></HTML>");
   }
}
```

2.6.2.3.1 Annotated Code: Java Login Servlet Import standard Java packages to support input and output and basic functionality.

```
import java.io.*;
import java.util.*;
```

Import two packages of Java extensions to provide servlet-related functionality.

```
import javax.servlet.*;
import javax.servlet.http.*;
```

Import the package `oracle.security.am.asdk.jar`, which is the Java implementation of the Access SDK API.

```
import oracle.security.am.asdk.*;
```

This servlet, which builds on the functionality of the generic `HttpServlet` supported by the Java Enterprise Edition, is named `LoginServlet`.

```
public class LoginServlet extends HttpServlet {
```

The `init` method is called once by the servlet engine to initialize the Access Client. In init method, Access SDK can be initialized by instantiating AccessClient by passing the location of the configuration file ObAccessClient.xml file. For more information for creating Access Client refer to *Oracle Access Manager Access SDK Java API Reference*. OAM_10g compatibility flag initialized Access SDK in a mode such that it is compatible with both OAM 10g server and OAM 11g server.

In the case of initialization failure, report that fact, along with the appropriate error message.

```
public void init() {
      AccessClient ac =
 AccessClient.createDefaultInstance("/myfolder" ,
 AccessClient.CompatibilityMode.OAM_10G);
   } catch (AccessException ae) {
         ae.printStackTrace();
   }
}
```

Invoke the `javax.servlet.service` method to process the user's resource request.

```
public void service(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException {
```

Initialize members as `null`. These will store the Access structures used to process the resource request, then set the response type used by this application to `text/html`.

```
AuthenticationScheme authnScheme = null;
UserSession user = null;
ResourceRequest resource = null;
response.setContentType("text/html");
```

Open an output stream titled `LoginServlet: Error Page` and direct it to the user's browser.

```
PrintWriter out = response.getWriter();
out.println("<HTML>");
out.println("<HEAD><TITLE>LoginServlet: Error Page</TITLE></HEAD>");
out.println("<BODY>");
```

Determine if a session already exists for this user. Invoke the `getSession` method with `false` as a parameter, so the value of the existing servlet session (and not the `UserSession`) will be returned if it is present; otherwise, NULL will be returned.

```
 HttpSession session = request.getSession(false);
```

Retrieve the name of the target resource, assign it to the variable `requestedPage`, then retrieve the name of the HTTP method (such as GET, POST, or PUT) with which the request was made and assign it to the variable `reqMethod`.

```
String requestedPage = request.getParameter(Constants.REQUEST);
String reqMethod = request.getMethod();
```

Create a hashtable named `cred` to hold the user's credentials.

```
 Hashtable cred = new Hashtable();
```

If the variable `requestedPage` is returned empty, report that the name of the target resource has not been properly specified, then terminate the servlet.

```
try {
   if (requestedPage == null) {
out.println("<p>REQUESTED PAGE NOT SPECIFIED\n");
out.println("</BODY></HTML>");
return;
```

```
   }
```

If the name of the requested page is returned, create a `ResourceRequest` structure and set the following:

- The resource type is HTTP
- The HTTP method is GET
- `resource` is the value stored by the variable `requestedPage`

```
 resource = new ResourceRequest("http", requestedPage, "GET");
```

If the target resource is protected, create an `AuthenticationScheme` structure for the resource request and name it `authnScheme`.

```
if (resource.isProtected()) {
   authnScheme = new AuthenticationScheme(resource);
```

If the authentication scheme associated with the target resource is HTTP `basic` and no user session currently exists, invoke `javax.servlet.servletrequest.getParameter` to return the user's credentials (user name and password) and assign them to the variables `sUserName` and `sPassword`, respectively.

For the `authnScheme.isBasic` call in the following statement to work properly, the user name and password must be included in the query string of the user's HTTP request, as in the following:

`http://host.example.com/resource?username=bob&userpassword=bobspassword`

where `resource` is the resource being requested, `bob` is the user making the request, and `bobspassword` is the user's password.

Additional Code for authnScheme.isForm

Note: If you substitute `authnScheme.isForm` for `authnScheme.isBasic`, you need to write additional code to implement the following steps.

1. Process the original request and determine that form-based login is required.
2. Send a 302 redirect response for the login form and also save the original resource information in the HTTP session.
3. Authenticate the user by processing the posted form data with the user's name and password.
4. Retrieve the original resource from the HTTP resource and sends a 302 redirect response for the original resource.
5. Process the original request once again, this time using the UserSession stored in the HTTP session.

```
if (authnScheme.isBasic()) {
   if (session == null) {
      String sUserName = request.getParameter(Constants.USERNAME);
      String sPassword = request.getParameter(Constants.PASSWORD);
```

If the user name exists, read it, along with the associated password, into the hashtable named `cred`.

```
if (sUserName != null) {
   cred.put("userid", sUserName);
   cred.put("password", sPassword);
```

Create a user session based on the information in the `ResourceRequest` structure named `resource` and the hashtable `cred`.

```
 user = new UserSession(resource, cred);
```

If the status code for the user returns as LOGGEDIN, that user has authenticated successfully.

```
 if (user.getStatus() == UserSession.LOGGEDIN) {
```

Determine if the user is authorized to access the target resource.

```
 if (user.isAuthorized(resource)) {
```

Create a servlet user session (which is not to be confused with an `UserSession`) and add the name of the user to it.

```
 session = request.getSession( true);
session.putValue( "user", user);
```

Redirect the user's browser to the target page.

```
 response.sendRedirect(requestedPage);
```

If the user is not authorized to access the target resource, report that fact.

```
 } else {
    out.println("<p>User " + sUserName + " not authorized
      for " + requestedPage + "\n");
 }
```

If the user is not properly authenticated, report that fact.

```
} else {
    out.println("<p>User" + sUserName + "NOT LOGGED IN\n");
}
```

If the user name has not been supplied, report that fact.

```
 } else {
out.println("<p>USERNAME PARAM REQUIRED\n");
}
```

If a session already exists, retrieve USER and assign it to the session variable `user`.

```
 } else {
    user = (UserSession)session.getValue("user");
```

If the user is logged in, which is to say, the user has authenticated successfully, report that fact along with the user's name.

```
 if (user.getStatus() == UserSession.LOGGEDIN) {
      out.println("<p>User " + user.getUserIdentity() + " already
      LOGGEDIN\n");
 }
}
```

If the target resource is not protected by a `basic` authentication scheme, report that fact.

```
} else {
   out.println("<p>Resource Page" + requestedPage + " is not protected
         with BASIC\n");
}
```

If the target resource is not protected by any authentication scheme, report that fact.

```
} else {
   out.println("<p>Page " + requestedPage + " is not protected\n");
}
```

If an error occurs, report the backtrace.

```
 } catch (AccessException ex) {
   oe.println(ex);
}
```

Complete the output stream to the user's browser.

```
    out.println("</BODY></HTML>");
  }
}
```

2.6.2.4 Example Using Additional Methods: access_test_java.java

Building on the basic pattern established in the sample application `JAccess Client.java`, discussed in Section 2.6.2.2, "Example of a Simple Access Client: JAccess Client.java", the following sample program invokes several additional OAM Server methods. For instance, it inspects the session object to determine which actions, also named responses, are currently configured in the policy rules associated with the current authentication scheme.

For this demonstration to take place, you must configure some actions through the OAM Server prior to running the application. For details about authentication action and configuring user authentication, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*. The complete listing for this sample application appears in Example 2–3.

An annotated version of the code is provided in Section 2.6.2.4.1, "Annotated Code: access_test_java.java".

Example 2–3 access_test_java.java

```
import java.util.*;
import oracle.security.am.asdk.*;

public class access_test_java {


  public static void main(String[] arg) {
    String userid, password, method, url, configDir, type,
 location;
    ResourceRequest res;
    Hashtable parameters = null;
    Hashtable cred = new Hashtable();
    AccessClient ac = null;
    if (arg.length < 5) {
      System.out.println("Usage: EXPECTED: userid password Type
 HTTP-method"
          +" URL [Installdir [authz-parameters] [location]]]");
      return;
```

```
   } else {
     userid    = arg[0];
     password  = arg[1];
     type      = arg[2];
     method    = arg[3];
     url      = arg[4];
   }
   if (arg.length >= 6) {
     configDir = arg[5];
   } else {
     configDir = null;
   }
   if (arg.length >= 7 && arg[6] != null) {
     parameters = new Hashtable();
     StringTokenizer tok1 = new StringTokenizer(arg[6], "&");
     while (tok1.hasMoreTokens()) {
       String nameValue = tok1.nextToken();
       StringTokenizer tok2 = new StringTokenizer(nameValue,
"=");
       String name = tok2.nextToken();
       String value = tok2.hasMoreTokens() ? tok2.nextToken() :
"";
       parameters.put(name, value);
     }
   }
   location = arg.length >= 8 ? arg[7] : null;
   try {
     ac = AccessClient.createDefaultInstance(configDir ,
AccessClient.CompatibilityMode.OAM_10G);

   } catch (AccessException ae) {
     System.out.println("OAM Server SDK Initialization
failed");
     ae.printStackTrace();
     return;
   }
   cred.put("userid", userid);
   cred.put("password", password);
   try {
     res = new ResourceRequest(type, url, method);
     if (res.isProtected()) {
       System.out.println("Resource " + type + ":" + url + "
protected");
     } else {
       System.out.println("Resource " + type + ":" + url + "
unprotected");
     }
   } catch (Throwable t) {
     t.printStackTrace();
     System.out.println("Failed to created new resource
request");
     return;
   }
   UserSession user = null;
   try {
     user = new UserSession(res, cred);
   } catch (Throwable t) {
     t.printStackTrace();
     System.out.println("Failed to create new user session");
     return;
```

```
    }
    try {
    if (user.getStatus() == UserSession.LOGGEDIN) {
      if (location != null) user.setLocation(location);
      System.out.println("user status is " + user.getStatus());

        if (parameters != null ? user.isAuthorized(res,
parameters) :
              user.isAuthorized(res)) {
          System.out.println("Permission GRANTED");
          System.out.println("User Session Token =" +
              user.getSessionToken());
          if (location != null) {
            System.out.println("Location = " +
user.getLocation());
          }
        } else {
          System.out.println("Permission DENIED");
          if (user.getError() == UserSession.ERR_NEED_MORE_DATA)
{
            int nParams =
res.getNumberOfAuthorizationParameters();
            System.out.print("Required Authorization Parameters
(" +
                nParams + ") :");
            Enumeration e =
res.getAuthorizationParameters().keys();
            while (e.hasMoreElements()) {
              String name = (String) e.nextElement();
              System.out.print(" " + name);
            }
            System.out.println();
          }
        }
   }
   else
   {
   System.out.println("user status is " + user.getStatus());
   }
    } catch (AccessException ae)
    {
    System.out.println("Failed to get user authorization");
    }
   String[] actionTypes = user.getActionTypes();
   for(int i =0; i < actionTypes.length; i++)
   {
   Hashtable actions = user.getActions(actionTypes[i]);
   Enumeration e = actions.keys();
   int item = 0;
   System.out.println("Printing Actions for type " +
actionTypes[i]);
   while(e.hasMoreElements())
   {
   String name = (String)e.nextElement();
   System.out.println("Actions[" + item +"]: Name " + name + "
value " +  actions.get(name));
   item++;
   }
}
   AuthenticationScheme auths;
   try
   {
   auths = new AuthenticationScheme(res);
       if (auths.isBasic())
       {
     System.out.println("Auth scheme is Basic");
       }
       else
       {
     System.out.println("Auth scheme is NOT Basic");
       }
   }
   catch (AccessException ase)
   {
   ase.printStackTrace();
   return;
   }
   try
   {
   ResourceRequest resNew = (ResourceRequest) res.clone();
   System.out.println("Clone resource Name: " +
 resNew.getResource());
   }
   catch (Exception e)
   {
   e.printStackTrace();
   }
   res = null;
   auths = null;
   ac.shutdown();
   }
  }
```

2.6.2.4.1 Annotated Code: access_test_java.java Import standard Java libraries to provide basic utilities, enumeration, and token processing capabilities.

```
import java.util.*;
```

Import the Access SDK API libraries.

```
import oracle.security.am.asdk.*;
```

This class is named `access_test_java`.

```
public class access_test_java {
```

Declare seven variable strings to store the values passed through the array named `arg`.

```
public static void main(String[] arg) {
   String userid, password, method, url, configDir, type, location;
```

Set the current ResourceRequest to `res`.

```
ResourceRequest res;
```

Initialize the hashtable parameters to `null`, just in case they were not already empty.

```
Hashtable parameters = null;
```

Create a new hashtable named `cred`.

```
Hashtable cred = new Hashtable();
```

Initialize AccessClient reference to null.

```
AccessClient ac = null;
```

If the array named `arg` contains less than five strings, report the expected syntax and content for command-line input, which is five mandatory arguments in the specified order, as well as the optional variables `configDir`, `authz-parameters`, and `location`.

```
if (arg.length < 5) {
  System.out.println("Usage: EXPECTED: userid password type
    HTTP-method URL [configDir [authz-parameters] [location]]]");
```

Since fewer than five arguments were received the first time around, break out of the main method, effectively terminating program execution.

```
 return;
} else {
```

If the array named `arg` contains five or more strings, assign the first five arguments (arg[0] through arg[4]) to the variables `userid`, `password`, `type`, `method`, and `url`, respectively.

```
  userid = arg[0];
  password = arg[1];
  type = arg[2];
  method = arg[3];
  url = arg[4];
}
```

If `arg` contains six or more arguments, assign the sixth string in the array to the variable `configDir`.

```
if (arg.length >= 6)
  configDir = arg[5];
```

If `arg` does not contain six or more arguments (in other words, we know it contains exactly five arguments, because we have already determined it does not contain fewer than five) then set configDir to NULL.

```
else
  configDir = null;
```

If `arg` contains at least seven strings, and arg[6] (which has been implicitly assigned to the variable authz-parameters) is not empty, create a new hashtable named `parameters`. The syntax for the string authz-parameters is: p1=v1&p2=v2&...

```
if (arg.length >= 7 && arg[6] != null) {
  parameters = new Hashtable();
```

Create a string tokenizer named `tok1` and parse arg[6], using the ampersand character (&) as the delimiter. This breaks arg[6] into an array of tokens in the form p*n*=v*n*, where n is the sequential number of the token.

```
 StringTokenizer tok1 = new StringTokenizer(arg[6], "&");
```

For all the items in `tok1`, return the next token as the variable `nameValue`. In this manner, `nameValue` is assigned the string pn=vn, where n is the sequential number of the token.

```
while (tok1.hasMoreTokens()) {
  String nameValue = tok1.nextToken();
```

Create a string tokenizer named `tok2` and parse `nameValue` using the equal character (=) as the delimiter. In this manner, pn=vn breaks down into the tokens pn and vn.

```
 StringTokenizer tok2 = new StringTokenizer(nameValue, "=");
```

Assign the first token to the variable `name`.

```
 String name = tok2.nextToken();
```

Assign the second token to `value`. If additional tokens remain in `tok2`, return the next token and assign it to `value`; otherwise, assign an empty string to `value`.

```
 String value = tok2.hasMoreTokens() ? tok2.nextToken() : "";
```

Insert `name` and `value` into the hashtable `parameters`.

```
    parameters.put(name, value);
  }
}
```

If there are eight or more arguments in `arg`, assign arg[7] to the variable `location`; otherwise make `location` empty.

```
location = arg.length >= 8 ? arg[7] : null;
```

Create AccessClient instance using `configDir`, in case if its null provide configuration file location using other options. For more information for creating Access Client, see *Oracle Access Manager Access SDK Java API Reference*.

```
try {
 ac = AccessClient.createDefaultInstance(configDir ,
   AccessClient.CompatibilityMode.OAM_10G);
}
```

If the initialization attempt produces an error, report the appropriate error message (`ae`) to the standard error stream along with the backtrace.

```
catch (AccessException ae) {
  System.out.println("
OAM Server SDK Initialize failed");
  ae.printStackTrace();
```

Break out of the main method, effectively terminating the program.

```
 return;
}
```

Read the variables, user ID, and password into the hashtable named `cred`.

```
 cred.put("userid", userid);
cred.put("password", password);
```

Create a `ResourceRequest` object named `res`, which returns values for the variables type, url and method from the OAM Server.

```
try {
res = new ResourceRequest(type, url, method);
```

Determine whether the requested resource `res` is protected and display the appropriate message.

```
if (res.isProtected())
```

```
  System.out.println("Resource " + type ":" + url + " protected");
else
  System.out.println("Resource " + type + ":" + url + " unprotected");
}
```

If the attempt to create the `ResourceRequest` structure does not succeed, report the failure along with the error message `t`.

```
catch (Throwable t) {
  t.printStackTrace();
  System.out.println("Failed to create new resource request");
```

Break out of the main method, effectively terminating the program.

```
  return;
}
```

Set the `UserSession` parameter `user` to empty.

```
UserSession user = null;
```

Create a `UserSession` structure named `user` so that it returns values for the `ResourceRequest` structure `res` and the `AuthenticationScheme` structure `cred`.

```
try
  user = new UserSession(res, cred);
```

If the attempt to create the `UserSession` structure does not succeed, then report the failure along with the error message `t`.

```
catch (Throwable t) {
  t.printStackTrace();
  System.out.println("Failed to create new user session");
```

Break out of the main method, effectively terminating the program.

```
 return;
}
```

Determine if the user is currently logged in, which is to say, authentication for this user has succeeded.

```
try
{
if (user.getStatus() == UserSession.LOGGEDIN) {
```

If the user is logged in, determine whether the variable `location` is not empty. If `location` is not empty, set the `location` parameter for `AccessClient` to the value of the variable `location`, then report that the user is logged in along with the status code returned by the OAM Server.

```
if (location != null) user.setLocation(location);
System.out.println("user status is " + user.getStatus());
```

Check authorization. To accomplish this, determine whether `parameters` exists. If it does, determine whether the user is authorized with respect to the target resource when the parameters stored in `parameters` are attached. If `parameters` does not exist, simply determine whether the user is authorized for the target resource.

```
try {
  if (parameters != null ? user.isAuthorized(res, parameters) :
    user.isAuthorized(res)) {
```

If the user is authorized to access the resource when all the appropriate parameters have been specified, report that permission has been granted.

```
System.out.println("Permission GRANTED");
```

Display also a serialized representation of the user session token.

```
System.out.println("User Session Token =" + user.getSessionToken());
```

If the variable location is not empty, report the location.

```
if (location != null) {
  System.out.println("Location = " + user.getLocation());
}
```

If the user is not authorized to access the resource, report that permission has been denied.

```
} else {
System.out.println("Permission DENIED");
```

If `UserSession` returns ERR_NEED_MORE_DATA, set the variable `nParams` to the number of parameters required for authorization, then report that number to the user.

```
if (user.getError() == UserSession.ERR_NEED_MORE_DATA) {
  int nParams = res.getNumberOfAuthorizationParameters();
  System.out.print("Required Authorization Parameters (" +
    nParams + ") :");
```

Set `e` to the value of the `keys` parameter in the hashtable returned by the `getAuthorizationParameters` method for the `ResourceRequest` object named "res."

```
 Enumeration e = res.getAuthorizationParameters().keys();
```

Report the names of all the elements contained in `e`.

```
while (e.hasMoreElements()) {
  String name = (String) e.nextElement();
  System.out.print(" " + name);
}
System.out.println();
}
```

Otherwise, simply proceed to the next statement.

```
    else
  }
}
```

If the user is not logged in, report the current user status.

```
else
  System.out.println("user status is " + user.getStatus());
```

In the case of an error, report that the authorization attempt failed.

```
  catch (AccessException ae)
  System.out.println("Failed to get user authorization");
}
```

Now report all the actions currently set for the current user session. Do this by creating an array named `actionTypes` from the strings returned by the `getActionTypes` method. Next, read each string in `actionTypes` into a hashtable named `actions`. Report the name and value of each of the keys contained in `actions`.

```
String[] actionTypes = user.getActionTypes();
for(int i =0; actionTypes[i] != null; i++){
  Hashtable actions = user.getActions(actionTypes[i]);
  Enumeration e = actions.keys();
  int item = 0;
  System.out.println("Printing Actions for type " + actionTypes[i]);
  while(e.hasMoreElements()) {
String name = (String)e.nextElement();
System.out.println("Actions[" + item +"]: Name " + name + " value " +
  actions.get(name));
item++;
  }
}
```

Attempt to create an `AuthenticationScheme` object named `auths` for the `ResourceRequest` object `res`.

```
AuthenticationScheme auths;
try
  auths = new AuthenticationScheme(res);
```

If the `AuthenticationScheme` creation attempt is unsuccessful, report the failure along with the error message `ase`.

```
catch (AccessException ase) {
  ase.printStackTrace();
```

Break out of the main method, effectively terminating the program.

```
  return;
}
```

Determine if the authorization scheme is basic.

```
try
{
if (auths.isBasic())
```

If it is, report the fact.

```
System.out.println("Auth scheme is Basic");
```

It it is not basic, report the fact.

```
else
  System.out.println("Auth scheme is NOT Basic");
```

Use the copy constructor to create a new `ResourceRequest` object named `resNEW` from the original object `res`.

```
  ResourceRequest resNew = (ResourceRequest) res.clone();
```

Report the name of the newly cloned object.

```
 System.out.println("Clone resource Name: " + resNew.getResource());
```

If the `ResourceRequest` object cannot be cloned for any reason, report the failure along with the associated backtrace.

```
}
```

```
catch (Exception e) {
  e.printStackTrace();
}
```

Set the `ResourceRequest` object `res` and the `AuthenticationScheme` object `auths` to NULL, then disconnect the Access SDK API.

```
    res = null;
    auths = null;
    ac.shutdown();
  }
}
```

2.6.2.5 Example of Implementing Certificate-Based Authentication in Java

The following is a code snippet that demonstrates implementing an Access Client in Java that processes an X.509 certificate. This snippet is appropriate when an administrator configures certificate-based authentication in the Access System.

Note that the certificate must be Base 64-encoded. The OAM Server uses this certificate only to identify the user. It does not perform validation such as the validity period, if the root certification is trusted or not, and so on.

```
File oCertFile = new File("sample_cert.pem");
 FileInputStream inStream = new FileInputStream(oCertFile);
 CertificateFactory cf =
 CertificateFactory.getInstance("X.509");

// cert must point to a valid java.security.cert.X509Certificate instance.
X509Certificate cert = (X509Certificate)
cf.generateCertificate(inStream);

// Convert the certificate into a byte array
    byte[] encodecCert = cert.getEncoded();

// Encode the byte array using Base 64-encoding and convert it into a string
String base64EncodedCert = new String(Base64.encodeBase64 (encodedCert));

// Create hashtable to hold credentials
    Hashtable creds = new Hashtable();

// Store the Base 64-encoded under the key "certificate"
    cred.put("certificate", base64EncodedCert);

// Create ResourceResource request object including all information about the //
// resource being accessed
    ResourceRequest resourceRequest = new ResourceRequest(resourceType,
resourceUrl, operation);

// Create a UserSession with the requestRequest and the cred hashtable
    UserSession userSession = new UserSession(resourceRequest, creds);

// The above statement will throw an exception if the certificate cannot be mapped
// to a valid user by the OAM Server.
```

The following import statements are associated with the snippet:

```
import java.security.cert.CertificateFactory;
     import java.security.cert.X509Certificate;
     import java.io.FileInputStream;
```

```
        import oracle.security.am.common.nap.util.Base64;
```

2.7 Building and Deploying an Access Client Program

The following topics are discussed in this section:

- Setting the Development Environment
- Compiling a New Access Client Program
- Configuring and Deploying a New Access Client Program

2.7.1 Setting the Development Environment

The required environment is as follows:

- Install JDK 1.6.0 or higher.
- Install Oracle Access Manager 11g Access SDK.
- Define a JAVA_HOME environment variable to point to JDK installation directory. For example, on UNIX-like operating systems, execute the following command:

  ```
  setenv JAVA_HOME <JDK install dir>/bin
  ```

- Modify the PATH environment variable to the same location where JAVA_HOME/bin points. For example, on UNIX-like operating systems, execute the following command:

  ```
  setenv PATH $JAVA_HOME/bin:$PATH
  ```

- Modify the CLASSPATH environment variable to point to JDK and Access SDK jar files. For example, on UNIX-like operating systems, execute the following command:

  ```
  setenv CLASSPATH $JAVA_HOME/lib/tools.jar:$ACCESSSDK_INSTALL_
  DIR/oamasdk-api.jar:$CLASSPATH
  ```

2.7.2 Compiling a New Access Client Program

After the development environment is configured (see Section 2.7.1, "Setting the Development Environment"), you can compile your Access Client program using a command similar to the following:

```
Javac -cp <location of Access SDK jar> SampleProgram.java
```

Modify details such as class path and Access Client program name as needed.

2.7.3 Configuring and Deploying a New Access Client Program

For information, see Section 2.5, "Configuring and Deploying Access Clients".

2.8 Compatibility: 11g versus 10g Access SDK and APIs

The following topics are discussed in this section:

- Compatibility of the Access SDK
- Compatibility of 10g JNI ASDK and 11g Access SDK
- Deprecated: Oracle Access Manager 10g JNI SDK

The 11g Access Manager API enables developers to write custom Access Client code in Java, which is functionally equivalent to the 10*g* (10.1.4.3) Java Access Client. With Oracle Access Manager 11g, your Java code will interact with underlying Java binaries in the API.

The automatic built-in Java garbage collector deallocates the memory for unused objects when it (the garbage collector) deems appropriate. Garbage collectors do not guarantee when an object will be cleaned up, but do ensure that all objects are destroyed when they are no longer referenced, and no memory leak occurs.

10g and 11g Access Manager API functionality has been organized into seven basic classes. Table 2–5 lists the corresponding class names for the Java language platform.

Table 2–5 Comparison: 11g versus 10g Access API Classes

Purpose of the Class	11g Java Class	10g Java Class
Supports parameter storage structures (lists or hashtables)	From the Java Development Kit: java.util.Hashtable, which extends java.util.Dictionary java.util.Set	java.util.Hashtable, which extends java.util.Dictionary (This is not a Com. Oblix.Access class)
Supports iteration within lists (Java enumerate hashtables)	From the Java Development Kit: java.util.Hashtable, which extends java.util.Dictionary java.util.Set	java.util.Hashtable, which extends java.util.Dictionary (This is not a Com. Oblix.Access class)
Creates and manipulates structures that handle user authentication	AuthenticationScheme class from oracle.security.am.asdk	ObAuthenticationScheme implements ObAuthenticationSchemeInterface
Creates and manipulates structures that handle user requests for resources	ResourceRequest class from oracle.security.am.asdk	ObResourceRequest implements ObResourceRequestInterface
Creates and manipulates structures that handle user sessions, which begin when the user authenticates and end when the user logs off or the session times out	UserSession class from oracle.security.am.asdk	ObUserSession implements ObUserSessionInterface
Retrieves and modifies Access Client configuration information	AccessClient class from oracle.security.am.asdk	ObConfig
Handles errors thrown by the Access Manager API	AccessException, OperationNotPermittedException from oracle.security.am.asdk	ObAccessException

2.8.1 Compatibility of the Access SDK

The Access SDK implements the same functionality that is supported by the 10g JNI ASDK. This functionality is implemented so that you can use it to develop custom access gates that work seamlessly with both the Oracle Access Manager 10g server and the Oracle Access Manager 11g server.

The Access SDK also implements some new and modified functionality that can only be used with an Oracle Access Manager 11g server. Consequently, the Access SDK can gracefully detect whether the application is trying to use this functionality with Oracle Access Manager 10g server.

The new functionalities in Oracle Access Manager 11g Access SDK (`oracle.security.am.asdk`) are as follows:

- Enumerating sessions for the given user
- Terminating the given session
- Setting attributes in the given user session
- Retrieving attributes set in the given session

- Validating user credentials without establishing a session
- Validating user credentials without establishing a session and performing authorization in the same request

---

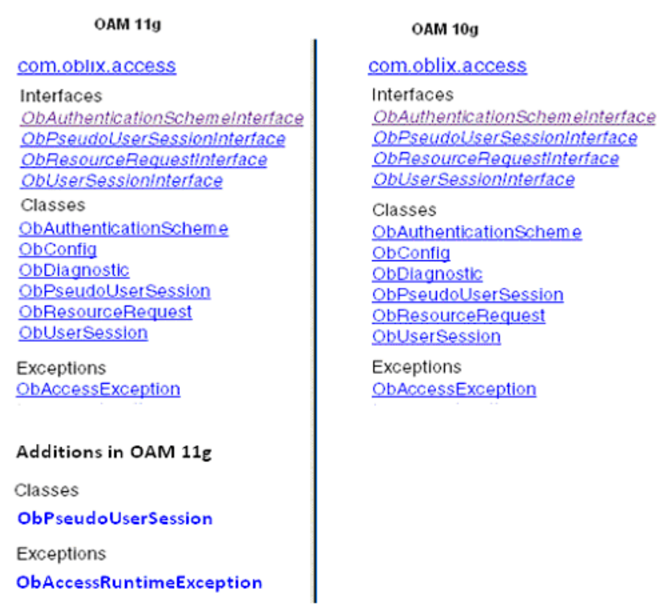
Note: The last two functions are also provided with the `com.oblix.access` package in the Oracle Access Manager 11g Access SDK.

Additionally, the Access SDK provides a modified implementation of the user logout functionality for removing the server side session. This functionality is not supported with Oracle Access Manager 10g server.

2.8.2 Compatibility of 10g JNI ASDK and 11g Access SDK

The following figure depicts the one-to-one mapping between the Oracle Access Manager 10g JNI version and the Oracle Access Manager 11g Access SDK version of the `com.oblix.access` package.

Figure 2–4 Mapping Between Versions of the com.oblix.access Package



OAM 11g

com.oblix.access

Interfaces
ObAuthenticationSchemeInterface
ObPseudoUserSessionInterface
ObResourceRequestInterface
ObUserSessionInterface

Classes
ObAuthenticationScheme
ObConfig
ObDiagnostic
ObPseudoUserSession
ObResourceRequest
ObUserSession

Exceptions
ObAccessException

Additions in OAM 11g

Classes
ObPseudoUserSession

Exceptions
ObAccessRuntimeException

OAM 10g

com.oblix.access

Interfaces
ObAuthenticationSchemeInterface
ObPseudoUserSessionInterface
ObResourceRequestInterface
ObUserSessionInterface

Classes
ObAuthenticationScheme
ObConfig
ObDiagnostic
ObPseudoUserSession
ObResourceRequest
ObUserSession

Exceptions
ObAccessException

Custom access gates developed using 10g JNI ASDK can continue to work with 11g Access SDK without any code changes.

As shown in Figure 2–4, the following classes have been added to the Oracle Access Manager 11g Access SDK `com.oblix.access` package:

- **ObPseudoUserSession**: This class provides the following functionalities, which you can only use with Oracle Access Manager 11g server:
  - Validating user credentials without establishing a session.
  - Validating user credentials without establishing a session and performing authorization in the same request.

- **ObAccessRuntimeException**: This class indicates a runtime error while performing operations that use `ObAuthenticationScheme` and `ObResourceRequest` classes.

2.8.3 Deprecated: Oracle Access Manager 10g JNI SDK

The Access SDK provides support for interfaces in the 10g JNI ASDK com.oblix.access package. However, all APIs in `com.oblix.access` are marked as deprecated. These APIs will not be enhanced or supported in future Oracle Access Manager 11g Access SDK releases.

2.9 Migrating Earlier Applications or Converting Your Code

This section describes the migration processes to follow if you want to use the Access SDK. Migrating to the Access SDK can be necessary for the following reasons:

- Migrate applications to replace the `com.oblix.access` API of Oracle Access Manager 10g JNI ASDK with the corresponding API in Oracle Access Manager 11g Access SDK without changing how those applications use Access SDK.
- Migrate application code to use `oracle.security.am.asdk` API instead of `com.oblix.access`, which is supported in Oracle Access Manager 11g Access SDK for backward compatibility.

This section contains the following topics:

- Modifying Your Development and Runtime Environment
- Migrating Your Application
- Converting Your Code

2.9.1 Modifying Your Development and Runtime Environment

Before migrating an application, ensure that your development environment is configured. Also ensure that the Oracle Access Manager 11g Access SDK is configured correctly. For more information, see Section 2.5, "Configuring and Deploying Access Clients".

2.9.2 Migrating Your Application

You can migrate Access Clients and plug-ins developed with the Oracle Access Manager 10g `com.oblix.access` package to operate with the OAM 11g Server. This section describes how programs written with the Oracle Access Manager 10g JNI ASDK can be used with Oracle Access Manager 11g.

Note: For information about the similarities and differences between the `com.oblix.access` APIs in Oracle Access Manager 10g JNI and in Oracle Access Manager 11g Access SDK, see Section 2.8.2, "Compatibility of 10g JNI ASDK and 11g Access SDK".

Support for the classes and interfaces provided in Oracle Access Manager 10g JNI SDK and in Oracle Access Manager 11g Access SDK is identical.

In general, you are not required to change or recompile any application code when migrating applications to use `com.oblix.access` classes from Oracle Access Manager 11g Access SDK.

A new runtime exception, `ObAccessRuntimeException`, was introduced in the `com.oblix.access` package. Oracle Access Manager throws this exception when performing operations of `AuthenticationScheme` and `ResourceRequest` classes.

Oracle recommends that you perform proper exception handling in the application code. If this is done, the application should be recompiled with the OAM 11g Access SDK jar file.

2.9.2.1 Configuration Specific to Migration

This discussion assumes that Oracle Access Manager 10g ASDK component is installed and configured with the OAM Server. This scenario uses existing Access Client applications developed using Oracle Access Manager 10g JNI ASDK. The following assumptions are made:

- The configuration items listed in Section 2.5.1, "Configuration Requirements" are referenced from the Oracle Access Manager 10g ASDK installation directory (ASDK_INSTALL_DIR).
- ObAccessClient.xml is read from ASDK_INSTALL_DIR/access/oblix/lib.
- password.xml is read from ASDK_INSTALL_DIR/access/oblix/config if the transport security mode is Simple or Cert.

Simple Mode

To configure the Oracle Access Manager 10g ASDK component in Simple mode, see the *Oracle Access Manager Administration Guide* for the 10g release.

Perform the following steps:

1. Import the aaa_cert.pem and aaa_key.pem files into oamclient-keystore.jks.

   The aaa_cert.pem and aaa_key.pem files are located in ASDK_INSTALL_DIR/access/oblix/config/simple.
2. Located the self-signed CA certificate used for issuing Simple mode certificates in ASDK_INSTALL_DIR/access/oblix/tools/openssl/simpleCA.
3. Import the self-signed CA certificate into oamclient-truststore.jks.
4. Import the certificate and key files into the JKS store by following the steps in Section 2.5.3, "SSL Certificate and Key Files".
5. Copy the JKS stores to ASDK_INSTALL_DIR/access/oblix/config/simple.

Cert Mode

To configure the Oracle Access Manager 10g ASDK component in Cert mode, see the *Oracle Access Manager Administration Guide* for the 10g release.

Perform the following steps:

1. Import the aaa_cert.pem and aaa_key.pem files into oamclient-keystore.jks. Import the aaa_chain.pem into oamclient-truststore.jks.

   The aaa_cert.pem, aaa_key.pemand aa_chain.pem files are located in ASDK_INSTALL_DIR/access/oblix/config.
2. Import the certificate and key files into the JKS store by following the steps in Section 2.5.3, "SSL Certificate and Key Files".
3. Copy the JKS stores to ASDK_INSTALL_DIR/access/oblix/config/simple.

Configuration File Location

An Access Client application migrated to use the `com.oblix.access` API can specify the Oracle Access Manager 10g JNI ASDK configuration file locations as follows:

- Either specify the direction location where Oracle Access Manager 10g ASDK is installed while initializing ASDK, or
- Set an environment variable OBACCESS_INSTALL_DIR, which points to the directory location where Oracle Access Manager 10g JNI ASDK is installed.

Oracle Access Manager 11g Access SDK then determines the path of the required files based on the location passed to it.

Environment

To set your environment, follow the instructions in Section 2.7.1, "Setting the Development Environment". The Oracle Access Manager 10g JNI ASDK is named jobaccess.jar. If jobaccess.jar is in your CLASSPATH, it must be removed.

2.9.3 Converting Your Code

This section describes how to use programs written with the Oracle Access Manager 10g JNI ASDK with Oracle Access Manager 11g.

The 11g Access SDK supports the functionality of 10g JNI ASDK APIs in the `com.oblix.access` package. Implementing these functionalities in the 11g Access SDK enables backward compatibility with the 10g JNI ASDK. However, all of the APIs in `com.oblix.access` are deprecated. These APIs will not be enhanced or supported in future 11g Access SDK releases.

The `oracle.security.am.asdk` package contains a new authentication and authorization API. In addition to functionality supplied by the `com.oblix.access` package, the oracle.security.am.asdk package also contains enhancements that take advantage of OAM 11g Server functionality.

2.9.3.1 Understanding Differences Between JNI ASDK and Access SDK

The following table compares the APIs from the Oracle Access Manager 10g JNI SDK `com.oblix.access package` with the APIs from the Oracle Access Manager 11g Access SDK `oracle.security.am.asdk` package. Where applicable, this table also maps the classes between Oracle Access Manager 10g ASDK and Oracle Access Manager 11g Access SDK.

Table 2–6 Differences Between JNI ASDK com.oblix.access Package and Access SDK oracle.security.am.asdk Package

JNI ASDK com.oblix.access Package	Access SDK oracle.security.am.asdk Package
Interface Summary: ▪ ObAuthenticationSchemeInterface ▪ ObResourceRequestInterface ▪ ObUserSessionInterface	**Interface Summary:** None

Table 2–6 (Cont.) Differences Between JNI ASDK com.oblix.access Package and Access SDK oracle.security.am.asdk Package

JNI ASDK com.oblix.access Package	Access SDK oracle.security.am.asdk Package
Class Summary:	**Class Summary:**
▪ ObAuthenticationScheme	▪ AuthenticationScheme
▪ ObConfig	▪ AccessClient
▪ ObDiagnostic	▪ Supported through AccessClient
▪ ObResourceRequest	▪ ResourceRequest
▪ ObUserSession	▪ UserSession
	▪ PseudoUserSession
	▪ BaseUserSession
Exception Summary:	**Exception Summary:**
ObAccessException	▪ AccessException
	▪ OperationNotPermittedException
Enumeration Summary:	**Enumeration Summary:**
None	AccessClient.CompatibilityMode.OAM_10G

Note that the Oracle Access Manager 11g Access SDK contains a new set of APIs that are functionally similar to the Oracle Access Manager 10g JNI SDK APIs, but with new interfaces.

2.9.3.2 Converting Code

You can migrate application code that was implemented using Oracle Access Manager 10g JNI ASDK to achieve the same functionality in Oracle Access Manager 11g Access SDK. This section explains how to modify existing application code to use the new API in Oracle Access Manager 11g Access SDK.

2.9.3.2.1 Initializing and Uninitializing Access SDK

In Oracle Access Manager 10g JNI SDK, the `com.oblix.access.ObConfig` class provides a function to perform ASDK initialization and uninitialization. In Oracle Access Manager 11g Access SDK, the `oracle.security.am.asdk.AccessClient` provides this function.

As with Oracle Access Manager 10g JNI SDK, the Access Client application instance can work with a given configuration.

Depending on the requirement, you can use the AccessClient class in two different ways:

- You can use the `createDefaultInstance` static function to create a single instance of the `AccessClient` class. Only a single default instance of this class is permitted. Invoking this method multiple times within a single instance of the Access Client application causes an exception.

  If you use the `createDefaultInstance` method, you must use the AccessClient class instance obtained using this method when instantiating any of `AuthenticationScheme`, `ResourceRequest`, or `UserSession` classes

- You can use the `createInstance` static function to create a new `AccessClient` class instance initialized with a given configuration. This class is required when it is within the same running instance of an Access Client application, and the application must work with different Oracle Access Manager systems or different configurations. Each `AccessClient` class instance can log its messages to different log files by passing in an appropriate logger name while constructing the Access Client instances.

You must pass `AccessClient.CompatibilityMode.OAM_10G` in compatibility mode when initializing `AccessClient` objects.

If you use the `createInstance` method, you must use the `AccessClient` class instance obtained using this method when instantiating the `AuthenticationScheme`, `ResourceRequest`, or `UserSession` classes.

While the application is shutting down, it should invoke the `AccessClient` class `shutdown` method to perform uninitialization as shown in the following examples:

- **For Oracle Access Manager 10g JNI ASDK**

```
Public static void main (String args[]) {
  try {
   ObConfig.Initialize (); // Configuration is read from the location pointed
by OBACCESS_INSTALL_DIR
                                          // environment variable
```

OR

```
  ObConfig.Initialize (configLocation); //Configuration is read from the
location provided
   ………..
   }catch (ObAccessException e){
   }
ObConfig.shutdown();
}//main ends here
```

- **For Oracle Access Manager 11g Access SDK**

```
import java.io.*;
import java.util.*;
import oracle.security.am.asdk.*; //Import classes from OAM11g Access ASDK
…………..
Public static void main (String args[]) {
  try {
     ac = AccessClient.createDefaultInstance ("",
AccessClient.CompatibilityMode.OAM_10G); // Refer to Oracle Access Manager
Access SDK Java API Reference
```

OR

```
    AccessClient.createInstance("",AccessClient.CompatibilityMode.OAM_10G); //
Refer to Oracle Access Manager Access SDK Java API Reference
   ………..
   }catch (AccessException e){
   }
ac.shutdown();
}//main ends here
```

2.9.3.2.2 Performing Access Operations

As shown in Table 2–6, there is a one-to-one mapping between the classes that are used to perform access operations. The classes in `oracle.security.am.asdk` are `AuthenticationScheme`, `ResourceRequest`, and `UserSession`.

Depending how the `AccessClient` class is instantiated, use the corresponding constructor of these classes.

Similar to Oracle Access Manager 10g JNI ASDK, any error that occurs during initialization or while performing access operations, is reported as an exception. `AccessException` is the exception class used in Oracle Access Manager 11g Access SDK as seen in the following examples:

- **For Oracle Access Manager 10g JNI ASDK**

```
Public static void main (String args[]) {
  try {
   ObConfig.Initialize (); // Configuration is read from the location pointed
by OBACCESS_INSTALL_DIR
                                        // environment variable
   ObResourceRequest rrq = new ObResourceRequest(ms_protocol, ms_resource,ms_
method);
   if (rrq.isProtected()) {
     System.out.println("Resource is protected.");
     ObAuthenticationScheme authnScheme = new ObAuthenticationScheme(rrq);
     if (authnScheme.isForm()) {
       System.out.println("Form Authentication Scheme.");
       Hashtable creds = new Hashtable();
       creds.put("userid", ms_login);
       creds.put("password", ms_passwd);
       ObUserSession session = new ObUserSession(rrq, creds);
       if (session.getStatus() == ObUserSession.LOGGEDIN) {
         if (session.isAuthorized(rrq)) {
           System.out.println("User is logged in and authorized for the
           request at level " + session.getLevel());
         } else {
           System.out.println("User is logged in but NOT authorized");
         }
      } else {
        System.out.println("User is NOT logged in");
     }
   } else {
     System.out.println("non-Form Authentication Scheme.");
  }
} else {
  System.out.println("Resource is NOT protected.");
}
}catch (ObAccessException oe) {
  System.out.println("Access Exception: " + oe.getMessage());
}
ObConfig.shutdown();
}//main ends here
```

- **For Oracle Access Manager 11g Access SDK**

```
import java.io.*;
import java.util.*;
import oracle.security.am.asdk.*; //Import classes from OAM11g Access ASDK

Public static void main (String args[]) {
  AccessClient ac;
  try {
    ac = AccessClient.createDefaultInstance("",
     AccessClient.CompatibilityMode.OAM_10G);

    ResourceRequest rrq = new ResourceRequest(ms_protocol,ms_resource, ms_
method);

    if (rrq.isProtected()) {
      System.out.println("Resource is protected.");
      AuthenticationScheme authnScheme =new AuthenticationScheme(rrq);
      if (authnScheme.isForm()) {
        System.out.println("Form Authentication Scheme.");
        Hashtable creds = new Hashtable();
        creds.put("userid", ms_login);
        creds.put("password", ms_passwd);
        creds.put("ip", ms_ip);
        creds.put("operation", ms_method);
        creds.put("resource", ms_resource);
        creds.put("targethost", ms_targethost);

        UserSession session = new UserSession(rrq, creds);
        if (session.getStatus() == UserSession.LOGGEDIN) {
          if (session.isAuthorized(rrq)) {
            System.out.println("User is logged in " +
                "and authorized for the request " +"at level " +
session.getLevel());
          } else {
            System.out.println("User is logged in but NOT authorized");
          }
        } else {
          System.out.println("User is NOT logged in");
        }
      }
    }catch (AccessException oe) {
      System.out.println("Access Exception: " + oe.getMessage());
    }
    ac.shutdown();
} //main ends here
```

2.10 Best Practices

This section presents a number of ways to avoid problems and to resolve the most common problems that crop up during development.

2.10.1 Avoiding Problems

Here are some suggestions for avoiding problems with the Access Clients you create:

- Make sure that your Access Client attempts to connect to the correct OAM Server.
- Make sure the configuration information on your OAM Server matches the configuration information on your Access Client. You can check the Access Client configuration information on your OAM Server, using the Oracle Access Suite. For details, see "Registering an Access Client" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.
- To ensure clean connect and disconnect from the OAM Server, use the `initialize` and `shutdown` methods in the `AccessClient` class.
- The environment variable, OBACCESS_INSTALL_DIR, *must* be set on your Windows or UNIX-like host computer so that you can compile and link your Access Client. In general, you also want the variable to be set whenever your Access Client is running.

- Use the exception handling features (try, throw, and catch) of the language used to write your custom Access Client code to trap and report problems during development.

2.10.1.1 Thread Safe Code

Your Access Client represents just one thread in your entire, multi threaded application.

To ensure safe operation within such an environment, Oracle recommends that developers observe the following practices:

- Use a thread safe function instead of its single thread counterpart. For instance, use localtime_r instead of localtime.
- Specify the appropriate build environment and compiler flags to support multithreading. For instance, use -D_REENTRANT. Also, use -mt for UNIX-like platforms and /MD for Windows platforms.
- Take care to use in thread-safe fashion shared local variables such as FILE pointers.
- If Access Client is developed using `com.oblix.access` API of Access SDK, the environment variable, OBACCESS_INSTALL_DIR, must be set on your Windows or UNIX-like host computer so that you can compile and link your Access Client. In general, you also want the variable to be set whenever your Access Client is running. If Access Client is developed using `oracle.security.am.asdk` API of Access SDK, make sure that environment is setup correctly. Please refer documentation of AccessClient class in *Oracle Access Manager Access SDK Java API Reference*.

2.10.2 Identifying and Resolving Problems

Here are some things to look at if your Access Client fails to perform:

- Make sure that your OAM Server is running. On Windows systems, you can check this by navigating to Computer Management, then to Services, then to *AccessServer*, where *AccessServer* is the name of the OAM Server to which you want to connect your Access Client.
- Make sure that Access Client performs user logout to ensure that OAM Server-side sessions are deleted. An accumulation of user sessions can prevent successful user authentication.
- Check that the domain policies your code assumes are in place and enabled for your Access System.
- Read the Release Notes that accompanies the Access System product you are working with.
- Check that your Access Client is not being answered by a lower-level Access System policy which overrides the one you think you are testing.
- The Oracle Access Manager 11g Access Tester enables you to check which policy applies to a particular resource. For details about using the Access Tester and protecting resources with application domains, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.

3 Creating Custom Authentication Plug-ins

The OAM Server uses both authentication and authorization controls to limit access to the resources that it protects. Authentication is governed by specific authenticating schemes, which rely on one or more plug-ins that test the credentials provided by a user when he or she tries to access a resource. The plug-ins can be taken from a standard set provided with OAM Server installation, or custom plug-ins created by your own Java developers.

This chapter provides the following sections:

- Section 3.1, "Introduction to Authentication Plug-ins,"
- Section 3.2, "Introduction to Plug-in Interfaces"
- Section 3.3, "Sample Code: Custom Database User Authentication Plug-in"
- Section 3.4, "Developing an Authentication Plug-in"
- Section 3.5, "Adding Custom Plug-ins"
- Section 3.6, "Creating a Custom Authentication Module for Custom Plug-ins"
- Section 3.7, "Creating Authentication Schemes with Custom Authentication Modules"
- Section 3.8, "Configuring Logging for Custom Plug-ins"

3.1 Introduction to Authentication Plug-ins

Oracle Access Manager 11g provides authentication modules for immediate use out-of-the-box, as well as the following:

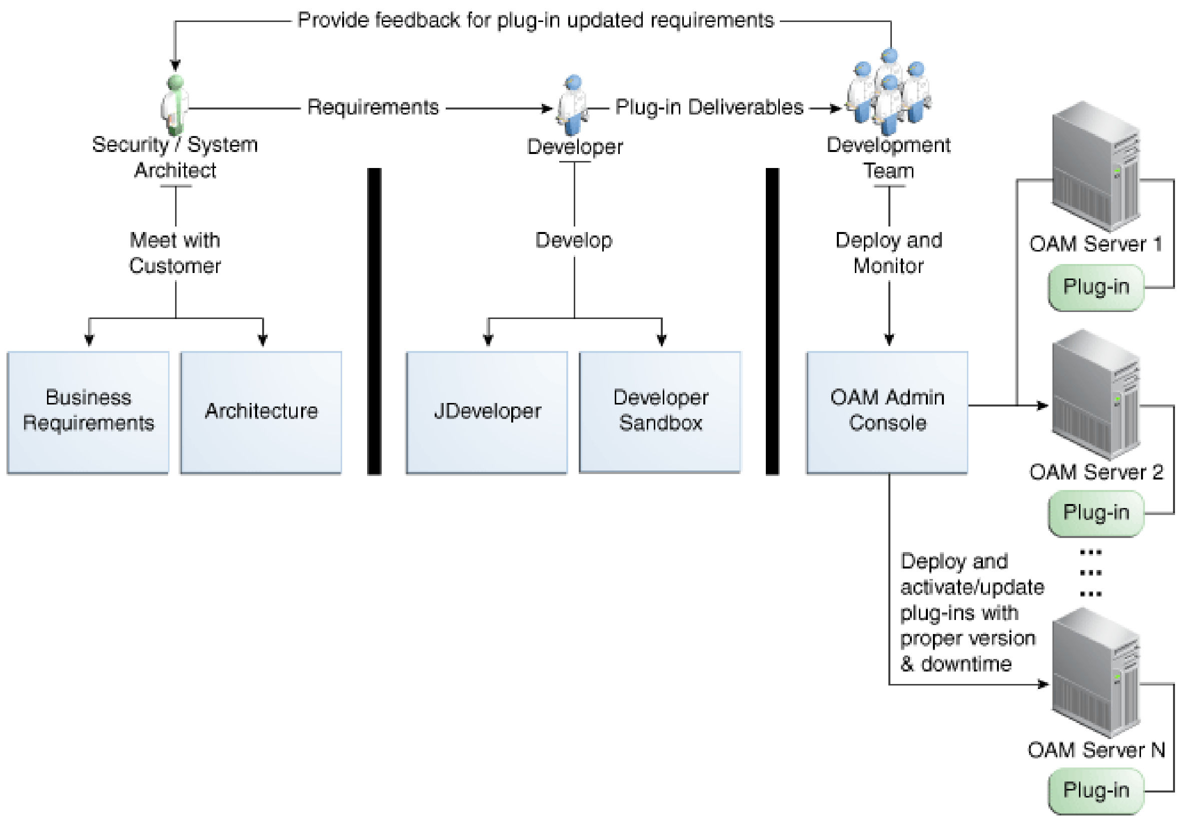
- Provides authentication plug-in interfaces and SDK tooling to build customized authentication modules (plug-ins) to bridge the out-of-the-box features with individual requirements. The new interfaces and SDK tooling:
  - Provide backward compatibility to support custom Oracle Access Manager 10g plug-ins.
  - Include a deterministic method to orchestrate custom plug-ins within an authentication module.
- Provides a mechanism that enables quick deployment of customized authentication plug-ins into Oracle Access Manager 11g
- Maintains the complete plug-in "State" lifecycle of Managed Server and the same to be propagated to AdminServer

The creation of custom plug-ins for credential collection is supported for authentication (steps you can orchestrate).

See Also: About the Plug-in Interfaces on page 3-6

Figure 3–1 provides an overview of the tasks involved in custom plug-in deployment.

Figure 3–1 Custom Plug-in Deployment Workflow



The following overview identifies the tasks involved in custom plug-in deployment.

Task overview: Deploying a custom plug-in requirements

1. **Planning**: Identify the business requirements for this plug-in and consider the authentication flow when a user requests a resource, as described in Section 3.1.2, "About Planning, the Authentication Model, and Plug-ins" on page 3-4.

   The security architect knows how Oracle Access Manager 11g is used and knows the customer's user base. System architects can identify points of improvement in a customer's implementation.

2. **Development**:

   The developer translates what a security architect has designed into the actual plug-in using common libraries to interface custom authentication modules.

   a. Write the plug-in.

   b. Write the metadata XML for the custom module.

   c. Prepare the manifest.

   d. Add the following jar files to the class path: felix.jar, identitystore.jar, oam-plugin.jar, utilities.jar.

3. **Deployment**:

Oracle Access Manager administrators deploy and orchestrate multiple plug-ins to work together in an authentication module and also tests and monitors plug-ins.

a. Adding Custom Plug-ins, which includes configuring the plug-in data source or domain, distributing, and activating the plug-in.

b. Creating a Custom Authentication Module for Custom Plug-ins, which includes adding and orchestrating steps and outcomes OnSuccess, OnFailure, and OnError.

c. Creating Authentication Schemes with Custom Authentication Modules.

d. Configuring Logging for Custom Plug-ins.

e. Test the plug-in using the Oracle Access Manager Access Tester as described in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*

f. Monitor the plug in and provide feedback to the security or system architects to allow for any revisions to the business requirements and architecture.

3.1.1 About the Custom Plug-in Life Cycle

The life cycle of a plug-in centers around the ability to add plug-ins to the OAM Server and use the plug-in to create more features. This allows users to build features and work flows based on the standard (out-of-the-box) plug-ins and user-added plug-ins that act as extension features to the server.

The following list outlines a typical plug-in life cycle:

- Planning
- Plug-in development time, includes generating the plug-in metadata artifact
- Load and lifecycle of the plug-in
  - Import: Upload the plug-in into Oracle Access Manager and use it without restarting servers
  - Distribute: Propagate the plug-in jar from one local AdminServer file system to all manage servers in a cluster, without server downtime
  - Activate: Load the plug-in implementation at run time when this plug-in is used in any Authentication module flow
  - Use the start-up parameters or configuration for the Plug-in
  - "Push" and "pull" plug-in configuration data into oam-config.xml
  - Maintain complete "State" life-cycle of Managed Server and the same to be propagated to AdminServer
- State of the deployed plug-in
- Monitoring and auditing the plug-in
  - Collect the matrix data of time taken to execute a plug-in and the number of times the plug-in is executed
  - Collect the matrix data of plug-in input and output
  - Collect the matrix data of plug-in execution start time and end time
  - Audit the plug-in life-cycle methods code

When a new plug-in JAR is available, the deployer can import it to AdminServer DOMAIN_HOME/oam/plugins from the Oracle Access Suite "Import" action.

Table 3–1 describes the states of a plug-in life cycle that are controlled by Oracle Access Manager administrators. For more information, see Section 3.5, "Adding Custom Plug-ins".

Table 3–1 Plug-in Life Cycle States

State	Description
Import	Adds the plug-in JAR file to the AdminServer DOMAIN_HOME/oam/plugins and begins plug-in validation.
Distribute	Propagates the plug-in to all registered OAM Servers.
Activate	After successful distribution the plug-in can be activated on all registered OAM Servers.
Deactivate	Deactivation checks the plug-in entry flag in oam-config.xml. If any OAM Server fails during the de-activation process, the "De-activation failed" message is propagated.
Remove	Removes the given plug-in (JAR) from DOMAIN_ HOME/config/fmwconfig/oam/plugins directory on AdminServer, which notifies all OAM Servers.

3.1.2 About Planning, the Authentication Model, and Plug-ins

Plug-ins on the OAM Server are part of a custom authentication scheme. Different types of plug-ins can be used for:

- User Identity Mapping

  Plug-ins can add functionality to deal with forms of user input not in the form of a log-in username. Fingerprints, a series of security questions, and other methods can be used. The plug-in translates these inputs and checks them against the database.

- User Authentication

  Responses (not provided out-of-the-box) might be needed when authenticating the user. Custom plug-ins can fulfill this need.

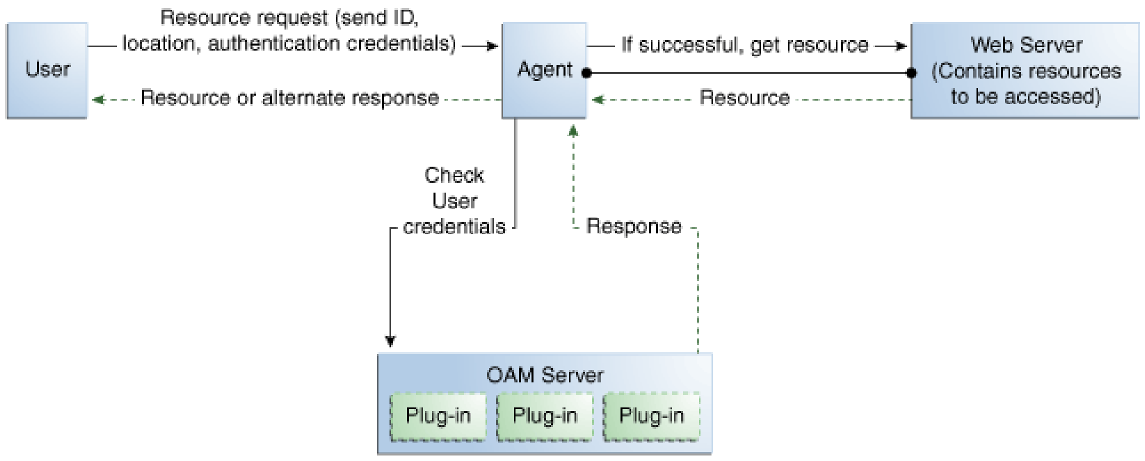
- Custom Responses

  Custom plug-ins can be used for responses and how these responses interact with the rest of the system.

- Other types of plug-ins are also supported

Figure 3–2 illustrates the authentication flow when a user requests a protected resource. Remember that authentication is a process and not a protocol. The green arrows are custom responses generated by plug-ins that are deployed on the OAM Server.

Figure 3–2 Authentication Model and Plug-ins


User
Resource request (send ID, location, authentication credentials)
Resource or alternate response
Agent
If successful, get resource
Resource
Web Server (Contains resources to be accessed)
Check User credentials
Response
OAM Server
Plug-in
Plug-in
Plug-in

Before designing and developing custom authentication plug-ins, Oracle recommends that developers analyze the Oracle Access Manager authentication decision process closely to determine how a user should be authenticated.

When a certain request comes in, there are two possible ways to deal with it. One is to have specific schemes be run depending on the attributes of the request, using a decision engine to run one or multiple schemes to properly authenticate the user. This requires less code within each scheme and allows for more modularity. The other option is to have every scheme be hard-coded to deal with various attributes of requests for specific purposes, not using a decision engine to piece together which schemes need to be run (only one scheme is run).

Example: Decision Engine versus Hard-Coded Authentication

Suppose a user wants to log in to his online bank account using his home computer, at midnight. Following overviews outline the processing differences between the decision engine approach and the hard-coded approach. Developers must decide with what approach best meets their requirements.

Process overview: Decision Engine Approach

The differences between the two approaches are simple but important.

1. The request comes from the user with a certain IP address at midnight.
2. The decision engine determines it has previously dealt with this IP address. It also determines that a user trying to authenticate at midnight is suspicious and requires the user to answer a security question, in addition to a username and password.
3. The security question scheme is run for the specified user, and is successful. This is the first of two authentication schemes selected by the decision engine.
4. The user-password scheme is run, and the user authenticates successfully. This is the second authentication scheme selected by the decision engine.

Process overview: Hard-Coded Approach

1. The request comes from the user with a certain IP address at midnight.
2. The online bank account access scheme is chosen from among other authentication schemes (credit card access scheme, new account creation and verification, and so on).

3. The scheme first checks the IP address to determine if the user has previously made attempts to connect from the computer. It determines the user has.
4. The scheme checks the time. It requires a security question to be answered, which is answered successfully.
5. The scheme requires the user to enter his login credentials, and he authenticates successfully.

Each approach has its own advantages and disadvantages. For the decision-engine model, code re-use is the primary advantage, while the hard-coded approach may result in more security. Developers will have to decide with what approach to go with.

Table 3–2 Approach Comparison

Approach	Description
Decision Engine	Divides authentication schemes into smaller sequential modules that can orchestrated to work together as needed. Advantages: ■ Code re-use is the primary advantage. ■ Mirroring the approach of Oracle Adaptive Access Manager is a secondary advantage.
Hard-coded	Leaves nothing to be decided; resembles a complete set of If-Else statements that the user must pass to authenticate. Advantages: Could result in greater security.

3.2 Introduction to Plug-in Interfaces

This section provides the following topics:

- About the Plug-in Interfaces
- About Plug-in Hierarchies

3.2.1 About the Plug-in Interfaces

This topic introduces the hierarchy for packages, classes, interfaces, and annotations.

Custom plug-in implementation includes writing plug-in implementation class artifacts. The plug-in implementation class must extend the `AbstractAuthenticationPlugIn` class and implement `initialize` and `process` methods. Custom plug-in implementers must implement actual custom authentication processing logic in this method and return the final authentication execution status.

A plug-in's configuration requirements must be given in XML format. This configuration data (metadata) includes plug-in name, author, creation date, version, interface class, implementation class, and configuration data in the form of Attribute / Value pairs.

Oracle Access Manager 11g provides a generic plug-in interface and a more specific authentication interface as described in:

- Section 3.2.1.1, "GenericPluginService"
- Section 3.2.1.2, "AuthnPluginService"

3.2.1.1 GenericPluginService

oracle.security.am.plugin

The public interface, `oracle.security.am.plugin`, is a generic plug-in interface that provides methods to get plug-in name, plug-in implementation class name, plug-in version, plug-in execution status, plug-in monitoring data, plug-in configuration data, start and stop the plain.

AbstractAMPlugin

The public abstract class `oracle.security.am.plugin.AbstractAMPlugin` extends `java.lang.Object implements GenericPluginService, org.osgi.framework.BundleActivator`.

oracle.security.am.plugin.AbstractAMPlugin

This is a Abstract plug-in class that needs to be extended by all Access Management plug-ins. This provides base implementations for plug-ins start and stop methods

See Also: *Oracle Fusion Middleware Oracle Access Manager Java API Reference*

3.2.1.2 AuthnPluginService

oracle.security.am.plugin.authn.AuthnPluginService

The public interface `oracle.security.am.plugin.authn.AuthnPluginService` extends `GenericPluginService`.

This is a authentication plug-in interface that provides an additional authentication specific method to access and process all the data available in the `AuthenticationContext` object and return the process execution status. Plug-in can then set response that will be added to SESSION, request and redirect contexts.

AbstractAuthenticationPlugIn

The public abstract class `oracle.security.am.plugin.authn.AbstractAuthenticationPlugIn` extends `AbstractAMPlugin` implements `AuthnPluginService`.

oracle.security.am.plugin.authn.AbstractAuthenticationPlugIn

This is an authentication Abstract plug-in class that will be exposed to the plug-in developers. All the custom plug-in implementations should extend this `AbstractPlugInService` class. Plug-ins that needs to handle the resource cleanup should override `shutdown(Map < String, Object > OAMEnvironmentContext)` method. This will also provide an instance of `java.util.Logger` to plug-ins.

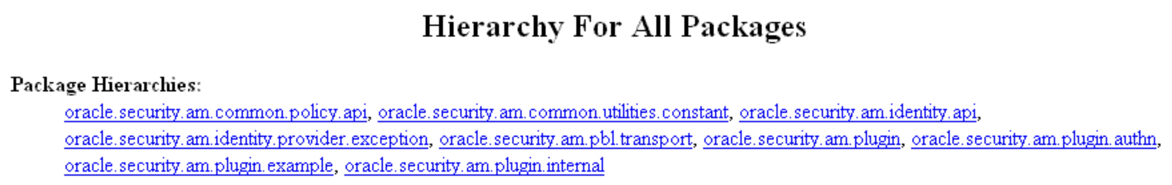
3.2.2 About Plug-in Hierarchies

This topic provides a look at the hierarchies:

- Figure 3–3, "Plug-in Package Hierarchy"
- Figure 3–4, "Plug-in Class Hierarchy"
- Figure 3–5, "Plug-in Interface Hierarchy"
- Figure 3–6, "Plug-in Annotation Type Hierarchy"
- Figure 3–7, "Plug-in Enum Hierarchy"

See Also: *Oracle Fusion Middleware Oracle Access Manager Java API Reference*

Figure 3–3 Plug-in Package Hierarchy

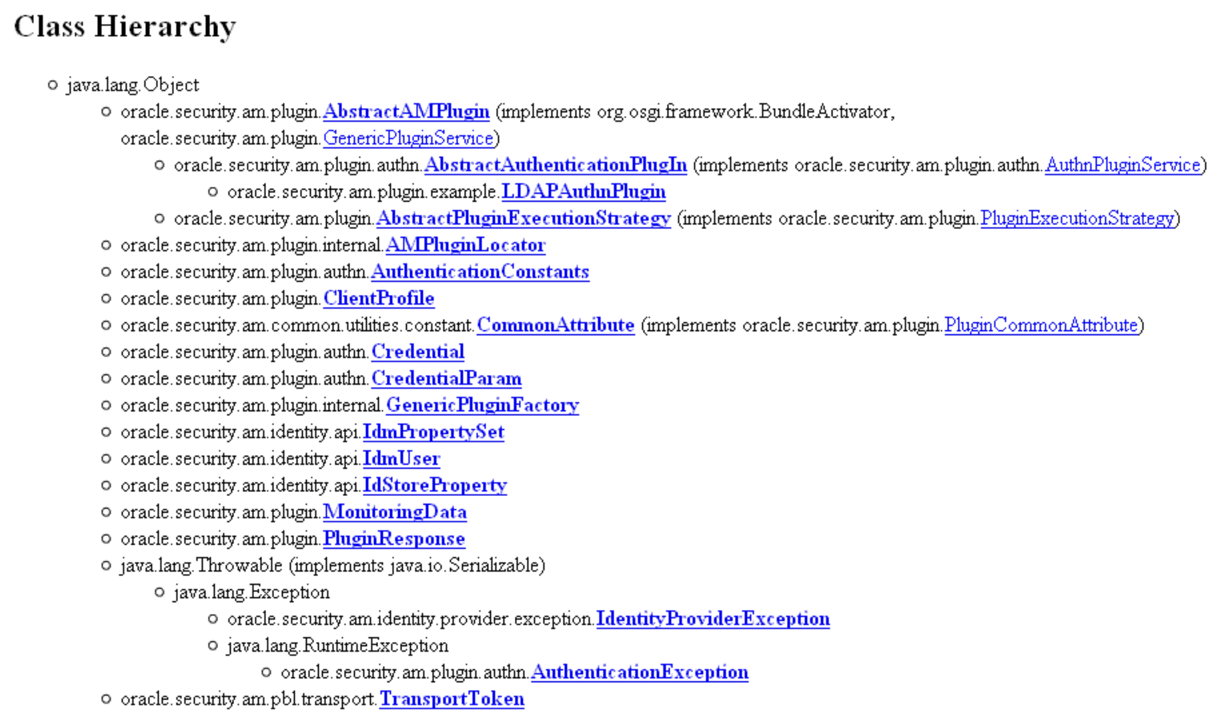


Hierarchy For All Packages

Package Hierarchies:

oracle.security.am.common.policy.api, oracle.security.am.common.utilities.constant, oracle.security.am.identity.api, oracle.security.am.identity.provider.exception, oracle.security.am.pbl.transport, oracle.security.am.plugin, oracle.security.am.plugin.authn, oracle.security.am.plugin.example, oracle.security.am.plugin.internal

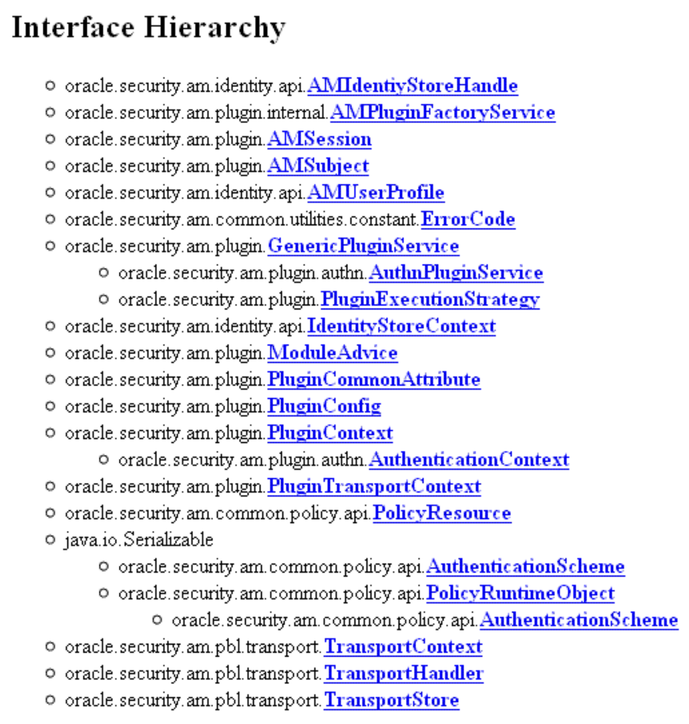
Figure 3–4 Plug-in Class Hierarchy



Class Hierarchy

- java.lang.Object
  - oracle.security.am.plugin.**AbstractAMPlugin** (implements org.osgi.framework.BundleActivator, oracle.security.am.plugin.GenericPluginService)
    - oracle.security.am.plugin.authn.**AbstractAuthenticationPlugIn** (implements oracle.security.am.plugin.authn.AuthnPluginService)
      - oracle.security.am.plugin.example.**LDAPAuthnPlugin**
    - oracle.security.am.plugin.**AbstractPluginExecutionStrategy** (implements oracle.security.am.plugin.PluginExecutionStrategy)
  - oracle.security.am.plugin.internal.**AMPluginLocator**
  - oracle.security.am.plugin.authn.**AuthenticationConstants**
  - oracle.security.am.plugin.**ClientProfile**
  - oracle.security.am.common.utilities.constant.**CommonAttribute** (implements oracle.security.am.plugin.PluginCommonAttribute)
  - oracle.security.am.plugin.authn.**Credential**
  - oracle.security.am.plugin.authn.**CredentialParam**
  - oracle.security.am.plugin.internal.**GenericPluginFactory**
  - oracle.security.am.identity.api.**IdmPropertySet**
  - oracle.security.am.identity.api.**IdmUser**
  - oracle.security.am.identity.api.**IdStoreProperty**
  - oracle.security.am.plugin.**MonitoringData**
  - oracle.security.am.plugin.**PluginResponse**
  - java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Exception
      - oracle.security.am.identity.provider.exception.**IdentityProviderException**
      - java.lang.RuntimeException
        - oracle.security.am.plugin.authn.**AuthenticationException**
  - oracle.security.am.pbl.transport.**TransportToken**

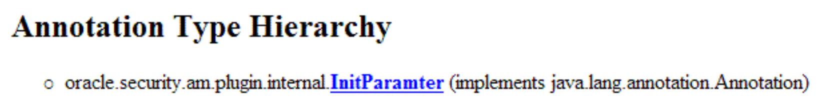
Figure 3–5 Plug-in Interface Hierarchy



Interface Hierarchy

- oracle.security.am.identity.api.AMIdentiyStoreHandle
- oracle.security.am.plugin.internal.AMPluginFactoryService
- oracle.security.am.plugin.AMSession
- oracle.security.am.plugin.AMSubject
- oracle.security.am.identity.api.AMUserProfile
- oracle.security.am.common.utilities.constant.ErrorCode
- oracle.security.am.plugin.GenericPluginService
    - oracle.security.am.plugin.authn.AuthnPluginService
    - oracle.security.am.plugin.PluginExecutionStrategy
- oracle.security.am.identity.api.IdentityStoreContext
- oracle.security.am.plugin.ModuleAdvice
- oracle.security.am.plugin.PluginCommonAttribute
- oracle.security.am.plugin.PluginConfig
- oracle.security.am.plugin.PluginContext
    - oracle.security.am.plugin.authn.AuthenticationContext
- oracle.security.am.plugin.PluginTransportContext
- oracle.security.am.common.policy.api.PolicyResource
- java.io.Serializable
    - oracle.security.am.common.policy.api.AuthenticationScheme
    - oracle.security.am.common.policy.api.PolicyRuntimeObject
        - oracle.security.am.common.policy.api.AuthenticationScheme
- oracle.security.am.pbl.transport.TransportContext
- oracle.security.am.pbl.transport.TransportHandler
- oracle.security.am.pbl.transport.TransportStore

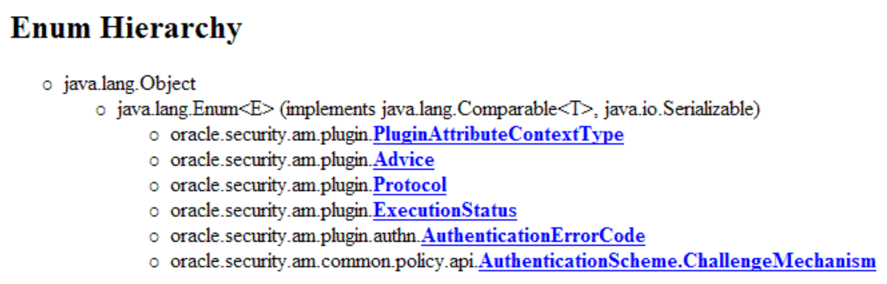
Figure 3–6 Plug-in Annotation Type Hierarchy



Annotation Type Hierarchy

- oracle.security.am.plugin.internal.InitParamter (implements java.lang.annotation.Annotation)

Figure 3–7 Plug-in Enum Hierarchy



Enum Hierarchy

- java.lang.Object
    - java.lang.Enum<E> (implements java.lang.Comparable<T>, java.io.Serializable)
        - oracle.security.am.plugin.PluginAttributeContextType
        - oracle.security.am.plugin.Advice
        - oracle.security.am.plugin.Protocol
        - oracle.security.am.plugin.ExecutionStatus
        - oracle.security.am.plugin.authn.AuthenticationErrorCode
        - oracle.security.am.common.policy.api.AuthenticationScheme.ChallengeMechanism

3.3 Sample Code: Custom Database User Authentication Plug-in

This section provides snapshots of a sample implementation for a database user authentication plug-in to illustrate developer tasks. The following topics are provided:

- Sample Code: Database User Authentication Plug-in

- Sample Plug-in Configuration Metadata Requirements
- Sample Manifest for the Plug-in
- Plug-in JAR File Structure

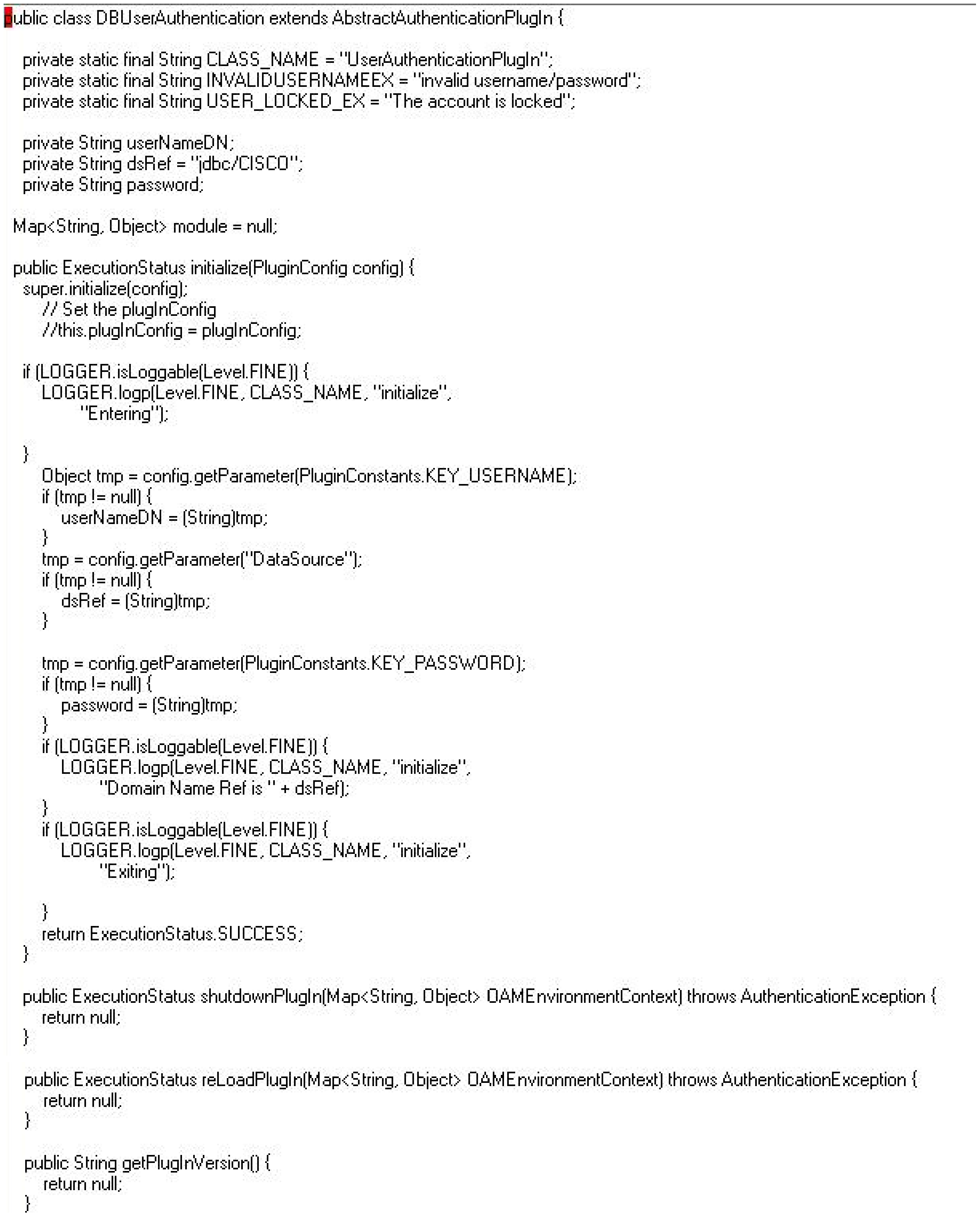
3.3.1 Sample Code: Database User Authentication Plug-in

Following figures illustrate a sample implementation for a Database user authentication plug-in, which is presented in three parts:

- Figure 3–8, "Database User Authentication Plug-in Part 1"
- Figure 3–9, "Database User Authentication Plug-in Part 2"
- Figure 3–10, "Database User Authentication Plug-in Part 3"

See Also: *Oracle Fusion Middleware Oracle Access Manager Java API Reference*

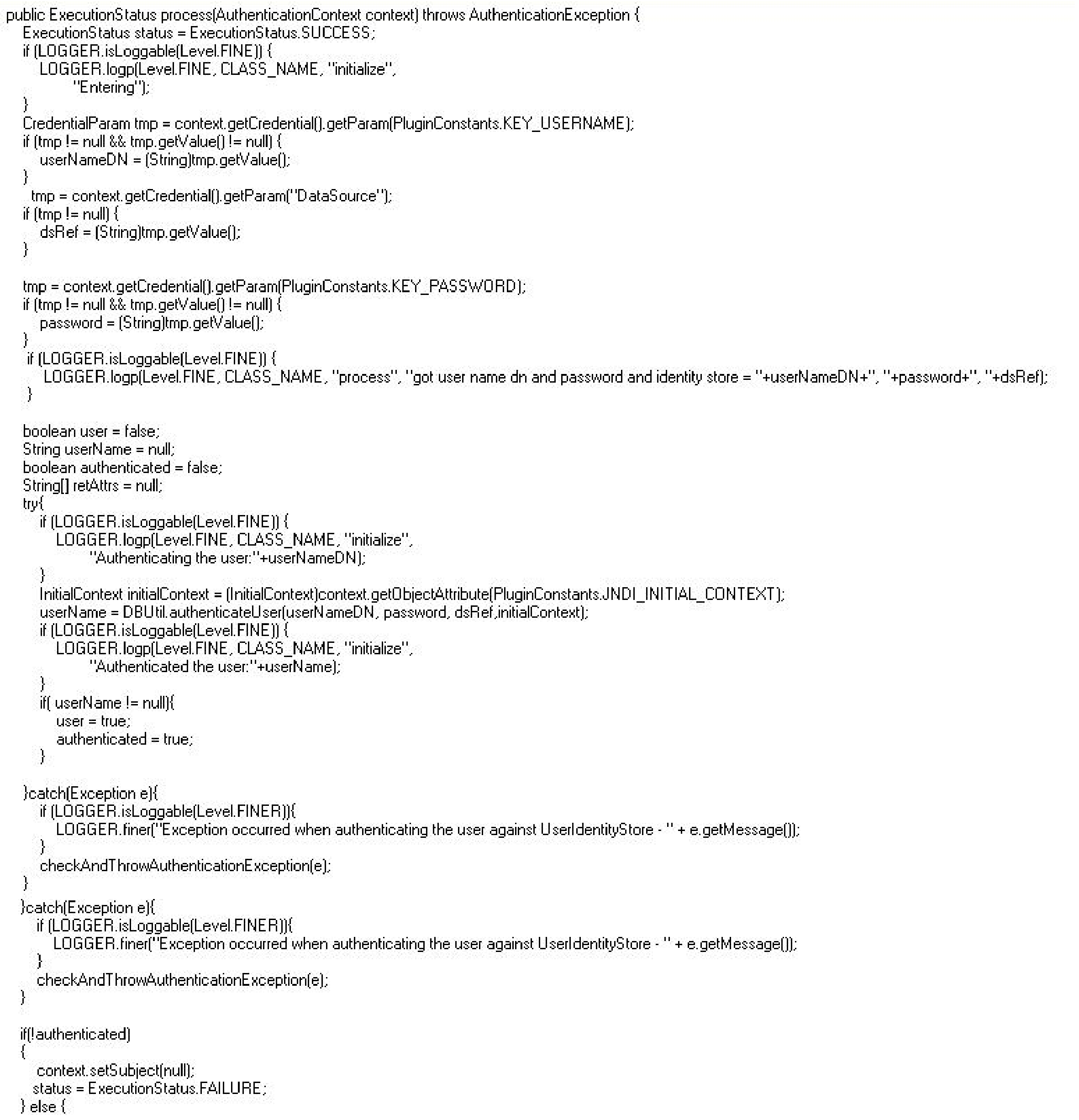
Figure 3–8 Database User Authentication Plug-in Part 1



```
public class DBUserAuthentication extends AbstractAuthenticationPlugIn {

  private static final String CLASS_NAME = "UserAuthenticationPlugIn";
  private static final String INVALIDUSERNAMEEX = "invalid username/password";
  private static final String USER_LOCKED_EX = "The account is locked";

  private String userNameDN;
  private String dsRef = "jdbc/CISCO";
  private String password;

 Map<String, Object> module = null;

 public ExecutionStatus initialize(PluginConfig config) {
  super.initialize(config);
     // Set the plugInConfig
     //this.plugInConfig = plugInConfig;

  if (LOGGER.isLoggable(Level.FINE)) {
     LOGGER.logp(Level.FINE, CLASS_NAME, "initialize",
             "Entering");

  }
     Object tmp = config.getParameter(PluginConstants.KEY_USERNAME);
     if (tmp != null) {
         userNameDN = (String)tmp;
     }
     tmp = config.getParameter("DataSource");
     if (tmp != null) {
         dsRef = (String)tmp;
     }

     tmp = config.getParameter(PluginConstants.KEY_PASSWORD);
     if (tmp != null) {
         password = (String)tmp;
     }
     if (LOGGER.isLoggable(Level.FINE)) {
         LOGGER.logp(Level.FINE, CLASS_NAME, "initialize",
                 "Domain Name Ref is " + dsRef);
     }
     if (LOGGER.isLoggable(Level.FINE)) {
         LOGGER.logp(Level.FINE, CLASS_NAME, "initialize",
                 "Exiting");

     }
     return ExecutionStatus.SUCCESS;
  }

  public ExecutionStatus shutdownPlugIn(Map<String, Object> OAMEnvironmentContext) throws AuthenticationException {
     return null;
  }

  public ExecutionStatus reLoadPlugIn(Map<String, Object> OAMEnvironmentContext) throws AuthenticationException {
     return null;
  }

  public String getPlugInVersion() {
     return null;
  }
```

Continued ..

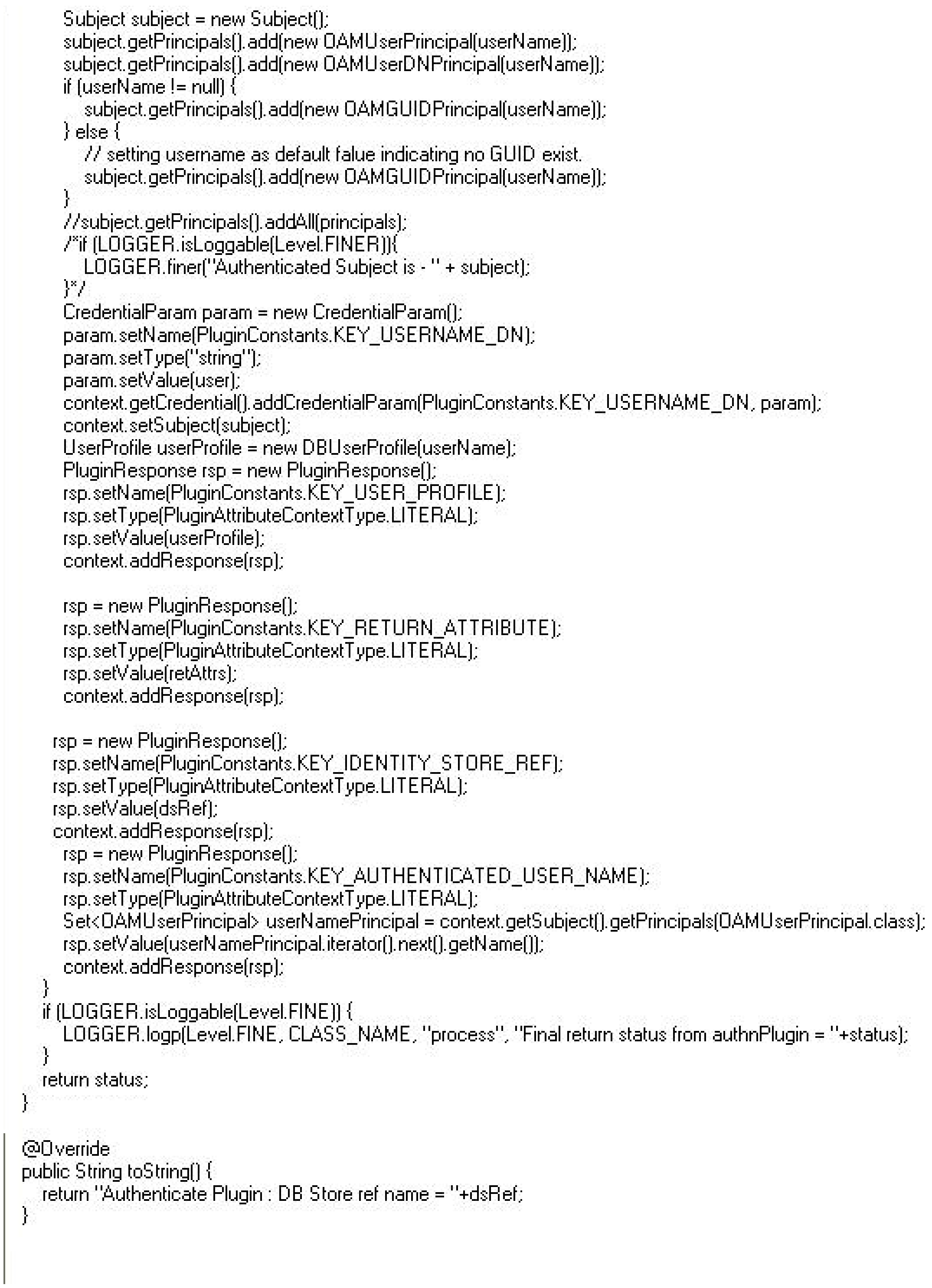
Figure 3–9 Database User Authentication Plug-in Part 2



```
public ExecutionStatus process(AuthenticationContext context) throws AuthenticationException {
    ExecutionStatus status = ExecutionStatus.SUCCESS;
    if (LOGGER.isLoggable(Level.FINE)) {
        LOGGER.logp(Level.FINE, CLASS_NAME, "initialize",
                "Entering");
    }
    CredentialParam tmp = context.getCredential().getParam(PluginConstants.KEY_USERNAME);
    if (tmp != null && tmp.getValue() != null) {
        userNameDN = (String)tmp.getValue();
    }
      tmp = context.getCredential().getParam("DataSource");
    if (tmp != null) {
        dsRef = (String)tmp.getValue();
    }

    tmp = context.getCredential().getParam(PluginConstants.KEY_PASSWORD);
    if (tmp != null && tmp.getValue() != null) {
        password = (String)tmp.getValue();
    }
     if (LOGGER.isLoggable(Level.FINE)) {
         LOGGER.logp(Level.FINE, CLASS_NAME, "process", "got user name dn and password and identity store = "+userNameDN+", "+password+", "+dsRef);
     }

    boolean user = false;
    String userName = null;
    boolean authenticated = false;
    String[] retAttrs = null;
    try{
        if (LOGGER.isLoggable(Level.FINE)) {
            LOGGER.logp(Level.FINE, CLASS_NAME, "initialize",
                    "Authenticating the user:"+userNameDN);
        }
        InitialContext initialContext = (InitialContext)context.getObjectAttribute(PluginConstants.JNDI_INITIAL_CONTEXT);
        userName = DBUtil.authenticateUser(userNameDN, password, dsRef,initialContext);
        if (LOGGER.isLoggable(Level.FINE)) {
            LOGGER.logp(Level.FINE, CLASS_NAME, "initialize",
                    "Authenticated the user:"+userName);
        }
        if( userName != null){
            user = true;
            authenticated = true;
        }

    }catch(Exception e){
        if (LOGGER.isLoggable(Level.FINER)){
            LOGGER.finer("Exception occurred when authenticating the user against UserIdentityStore - " + e.getMessage());
        }
        checkAndThrowAuthenticationException(e);
    }
    }catch(Exception e){
       if (LOGGER.isLoggable(Level.FINER)){
           LOGGER.finer("Exception occurred when authenticating the user against UserIdentityStore - " + e.getMessage());
       }
       checkAndThrowAuthenticationException(e);
    }

    if(!authenticated)
    {
       context.setSubject(null);
      status = ExecutionStatus.FAILURE;
    } else {
```

Continued...

Figure 3–10 Database User Authentication Plug-in Part 3



```
        Subject subject = new Subject();
        subject.getPrincipals().add(new OAMUserPrincipal(userName));
        subject.getPrincipals().add(new OAMUserDNPrincipal(userName));
        if (userName != null) {
            subject.getPrincipals().add(new OAMGUIDPrincipal(userName));
        } else {
            // setting username as default falue indicating no GUID exist.
            subject.getPrincipals().add(new OAMGUIDPrincipal(userName));
        }
        //subject.getPrincipals().addAll(principals);
        /*if (LOGGER.isLoggable(Level.FINER)){
            LOGGER.finer("Authenticated Subject is - " + subject);
        }*/
        CredentialParam param = new CredentialParam();
        param.setName(PluginConstants.KEY_USERNAME_DN);
        param.setType("string");
        param.setValue(user);
        context.getCredential().addCredentialParam(PluginConstants.KEY_USERNAME_DN, param);
        context.setSubject(subject);
        UserProfile userProfile = new DBUserProfile(userName);
        PluginResponse rsp = new PluginResponse();
        rsp.setName(PluginConstants.KEY_USER_PROFILE);
        rsp.setType(PluginAttributeContextType.LITERAL);
        rsp.setValue(userProfile);
        context.addResponse(rsp);

        rsp = new PluginResponse();
        rsp.setName(PluginConstants.KEY_RETURN_ATTRIBUTE);
        rsp.setType(PluginAttributeContextType.LITERAL);
        rsp.setValue(retAttrs);
        context.addResponse(rsp);

      rsp = new PluginResponse();
      rsp.setName(PluginConstants.KEY_IDENTITY_STORE_REF);
      rsp.setType(PluginAttributeContextType.LITERAL);
      rsp.setValue(dsRef);
      context.addResponse(rsp);
        rsp = new PluginResponse();
        rsp.setName(PluginConstants.KEY_AUTHENTICATED_USER_NAME);
        rsp.setType(PluginAttributeContextType.LITERAL);
        Set<OAMUserPrincipal> userNamePrincipal = context.getSubject().getPrincipals(OAMUserPrincipal.class);
        rsp.setValue(userNamePrincipal.iterator().next().getName());
        context.addResponse(rsp);
    }
    if (LOGGER.isLoggable(Level.FINE)) {
        LOGGER.logp(Level.FINE, CLASS_NAME, "process", "Final return status from authnPlugin = "+status);
    }
    return status;
}

@Override
public String toString() {
    return "Authenticate Plugin : DB Store ref name = "+dsRef;
}
```

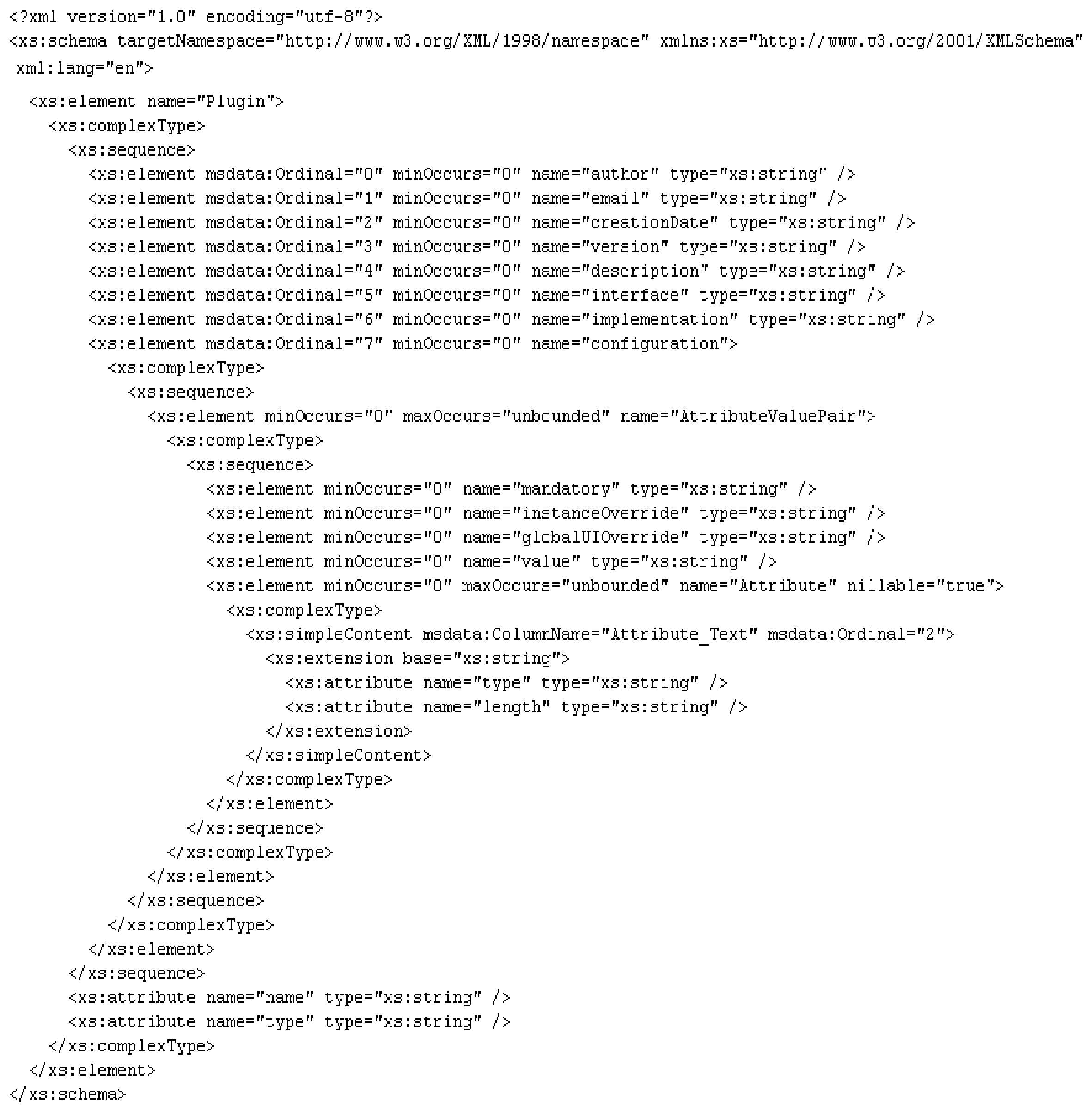
3.3.2 Sample Plug-in Configuration Metadata Requirements

The plug-in's configuration requirements must be given in XML format.

This configuration data (metadata) includes plug-in name, plug-in author, creation date, plug-in version, plug-in interface class, plug-in implementation class, and plug-in configuration data in the form of Attribute / Value pairs.

Figure 3–11 shows the XML Schema Definition (XSD) file containing metadata for the sample: Database User Authentication Plug-in implementation.

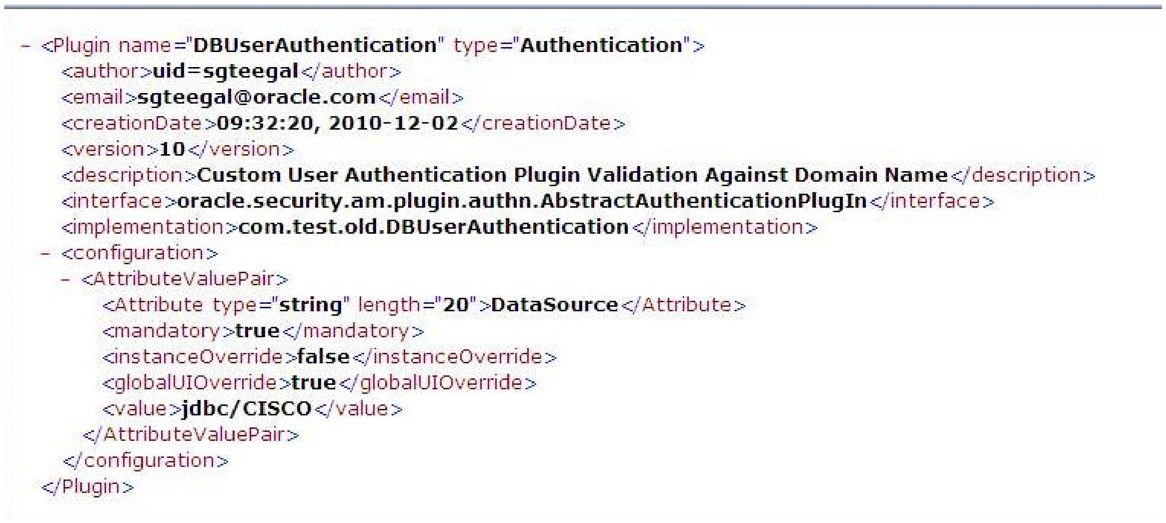
Figure 3–11   XSD Configuration Data: Database User Authentication Plug-in



```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://www.w3.org/XML/1998/namespace" xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xml:lang="en">
  <xs:element name="Plugin">
    <xs:complexType>
      <xs:sequence>
        <xs:element msdata:Ordinal="0" minOccurs="0" name="author" type="xs:string" />
        <xs:element msdata:Ordinal="1" minOccurs="0" name="email" type="xs:string" />
        <xs:element msdata:Ordinal="2" minOccurs="0" name="creationDate" type="xs:string" />
        <xs:element msdata:Ordinal="3" minOccurs="0" name="version" type="xs:string" />
        <xs:element msdata:Ordinal="4" minOccurs="0" name="description" type="xs:string" />
        <xs:element msdata:Ordinal="5" minOccurs="0" name="interface" type="xs:string" />
        <xs:element msdata:Ordinal="6" minOccurs="0" name="implementation" type="xs:string" />
        <xs:element msdata:Ordinal="7" minOccurs="0" name="configuration">
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" maxOccurs="unbounded" name="AttributeValuePair">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element minOccurs="0" name="mandatory" type="xs:string" />
                    <xs:element minOccurs="0" name="instanceOverride" type="xs:string" />
                    <xs:element minOccurs="0" name="globalUIOverride" type="xs:string" />
                    <xs:element minOccurs="0" name="value" type="xs:string" />
                    <xs:element minOccurs="0" maxOccurs="unbounded" name="Attribute" nillable="true">
                      <xs:complexType>
                        <xs:simpleContent msdata:ColumnName="Attribute_Text" msdata:Ordinal="2">
                          <xs:extension base="xs:string">
                            <xs:attribute name="type" type="xs:string" />
                            <xs:attribute name="length" type="xs:string" />
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="type" type="xs:string" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 3–12 shows the XML metadata for the sample: Database User Authentication Plug-in.

Figure 3–12 XML Metadata: Database User Authentication Plug-in

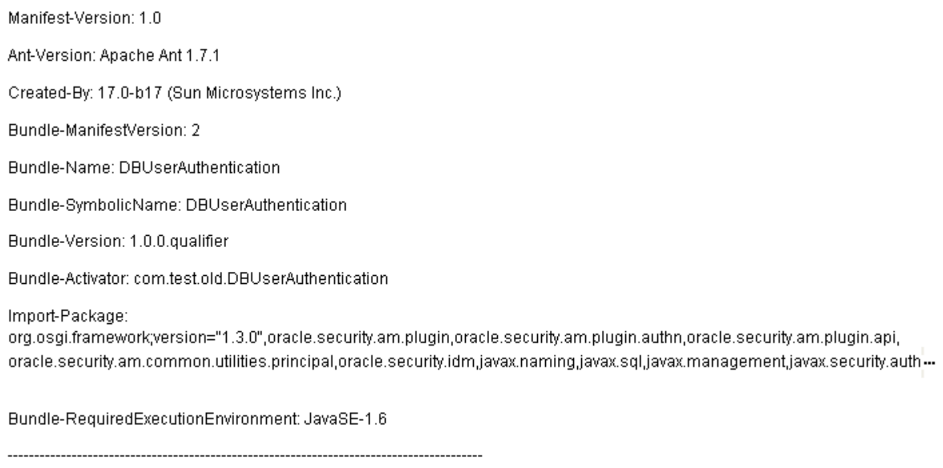


```
- <Plugin name="DBUserAuthentication" type="Authentication">
    <author>uid=sgteegal</author>
    <email>sgteegal@oracle.com</email>
    <creationDate>09:32:20, 2010-12-02</creationDate>
    <version>10</version>
    <description>Custom User Authentication Plugin Validation Against Domain Name</description>
    <interface>oracle.security.am.plugin.authn.AbstractAuthenticationPlugIn</interface>
    <implementation>com.test.old.DBUserAuthentication</implementation>
  - <configuration>
    - <AttributeValuePair>
        <Attribute type="string" length="20">DataSource</Attribute>
        <mandatory>true</mandatory>
        <instanceOverride>false</instanceOverride>
        <globalUIOverride>true</globalUIOverride>
        <value>jdbc/CISCO</value>
      </AttributeValuePair>
    </configuration>
  </Plugin>
```

3.3.3 Sample Manifest for the Plug-in

Figure 3–13 illustrates the MANIFEST.MF file for the sample: Database User Authentication Plug-in.

Figure 3–13 MANIFEST.MF for Sample Database User Authentication Plug-in



```
Manifest-Version: 1.0

Ant-Version: Apache Ant 1.7.1

Created-By: 17.0-b17 (Sun Microsystems Inc.)

Bundle-ManifestVersion: 2

Bundle-Name: DBUserAuthentication

Bundle-SymbolicName: DBUserAuthentication

Bundle-Version: 1.0.0.qualifier

Bundle-Activator: com.test.old.DBUserAuthentication

Import-Package:
org.osgi.framework;version="1.3.0",oracle.security.am.plugin,oracle.security.am.plugin.authn,oracle.security.am.plugin.api,
oracle.security.am.common.utilities.principal,oracle.security.idm,javax.naming,javax.sql,javax.management,javax.security.auth…

Bundle-RequiredExecutionEnvironment: JavaSE-1.6
-----------------------------------------------------------------------------------------
```

3.3.4 Plug-in JAR File Structure

The JAR file structure for the sample (Database User Authentication Plug-in) is listed here:

- <*plugin*>.xml
- <*plugin*>.class (per the package structure, as shown in Section 3.2, "Introduction to Plug-in Interfaces")

- META-INF (MANIFEST.MF)

3.4 Developing an Authentication Plug-in

The developer translates what a security architect has designed into the actual plug-in using common libraries to interface custom authentication modules.

This section guides as you develop an authentication plug-in for use with Oracle Access Manager 11g authentication schemes. The following topics are discussed:

- About Writing a Custom Authentication Plug-in
- Writing a Custom Authentication Plug-in
- JARs Required for Compiling a Custom Authentication Plug-in

3.4.1 About Writing a Custom Authentication Plug-in

Writing the custom plug-in implementation includes writing the plug-in implementation class to:

- Extend `AbstractAuthenticationPlugIn` class (see Section 3.2.1, "About the Plug-in Interfaces")
- Implement `initialize` method
- Implement `process` method

Table 3–3 describes the methods required for the plug-in's functionality.

Table 3–3 Required Plug-in Methods

Required Method	Description
initialize	Gives a handle to the `PluginConfig` object.
	The `PluginConfig` object can be exercised to get plug-in specific system configuration data that is entered when the plug-in is uploaded. This data is required for the plug-in's own functionality
process	Gives a handle to the `AuthenticationContext` object, which can be exercised to get plug-in specific run time configuration data that is:
	■ either updated at plug-in instance level
	■ or updated during plug-in orchestration steps
	The `AuthenticationContext` object extends `PluginContext` object which gives different methods to get the:
	■ plug-in configuration data
	■ exception data
	■ plug-in environment data
	In addition, the `AuthenticationContext` object provides methods to get the:
	■ Authentication scheme
	■ Authenticated Subject
	■ Credential object
	■ Run time policy resource

Note: Custom plug-in developers must implement actual custom authentication processing logic in this method and return the final authentication execution status.

3.4.2 Writing a Custom Authentication Plug-in

This section provides steps to write a custom authentication plug-in.

The following overview describes the actions a developer must take after the system architect identifies the business requirements for this plug-in and considers the authentication flow when a user requests a resource. For more information, see Section 3.1.2, "About Planning, the Authentication Model, and Plug-ins".

Prerequisites

Introduction to Authentication Plug-ins

Sample Code: Custom Database User Authentication Plug-in

Task overview: Developers write a custom authentication plug-in

1. Extend `AbstractAuthenticationPlugIn` class and implement the following methods (see also Section 3.4.1, "About Writing a Custom Authentication Plug-in"):
   - Implement `initialize` method
   - Implement `process` method
2. Develop plug-in code using appropriate Oracle Access Manager 11g interfaces and packages. See:
   - Section 3.1, "Introduction to Authentication Plug-ins"
   - Section 3.3, "Sample Code: Custom Database User Authentication Plug-in"
3. Prepare Metadata for the Custom Plug-in. See:
   - Section 3.3.2, "Sample Plug-in Configuration Metadata Requirements"
4. Prepare the Plug-in Jar file and manifest and turn these over to your deployment team. See:
   - Section 3.3.3, "Sample Manifest for the Plug-in"
   - Section 3.3.4, "Plug-in JAR File Structure"
5. Proceed to:
   - Section 3.4.3, "JARs Required for Compiling a Custom Authentication Plug-in"
   - Section 3.5, "Adding Custom Plug-ins"

3.4.3 JARs Required for Compiling a Custom Authentication Plug-in

Several JAR files are required to compile a custom authentication plug-in. Those jars can be found under:

- extensibility_lifecycle.jar
- . felix.jar
- .felix-service.jar
- oam-plugin.jar

These JAR files are located in the following path:

DOMAIN_HOME/servers/*MANAGED_INSTANCE_NAME*/tmp/_WL_user/oam_server/RANDOM_STRING
/APP-INF/lib

3.5 Adding Custom Plug-ins

This section provides the following topics:

- About Managing Custom Plug-ins
- Adding Custom Plug-ins
- Deleting Custom Authentication Plug-ins

3.5.1 About Managing Custom Plug-ins

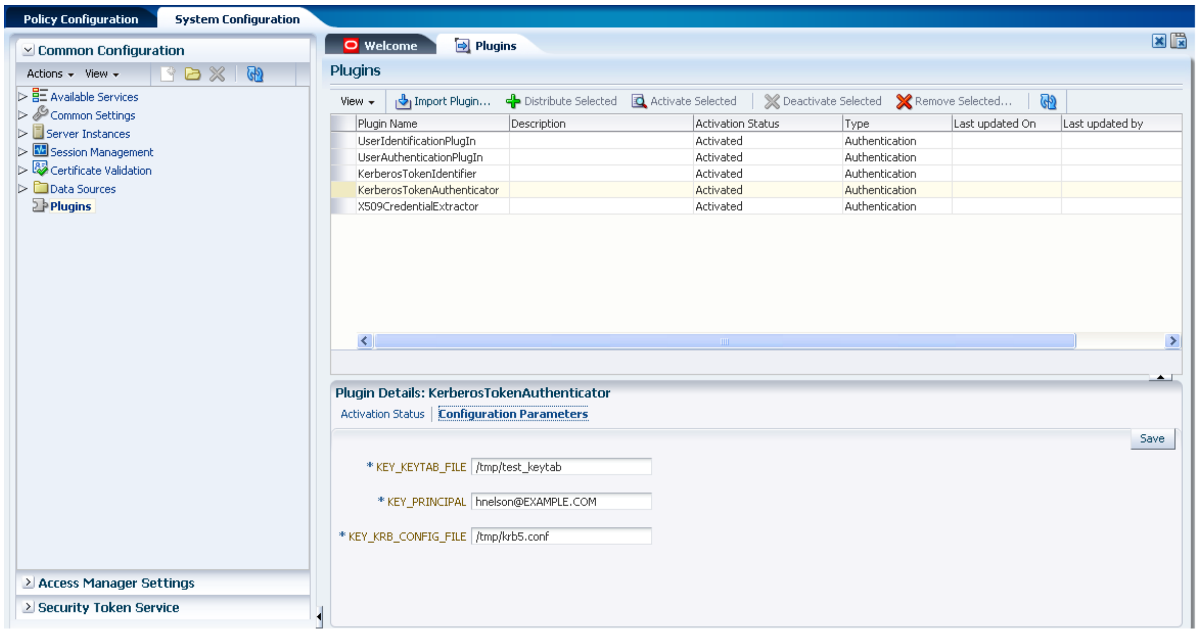
Custom authentication plug-ins can be created and used in custom authentication modules, and, in turn, used in authentication schemes.

After development, the plug-in must be deployed on the admin server, as a JAR file, which is validated automatically. After validation, an administrator can configure and distribute the plug-in using the Oracle Access Suite.

The server processes the XML configuration file within the plug-in JAR file to extract data about the plug-in. After the plug-in is imported, an administrator can see and modify the various plug-in states based on information available from the AdminServer.

Figure 3–14 illustrates the Plug-ins Node under the Common Configuration section of the System Configuration tab, and the Plugins page. This page includes a tool bar with command buttons, most of which operate on the plug-in that is selected in the table. The table provides information about the existing custom plug-ins and their state. The Plugin Details section at the bottom of the page reflects configuration details for the selected plug-in the table.

Figure 3–14   Plug-ins Node Under Common Configuration and the Plugins Page



Plugins

Plugin Name	Description	Activation Status	Type	Last updated On	Last updated by
UserIdentificationPlugIn		Activated	Authentication		
UserAuthenticationPlugIn		Activated	Authentication		
KerberosTokenIdentifier		Activated	Authentication		
KerberosTokenAuthenticator		Activated	Authentication		
X509CredentialExtractor		Activated	Authentication		

Plugin Details: KerberosTokenAuthenticator

* KEY_KEYTAB_FILE	/tmp/test_keytab
* KEY_PRINCIPAL	hnelson@EXAMPLE.COM
* KEY_KRB_CONFIG_FILE	/tmp/krb5.conf

Administrators control plug-in states using the command buttons across the table at the top of the Plugins page, as described in Table 3–4.

Table 3–4 Managing Custom Plug-ins Actions

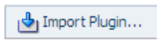
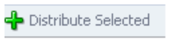
Action	Description
Import Plugin... 	Adds the plug-in JAR file to the AdminServer *DOMAIN_HOME*/oam/plugins and begins plug-in validation. ■ **Same JAR Name**: If the new plug-in JAR name (in *DOMAIN_HOME*/oam/plugins) matches an existing plug-in JAR name (in *DOMAIN_HOME*/config/fmwconfig/oam/plugins), Oracle Access Manager extracts new configuration metadata from the XML file in the JAR (in *DOMAIN_HOME*/oam/plugins) and checks the version of the new plug-in. ■ **XML Version**: If the new plug-in XML version (in *DOMAIN_HOME*/oam/plugins) is greater than the existing XML version (in *DOMAIN_HOME*/config/fmwconfig/oam/plugins), validation is successful. Otherwise, "invalid plugin name with invalid version" is returned and the new plug-in JAR is removed (from *DOMAIN_HOME*/oam/plugins). ■ **Different JAR Name**: If the new plug-in JAR name (in *DOMAIN_HOME*/oam/plugins) is different then existing plug-in JAR names (in *DOMAIN_HOME*/config/fmwconfig/oam/plugins), the new plug-in JAR is uploaded and validation is successful. **On Success**: Status is reported as "Uploaded" (even if an OAM Server is down). If all registered OAM Servers report "Uploaded", then the status on AdminServer is also "Uploaded". **On Failure**: Status is reported as "Upload Failed" See Also: "About the Custom Plug-in Life Cycle" in the *Oracle Fusion Middleware Developer's Guide for Oracle Access Manager and Oracle Security Token Service*
Distribute Selected ... 	■ Propagates the plug-in to all registered OAM Servers. ■ Sets the plug-in flag in oam-config.xml to "Distribute=true". ■ Starts the distribution listener and notification mechanism between AdminServer and OAM Servers ■ Distributes the plug-in JAR from AdminServer node to each OAM Server node under *DOMAIN_HOME*/config/fmwconfig/oam/plugins **On Success**: Status is reported as "Distributed" (even if an OAM Server is down). If all registered OAM Servers report "Distributed", then the status on AdminServer is also "Distributed". **On Failure**: Status is reported as "Distribution Failed"

Table 3–4 (Cont.) Managing Custom Plug-ins Actions

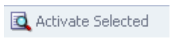
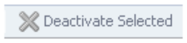
Action	Description
Activate Selected ... 	After successful distribution the plug-in can be activated on all registered OAM Servers. Activation: ■ Updates the plug-in flag in oam-config.xml to "Activate=true" ■ Starts the Message listener and notification mechanism between AdminServer and OAM Servers ■ AdminServer sends message "Activate" to all registered OAM Servers **On Success**: Status is reported as "Activated" (even if an OAM Server is down). If all registered OAM Servers report "Activated", then the status on AdminServer is also "Activated". **On Failure**: Status is reported as "Activation Failed" Following activation on all OAM Servers, the plug-in can be used and executed in any authentication module construction or orchestration.
Deactivate Selected ... 	Following plug-in activation, an administrator can choose to deactivate the plug-in: if the plug-in is not used in any authentication module or scheme, for example. The selected plug-in from all registered OAM Servers. Deactivate: ■ Updates the plug-in flag in oam-config.xml to "De-activate=true" ■ Starts the Distribution listener and notification mechanism between AdminServer and OAM Servers ■ Removes the plug-in JAR from AdminServer and each registered OAM Server (*DOMAIN_HOME*/config/fmwconfig/oam/plugins) ■ AdminServer sends message "De-activation" to all registered OAM Servers ■ OAM Servers send status message to AdminServer using the "Message" listeners on both AdminServer and OAM Server **On Success**: Status is reported as "De-activation" (even if an OAM Server is down). If all registered OAM Servers report "De-activation", then the status on AdminServer is also "De-activation". Plug-in configuration is removed from oam-config.xml. **Note**: After deactivation, the plug-in cannot be used or executed in any authentication module or orchestration. **On Failure**: Status is reported as "De-activation Failed"
Remove Selected ... 	Following plug-in deactivation, an administrator can delete the selected plug-in. During this process, Oracle Access Manager: Delete: ■ Updates the plug-in flag in oam-config.xml to "Remove=true" ■ Starts the Distribution listener and notification mechanism between AdminServer and OAM Servers ■ Removes the plug-in JAR from AdminServer and each registered OAM Server (*DOMAIN_HOME*/config/fmwconfig/oam/plugins) ■ AdminServer sends message "Activate" to all registered OAM Servers On Success: Status is reported as "Removed" (even if an OAM Server is down). If all registered OAM Servers report "Removed", then the status on AdminServer is also "Removed". Plug-in configuration is removed from oam-config.xml. **On Failure**: Status is reported as "Removal Failed"

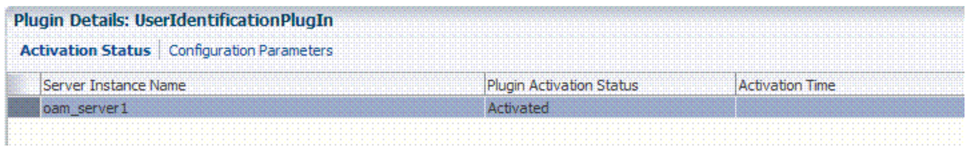
Table 3–4 describes elements in the Plugins status table.

Table 3–5 Elements in the Plugins Status Table

Element	**Description**
Plugin Name	Extracted from the Plugin name element of the XML metadata file.
Description	Extracted from the description element of the XML metadata file.
Activation Status	Reported activation status based on information from AdminServer.
Type	Extracted from the type element of the XML metadata file.
Last Updated on	Extracted from the creation date element of the XML metadata file.
Last Updated by	Extracted from the author element of the XML metadata file.

In the Plugin Details section of the page, the Activation Status is maintained by the AdminServer, as shown in Table 3–6.

Figure 3–15 Activation Status of the Selected Plug-in



Plugin Details: UserIdentificationPlugIn

Server Instance Name	Plugin Activation Status	Activation Time
oam_server1	Activated	

Depending on your plug-in, various configuration details are extracted from the configuration element of the XML metadata file to populate Configuration Parameters in the Plugin Details section. Examples are shown in Table 3–6.

Table 3–6 Example of Plugin Details Extracted from XML Metadata File

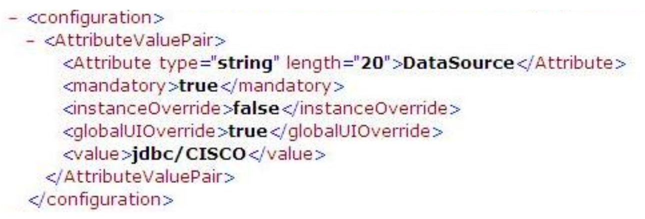
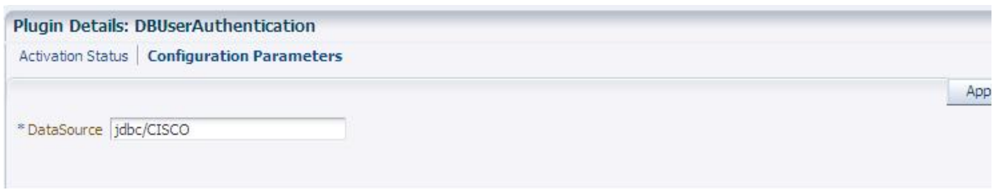
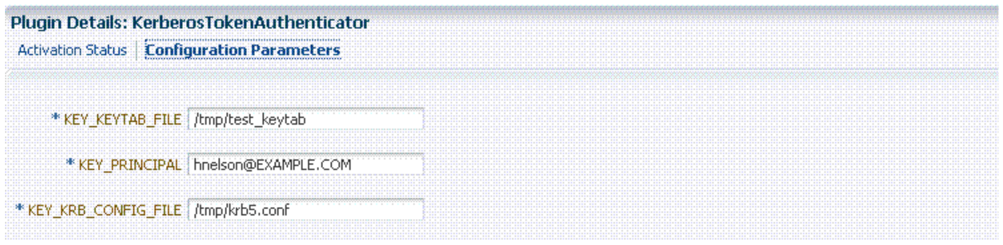
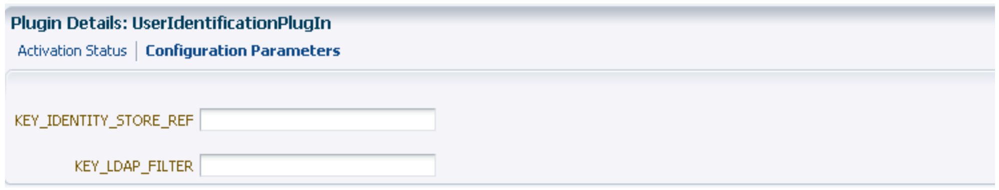
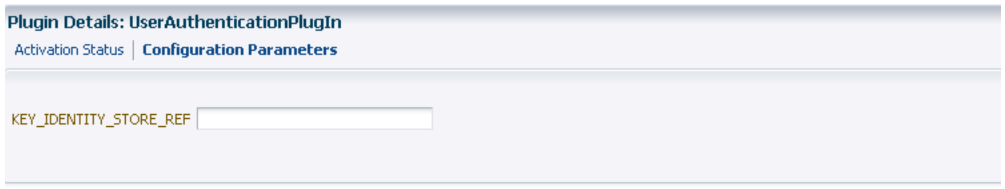
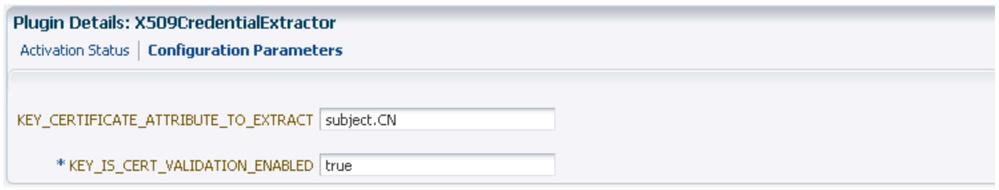
Configuration Element	Description
DataSource	 - <configuration> - <AttributeValuePair> <Attribute type="**string**" length="**20**">**DataSource**</Attribute> <mandatory>**true**</mandatory> <instanceOverride>**false**</instanceOverride> <globalUIOverride>**true**</globalUIOverride> <value>**jdbc/CISCO**</value> </AttributeValuePair> </configuration>  **Plugin Details: DBUserAuthentication** Activation Status | **Configuration Parameters** App * DataSource jdbc/CISCO
Kerberos Details	Defines Kerberos details for his plug-in to use.  **Plugin Details: KerberosTokenAuthenticator** Activation Status | **Configuration Parameters** * KEY_KEYTAB_FILE /tmp/test_keytab * KEY_PRINCIPAL hnelson@EXAMPLE.COM * KEY_KRB_CONFIG_FILE /tmp/krb5.conf

Table 3–6 (Cont.) Example of Plugin Details Extracted from XML Metadata File

Configuration Element	Description
User Identification Details	Defines the User Identity Store and filter details for this plug-in to use.  Plugin Details: UserIdentificationPlugIn; Activation Status; Configuration Parameters; KEY_IDENTITY_STORE_REF; KEY_LDAP_FILTER
User Authentication Details	Defines the User Identity Store for this plug-in to use.  Plugin Details: UserAuthenticationPlugIn; Activation Status; Configuration Parameters; KEY_IDENTITY_STORE_REF
X.509 Details	Defines the certificate details for this plug-in to use.  Plugin Details: X509CredentialExtractor; Activation Status; Configuration Parameters; KEY_CERTIFICATE_ATTRIBUTE_TO_EXTRACT: subject.CN; * KEY_IS_CERT_VALIDATION_ENABLED: true

3.5.2 Adding Custom Plug-ins

Users with valid administrator credentials can perform the following task to add, validate, distribute, and activate a custom plug-in.

Prerequisites

Developing an Authentication Plug-in

To add the custom authentication plug-in

1. **Import the Plug-in**:

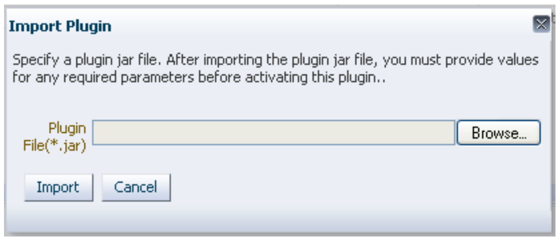
   a. Go to the Oracle Access Suite and log in, as usual. For example:

      https://*hostname*:*port*/oamconsole/

   b. From the System Configuration tab, Common Configuration section, click Plugins and then click Open from the Actions menu.

   c. Click the Import Plugin button.

   d. In the Import Plugin dialog box, click Browse and select the name of your plug-in JAR file.



Import Plugin

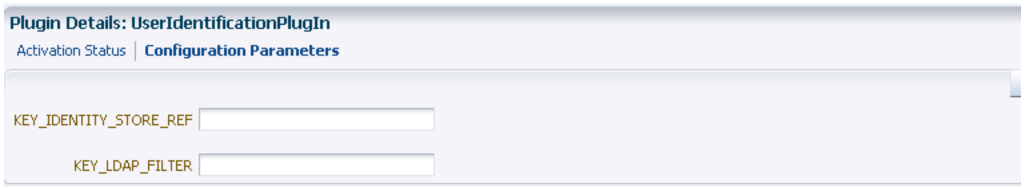
Specify a plugin jar file. After importing the plugin jar file, you must provide values for any required parameters before activating this plugin..

Plugin File(*.jar)

e. Review the message in the dialog box, then click Import.

The JAR file is validated as described in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.

2. **Configure Parameters**: Expand the Plugin Details section, click Configuration Parameters, and enter appropriate information as needed. For example:



Plugin Details: UserIdentificationPlugIn

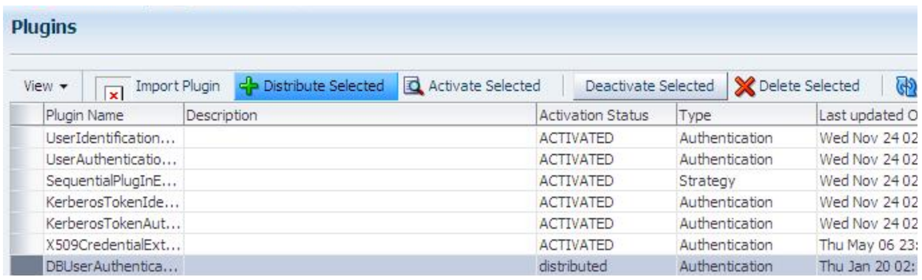
KEY_IDENTITY_STORE_REF

KEY_LDAP_FILTER

3. **Distribute the Plug-in to OAM Servers**:

a. In the Plugins table, click your plug-in name to select it.

b. Click the Distribute Selected button, then check its Activation Status.



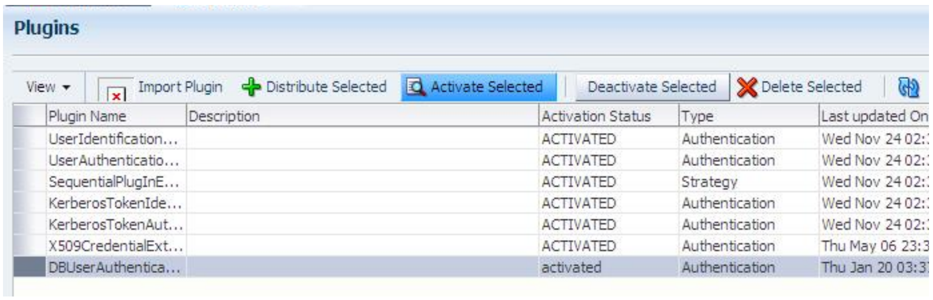
Plugins

Plugin Name	Description	Activation Status	Type	Last updated O
UserIdentification...		ACTIVATED	Authentication	Wed Nov 24 02
UserAuthenticatio...		ACTIVATED	Authentication	Wed Nov 24 02
SequentialPlugInE...		ACTIVATED	Strategy	Wed Nov 24 02
KerberosTokenIde...		ACTIVATED	Authentication	Wed Nov 24 02
KerberosTokenAut...		ACTIVATED	Authentication	Wed Nov 24 02
X509CredentialExt...		ACTIVATED	Authentication	Thu May 06 23:
DBUserAuthentica...		distributed	Authentication	Thu Jan 20 02:

4. **Activate the Plug-in** (and the custom plugin implementation class) so it is ready to be used by OAM Server:

a. In the Plugins table, click your plug-in name to select it.

b. Click the Activate Selected button, then check its Activation Status.



Plugins

Plugin Name	Description	Activation Status	Type	Last updated On
UserIdentification...		ACTIVATED	Authentication	Wed Nov 24 02::
UserAuthenticatio...		ACTIVATED	Authentication	Wed Nov 24 02::
SequentialPlugInE...		ACTIVATED	Strategy	Wed Nov 24 02::
KerberosTokenIde...		ACTIVATED	Authentication	Wed Nov 24 02::
KerberosTokenAut...		ACTIVATED	Authentication	Wed Nov 24 02::
X509CredentialExt...		ACTIVATED	Authentication	Thu May 06 23:3
DBUserAuthentica...		activated	Authentication	Thu Jan 20 03:3

5. Perform the following tasks as needed:
   - Section 3.5.3, "Checking a Plug-in's Activation Status"
   - Section 3.5.4, "Deleting Custom Authentication Plug-ins"
   - Section 3.6, "Creating a Custom Authentication Module for Custom Plug-ins"

3.5.3 Checking a Plug-in's Activation Status

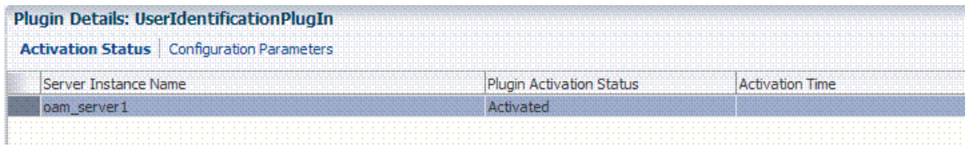
Users with valid administrator credentials can perform the following task to add, validate, distribute, and activate a custom plug-in.

Prerequisites

Developing an Authentication Plug-in

To check the activation status of a custom authentication plug-i

1. From the System Configuration tab, Common Configuration section, click Plugins and then click Open from the Actions menu.
2. In the Plugins table, click the desired plug-in name to select it.
3. **Server Instance Name**: Expand the Plugin Details section and click Activation Status to display the location and status of the plug-in. For example:



Plugin Details: UserIdentificationPlugIn

Activation Status | Configuration Parameters

Server Instance Name	Plugin Activation Status	Activation Time
oam_server1	Activated	

4. Perform the following tasks as needed:
   - Section 3.5, "Adding Custom Plug-ins"
   - Section 3.5.4, "Deleting Custom Authentication Plug-ins"
   - Section 3.6, "Creating a Custom Authentication Module for Custom Plug-ins"

3.5.4 Deleting Custom Authentication Plug-ins

Users with valid administrator credentials can use the following procedure to deactivate and then delete a custom plug-in.

When an administrator deletes a custom authentication plug-in, its name is not removed from the list of plug-ins. To delete the plug-in (for the purpose of re-importing the same plug-in later), the Administration must stop the WebLogic Server and edit the oam-config.xml manually.

Prerequisites

Adding Custom Plug-ins

To delete a custom authentication plug-in

1. Go to the Oracle Access Suite and log in, as usual. For example:

   https://*hostname*:*port*/oamconsole/

2. From the System Configuration tab, Common Configuration section, click Plugins and then click Open from the Actions menu.
3. **Deactivate the Plug-in**: You must perform this before removing a plug-in.
   a. In the Plugins table, click your plug-in name to select it.
   b. Click the Deactivate Selected button, then check the plug-ins Activation Status.
4. **Delete a Deactivated Plug-in**:
   a. In the Plugins table, click your plug-in name to select it.
   b. Click the Delete Selected button.
   c. Stop the WebLogic Administration Server, locate and edit oam-config.xml manually to remove the deactivated plug-in, and then restart the WebLogic Administration Server.
5. Perform the following tasks as needed:
   - Section 3.5, "Adding Custom Plug-ins"
   - Section 3.5.3, "Checking a Plug-in's Activation Status"
   - Section 3.6, "Creating a Custom Authentication Module for Custom Plug-ins"

3.6 Creating a Custom Authentication Module for Custom Plug-ins

This section provides the following topics:

- About Creating Custom Authentication Modules
- Creating a Custom Authentication Module

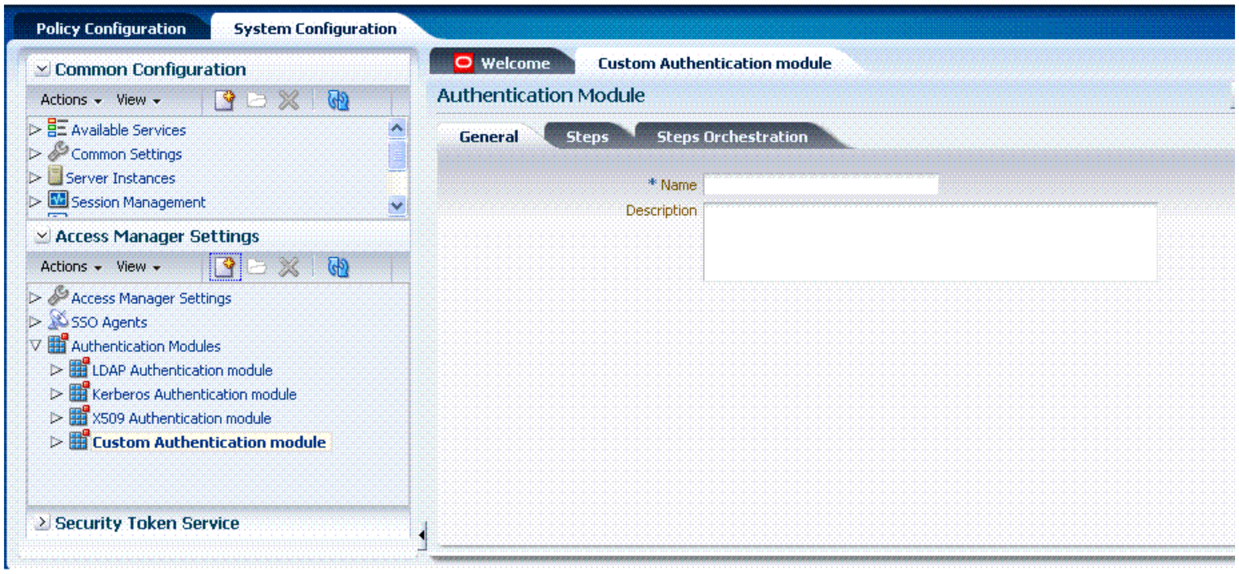
3.6.1 About Creating Custom Authentication Modules

The Access Manager Settings section of the System Configuration navigation tree includes the Authentication Modules node. When you create a custom authentication module, you are presented with subtabs for each type of information required for the module:

- General
- Steps
- Step Orchestration

Figure 3–16 shows the Authentication Modules node in the Access Manager Settings section of the System Configuration navigation tree, as well as the three subtabs where you enter information for the module.

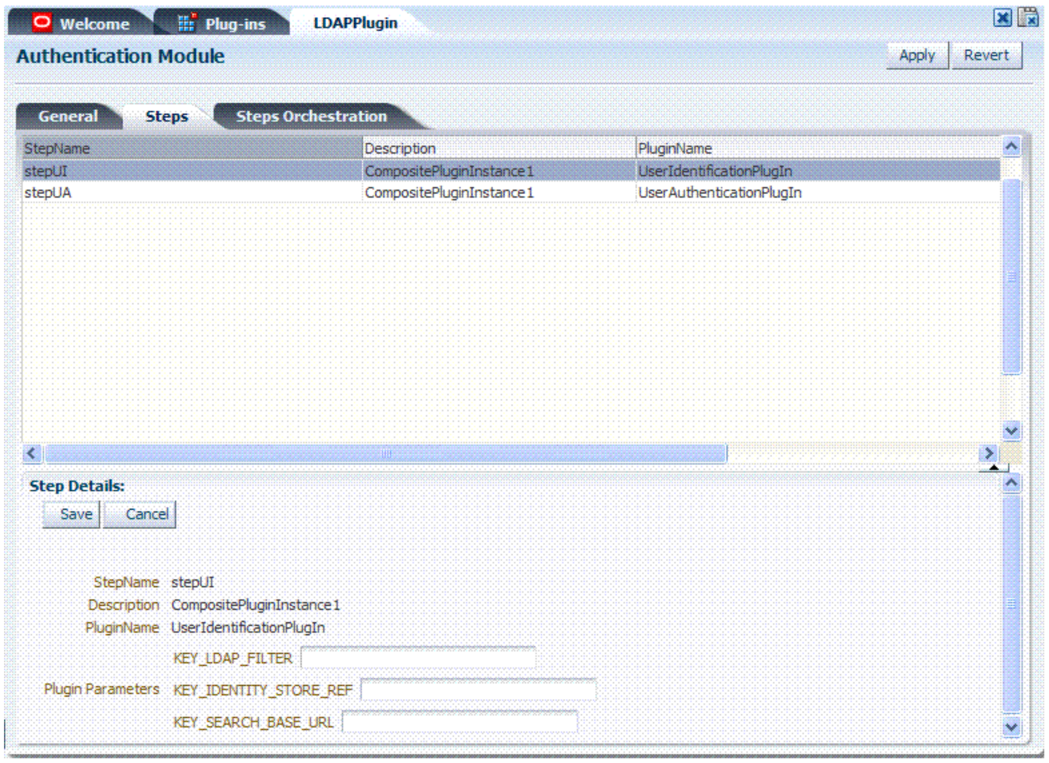
Figure 3–16 Custom Authentication Modules Node and General Subtab



The General subtab provides space for the module Name and an optional description. The name can be up to 60 characters. The optional description can be up to 250 characters.

Figure 3–17 illustrates the Steps subtab and Details section for a custom authentication module. Only valid values are accepted for each step as Plugin Parameters under Step Details. Invalid values result in an error when you attempt to save the custom authentication module. When adding Steps, there is no data to display in the table. However, once there are one or more Steps the table and Details sections are populated.

Figure 3–17 Custom Authentication Module Steps Subtab and Details Section

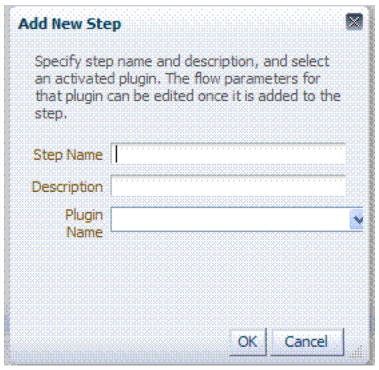


StepName	Description	PluginName
stepUI	CompositePluginInstance1	UserIdentificationPlugIn
stepUA	CompositePluginInstance1	UserAuthenticationPlugIn

StepName stepUI
Description CompositePluginInstance1
PluginName UserIdentificationPlugIn
Plugin Parameters KEY_LDAP_FILTER
KEY_IDENTITY_STORE_REF
KEY_SEARCH_BASE_URL

When you add a new Step, the following dialog box appears. Information that you enter is used to populate the table and Details sections of the page, as described in Table 3–7.

Figure 3–18 Adding a Step



Specify step name and description, and select an activated plugin. The flow parameters for that plugin can be edited once it is added to the step.

Step Name
Description
Plugin Name

Table 3–7 describes the information required for adding a new step.

Table 3–7 Add New Step Entries, Steps Results Table, and Details Section

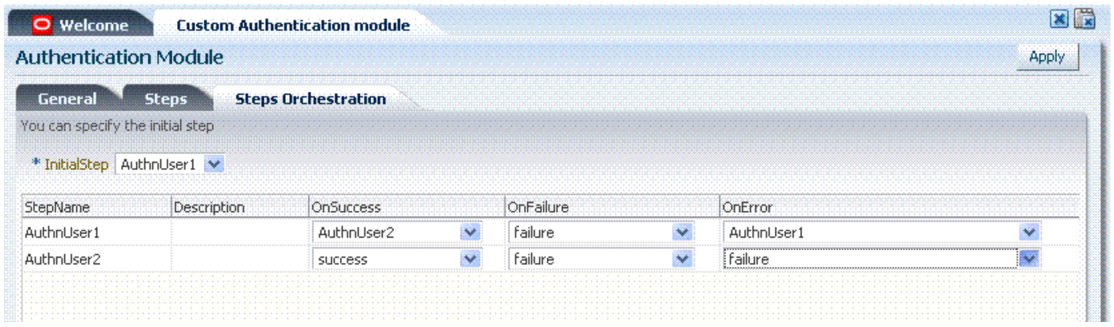
Element	Description
Step Name	The name that was entered when this step was added.
Description	The optional description for this step, entered when this step was added.
Plugin Name	The plug-in name that was selected when this step was added.

Table 3–7 (Cont.) Add New Step Entries, Steps Results Table, and Details Section

Element	Description
Step Details	Details of the selected step in the results table, and Plugin configuration details that were set when the plug-in was added and activated. See Also: Table 3–6, " Example of Plugin Details Extracted from XML Metadata File".

Figure 3–19 illustrates the Steps Orchestration subtab of a custom authentication module, which is populated by information for each defined step (and the action you choose for each operational condition).

Figure 3–19 Custom Authentication Module Steps Orchestration Subtab



You can specify the initial step

* InitialStep AuthnUser1

StepName	Description	OnSuccess	OnFailure	OnError
AuthnUser1		AuthnUser2	failure	AuthnUser1
AuthnUser2		success	failure	failure

Table 3–8 describes the elements on the Steps Orchestration subtab. The lists available for OnSuccess, OnFailure, and OnError include the following choices:

- success
- failure
- *StepName* (any step in the module can be selected as the action for an operational condition)

Table 3–8 Steps Orchestration Subtab

Element	Description
Step Name	The name that was entered when this step was added.
Description	The optional description for this step, entered when this step was added.
OnSuccess	The action selected for successful operation of this step.
OnFailure	The action selected for failure of this step.
OnError	The action selected for an error when executing this step.

3.6.2 Creating a Custom Authentication Module

Users with valid administrator credentials can use the following procedure to create an authentication module that uses one or more custom authentication plug-ins that were imported and activated in the Oracle Access Suite.

Note: You cannot duplicate an existing custom module to use as a template.

Prerequisites

Developing an Authentication Plug-in

Adding Custom Plug-ins

To create a custom authentication module to use custom plug-ins

1. From System Configuration tab, Access Manager Settings section, expand the Authentication Modules node.
2. From the navigation tree, click Custom Authentication Module.
3. Click the Create button in the tool bar.
4. Add General Information: Name and optional Description.
5. **Add a Step to The Module**:
    a. Click the Steps subtab.
    b. Click the Add button above the Steps table.
    c. In the Add New Step dialog box, enter the Step Name and optional Description.
    d. Browse for and select the desired custom plug-in name and click OK.
    e. Confirm information in the results table.
    f. Repeat b through e to add other steps until you have listed all required plug-ins for this module.
6. **Configure Each Step**: Use appropriate values for requested parameters:
    a. Click a StepName in the table to reveal required details.
    b. Enter valid values for the requested parameters.
    c. Click the Save button.
    d. Repeat a through c to configure each step appropriately.
7. **Orchestrate Step Usage**:
    a. Click the Steps Orchestration subtab.
    b. From the InitialStep list, choose the name of the first step to use.
    c. Select a StepName in the table.
    d. From the OnSuccess List, choose a condition (success or failure) or a step name name.
    e. From the OnFailure List, choose the desired condition or a *StepName*.
    f. From the OnError List, choose the desired condition or a *StepName*.
    g. Repeat c through e to orchestrate operations for each plug-in this module.
    h. Review your orchestration.
8. **Initiate Strategy Validation**: Click Apply to initiate validation of your orchestration strategy:
    - **Successful Strategy**: The orchestration strategy is applied and the module is ready to include in an authentication scheme. Continue with Steps 9 and 10.
    - **Invalid Strategy**: Click OK in the Error box, then edit your OnSuccess, OnFailure, OnError strategies (or add or remove plug-ins) to correct the problem. Repeat this step until your strategy is successful.

9. In the navigation tree, confirm the new Custom Authentication Module is listed, and then close the page when you finish.
10. "Creating Authentication Schemes with Custom Authentication Modules".

3.7 Creating Authentication Schemes with Custom Authentication Modules

Users with valid administrator credentials can use the following procedure to create a new authentication scheme that includes a custom authentication module.

This is essentially the same procedure that you would use when creating an authentication scheme with a standard authentication module. The only difference is that you can choose authentication modules with orchestrated steps that are defined to use custom plug-ins.

> **See Also:** *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service* for details about authentication schemes

Prerequisites

Creating a Custom Authentication Module for Custom Plug-ins

To create a custom authentication scheme

1. From the Policy Configuration tab, navigation tree, expand the Shared Components node.
2. Click the Authentication Schemes node, then click the Create button in the tool bar.
3. Fill in the fresh Authentication Scheme page:
   a. Name
   b. Description
   c. Authentication Level
   d. Default
   e. Challenge Method
   f. Challenge Redirect
   g. Authentication Module (includes those with custom plug-ins)
4. Click Apply to submit the new scheme (or close the page without applying changes).
5. Dismiss the Confirmation window.
6. Optional: Click the Set as Default button to automatically use this with new application domains, then close the Confirmation window.
7. In the navigation tree, confirm the new scheme is listed, and then close the page

3.8 Configuring Logging for Custom Plug-ins

Oracle Access Manager with Oracle Security Token Service uses the WebLogic Server container's logging defaults. To designate a custom Oracle Access Manager-specific logger and log handler with required attributes for custom plug-ins, you can use WLST commands as described here.

> **See Also:** *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*

To modify the logger and handler for a custom plug-in

1. Confirm that the OAM Server is running.
2. Acquire the custom WLST script for Oracle Access Manager. For example:

   ```
   <ORACLE_HOME>/common/bin/wlst.sh
   ```

3. Connect to the WebLogic Server and log in as the WebLogic administrator. For example:

   ```
   sh wlst.sh wls:/offline> connect ('adminID','password','adminURL')
   ```

4. List available loggers for the custom plug-in. For example:

   ```
   wls:/base_domain/serverConfig> listLoggers(pattern="oracle.oam.*",target="oam_
   server1")
   ```

   Here pattern= represents the oam.controller component and target= represents the desired OAM Server as it was specified during registration.
5. View the list of Oracle Access Manager loggers associated with this OAM Server. For example:

   ```
   Logger                                      | Level
   --------------------------------------------+-----------------
   oracle.oam                                  | <Inherited>
   oracle.oam.commonutil                       | <Inherited>
   oracle.oam.config                           | <Inherited>
   oracle.oam.controller                       | <Inherited>
   ...
   ```

6. Modify the oracle.oam.controller log level based on your requirements. For example, TRACE:32 with no persistence:

   ```
   wls:/base_domain/serverConfig> domainRuntime()
   wls:/base_domain/domainRuntime> setLogLevel(logger="oracle.oam.controller",
   level="TRACE:32", persist="0", target="oam_server1")
   ```

7. Repeat step 4 to list the loggers again and verify the log level change. For example:

   ```
   wls:/base_domain/serverConfig> listLoggers(pattern="oracle.oam.*",target="oam_
   server1")

   Logger                                      | Level
   --------------------------------------------+-----------------
   oracle.oam                                  | <Inherited>
   oracle.oam.commonutil                       | <Inherited>
   oracle.oam.config                           | <Inherited>
   oracle.oam.controller                       | TRACE:32
   ...
   ```

8. Verify the generated log file to confirm the controller is logged at the designated level:

   ```
   DOMAIN_HOME/server/SERVER_INSTNCE_NAME/logs/
   ```

9. Add a custom Oracle Access Manager-specific logger and log handler to specify a log file path and required attributes, as follows:

   a. Add oam logger, as follows:

      ```
      wls:/base_domain/serverConfig> domainRuntime>
      wls:/base_domain/domainRuntime> setLogLevel(logger="oracle.oam",level="WAR
      NING", persist="0", target="oam_server1")
      ```

   b. Add a custom log handler and associate it with oam logger, as shown here:

      ```
      wls:/base_domain/domainRuntime> configureLogHandler(name="oam-log-handler",
      target="oam_server1", rotationFrequency="daily", retentionPeriod="week",
      path="${domain.home}/oamlogs", maxFileSize ="10485760", maxLogSize =
      "104857600",addHandler="true", handlerType="oracle.core.ojdl.logging.ODLHan
      dlerFactory", addToLogger="oracle.oam")

      wls:/base_domain/domainRuntime>configureLogHandler(name="oam-log-handler",
      addProperty="true", propertyName="supplementalAttributes",
      propertyValue="OAM.USER, OAM.COMPONENT", target="oam_server1")
      ```

   c. Verify that all the OAM logs appear in the DOMAIN_HOME/oamlogs folder.

10. Verify the generated log file to confirm the controller is logged at the TRACE:32 level:

    ```
    DOMAIN_HOME/server/SERVER_INSTNCE_NAME/logs/
    ```

4 Writing Oracle Security Token Service Module Classes

This chapter discusses Oracle Access Manager 11g and Oracle Security Token Service custom token options. It includes the following sections:

4.1 Introduction to Oracle Security Token Service Custom Token Module Classes

When Oracle Security Token Service does not support the token that you want to validate or issue out-of-the-box, you can write your own validation and issuance module classes. One of the two (validation or issuance class) is required for custom tokens:

- Oracle Security Token Service uses the custom validation class to validate a custom token.
- Oracle Security Token Service uses the custom issuance class to issue a custom token.

Note: One of the two (validation or issuance class) is required for custom tokens.

The following overview outlines the tasks you must perform.

Task overview: Deploying custom token module classes

1. Writing a TokenValidatorModule Class to validate a custom token with Oracle Security Token Service, if needed.
2. Writing a TokenIssuanceModule Class to issue a custom token with Oracle Security Token Service, if needed.

3. Making Custom Classes Available to create a Custom Token module that will allow the user to create Validation Templates and Issuance Templates for their custom token.
4. Managing a Custom Oracle Security Token Service Configuration to create Validation and Issuance Templates for the custom token, and use the custom templates in Endpoints and Partner Profiles as you would use the templates of standard tokens.

4.2 Writing a TokenValidatorModule Class

This section provides the following topics:

- About Writing a TokenValidatorModule Class
- Writing a TokenValidatorModule Class

4.2.1 About Writing a TokenValidatorModule Class

The Oracle Security Token Service Validation module class implements the `oracle.security.fed.sts.token.tpe.TokenValidatorModule interface`. The following properties can be fetched from the `TokenContext` during the validation process:

- XML_TOKEN: The bytes of the XML message that contains the token that must be validated.
- BST_VALUE_TYPE: If the custom token is sent as a Binary Security Token, this will contain the Binary Security Token value type.
- BST_ENCODING: If the token is sent as a Binary Security Token, this will contain the encoding.
- BST_CONTENT: If the token is sent as a Binary Security Token, this will contain the Binary Security Token content.
- TOKEN_ELEMENT: If the token is not a binary security token and does not have a jaxb representation in the Oracle Security Token Service internal classes, this will contain the XML element or custom JAXB class representing the token.
- XML_DOM: This is the DOM representation of the incoming message. This will be present only if a DOM object was created as a part of Oracle Security Token Service processing thus far.

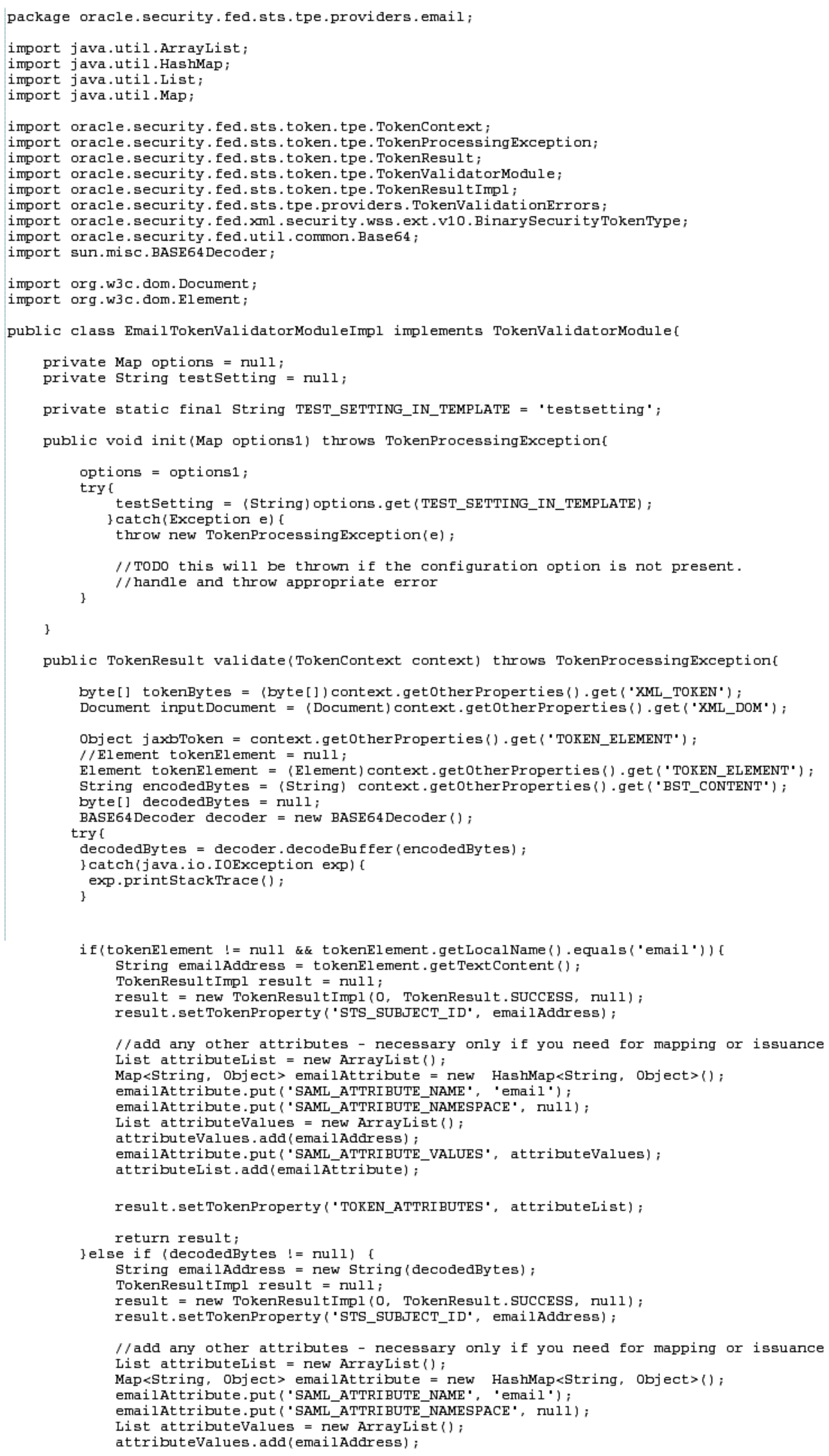
The token should be validated using the information in the properties in the `TokenContext` and a `TokenResult` should be returned. The following properties can be set on a `TokenResult` object to return information to Oracle Security Token Service:

- TPE_RESULT_FAILURE_CODE: The failure code if there was a failure.
- TPE_RESULT_FAILURE_STRING: A string describing the failure.
- Any other properties that are set in the result are available in the context to be used for token mapping. Usually, validators set STS_SUBJECT_ID property to the name ID and use this to map to a user record.

See the following figures contain examples for the full implementation of `EmailTokenValidatorModuleImplforBinary.java`:

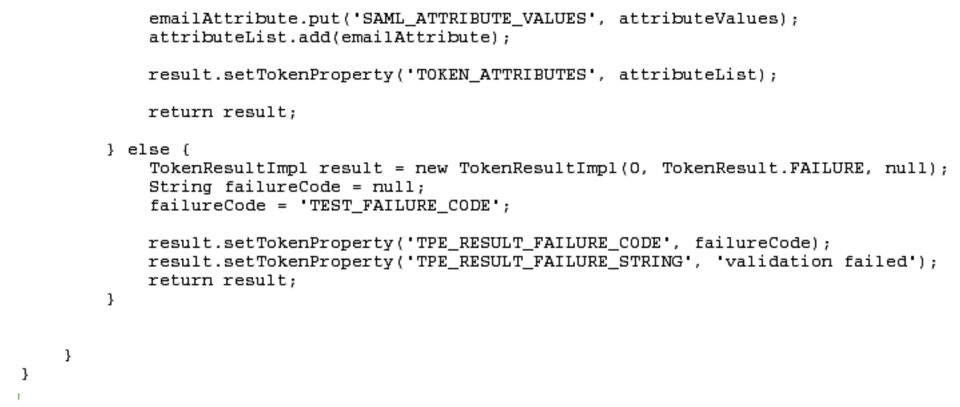
- Figure 4–1, "Part 1: EmailTokenValidatorModuleImplforBinary.java"
- Figure 4–2, "Part 2: EmailTokenValidatorModuleImplforBinary.java"

Figure 4–1 Part 1: EmailTokenValidatorModuleImplforBinary.java



```
package oracle.security.fed.sts.tpe.providers.email;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import oracle.security.fed.sts.token.tpe.TokenContext;
import oracle.security.fed.sts.token.tpe.TokenProcessingException;
import oracle.security.fed.sts.token.tpe.TokenResult;
import oracle.security.fed.sts.token.tpe.TokenValidatorModule;
import oracle.security.fed.sts.token.tpe.TokenResultImpl;
import oracle.security.fed.sts.tpe.providers.TokenValidationErrors;
import oracle.security.fed.xml.security.wss.ext.v10.BinarySecurityTokenType;
import oracle.security.fed.util.common.Base64;
import sun.misc.BASE64Decoder;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class EmailTokenValidatorModuleImpl implements TokenValidatorModule{

    private Map options = null;
    private String testSetting = null;

    private static final String TEST_SETTING_IN_TEMPLATE = 'testsetting';

    public void init(Map options1) throws TokenProcessingException{

        options = options1;
        try{
            testSetting = (String)options.get(TEST_SETTING_IN_TEMPLATE);
           }catch(Exception e){
            throw new TokenProcessingException(e);

            //TODO this will be thrown if the configuration option is not present.
            //handle and throw appropriate error
        }

    }

    public TokenResult validate(TokenContext context) throws TokenProcessingException{

        byte[] tokenBytes = (byte[])context.getOtherProperties().get('XML_TOKEN');
        Document inputDocument = (Document)context.getOtherProperties().get('XML_DOM');

        Object jaxbToken = context.getOtherProperties().get('TOKEN_ELEMENT');
        //Element tokenElement = null;
        Element tokenElement = (Element)context.getOtherProperties().get('TOKEN_ELEMENT');
        String encodedBytes = (String) context.getOtherProperties().get('BST_CONTENT');
        byte[] decodedBytes = null;
        BASE64Decoder decoder = new BASE64Decoder();
       try{
        decodedBytes = decoder.decodeBuffer(encodedBytes);
        }catch(java.io.IOException exp){
         exp.printStackTrace();
        }


        if(tokenElement != null && tokenElement.getLocalName().equals('email')){
            String emailAddress = tokenElement.getTextContent();
            TokenResultImpl result = null;
            result = new TokenResultImpl(0, TokenResult.SUCCESS, null);
            result.setTokenProperty('STS_SUBJECT_ID', emailAddress);

            //add any other attributes - necessary only if you need for mapping or issuance
            List attributeList = new ArrayList();
            Map<String, Object> emailAttribute = new  HashMap<String, Object>();
            emailAttribute.put('SAML_ATTRIBUTE_NAME', 'email');
            emailAttribute.put('SAML_ATTRIBUTE_NAMESPACE', null);
            List attributeValues = new ArrayList();
            attributeValues.add(emailAddress);
            emailAttribute.put('SAML_ATTRIBUTE_VALUES', attributeValues);
            attributeList.add(emailAttribute);

            result.setTokenProperty('TOKEN_ATTRIBUTES', attributeList);

            return result;
        }else if (decodedBytes != null) {
            String emailAddress = new String(decodedBytes);
            TokenResultImpl result = null;
            result = new TokenResultImpl(0, TokenResult.SUCCESS, null);
            result.setTokenProperty('STS_SUBJECT_ID', emailAddress);

            //add any other attributes - necessary only if you need for mapping or issuance
            List attributeList = new ArrayList();
            Map<String, Object> emailAttribute = new  HashMap<String, Object>();
            emailAttribute.put('SAML_ATTRIBUTE_NAME', 'email');
            emailAttribute.put('SAML_ATTRIBUTE_NAMESPACE', null);
            List attributeValues = new ArrayList();
            attributeValues.add(emailAddress);
```

Figure 4–2 Part 2: EmailTokenValidatorModuleImplforBinary.java



```
            emailAttribute.put('SAML_ATTRIBUTE_VALUES', attributeValues);
            attributeList.add(emailAttribute);

            result.setTokenProperty('TOKEN_ATTRIBUTES', attributeList);

            return result;

        } else {
            TokenResultImpl result = new TokenResultImpl(0, TokenResult.FAILURE, null);
            String failureCode = null;
            failureCode = 'TEST_FAILURE_CODE';

            result.setTokenProperty('TPE_RESULT_FAILURE_CODE', failureCode);
            result.setTokenProperty('TPE_RESULT_FAILURE_STRING', 'validation failed');
            return result;
        }


    }
}
```

The following overview outlines development highlights for this module class.

Development highlights: Writing a TokenValidatorModule class

1. Implement the `init(Map options)` method, called when the `TokenValidatorModule` is initialized. The `init` method is passed in a map containing the parameters defined in the validation template.
2. Implement the `validate(TokenContext context)` method, called when a particular incoming custom token must be validated.
   a. Fetch token information from the properties in the `TokenContext` object.
   b. Validate the token and return a `TokenResult` object:

      On Success, return:

      ```
      TokenResultImpl result = new TokenResultImpl(0, TokenResult.SUCCESS,
      token);
      ```

      On Failure, return:

      ```
      TokenResultImpl result = new TokenResultImpl(0, TokenResult.FAILURE,
      token);
      result.setTokenProperty("TPE_RESULT_FAILURE_CODE", failureCode);
      result.setTokenProperty("TPE_RESULT_FAILURE_STRING", "validation failed");
      ```

   c. Confirm the validated token result returns the `NameId` in the token and any attributes that are parsed from the token, in the following format:

      ```
      result.setTokenProperty(TPEConstants.NAMEID_VALUE, emailAddress);

       //attributes
      List attributeList = new ArrayList();
      Return any other properties that should be available in the context for :
      token mapping and issuance by setting the properties on the result
      result.setTokenProperty(name, value);
      Return any other properties that should be available in the context for
      token mapping and issuance by setting the properties on the result as
      follows: result.setTokenProperty(name, value);
      emailAttribute.put(TPEConstants.SAML_ATTRIBUTE_NAMESPACE, null);
      List attributeValues = new ArrayList();
      attributeValues.add(emailAddress);
      emailAttribute.put(TPEConstants.SAML_ATTRIBUTE_VALUES, attributeValues);
      attributeList.add(emailAttribute);
      ```

```
        result.setTokenProperty(TPEConstants.TOKEN_ATTRIBUTES, attributeList);
```

4.2.2 Writing a TokenValidatorModule Class

Perform the following tasks to write a custom TokenValidatorModule class.

Task overview: Writing a TokenValidatorModule class

1. Develop your own module class while referring to:
   - Section 4.2.1, "About Writing a TokenValidatorModule Class"
   - *Oracle Security Token Service Java API Reference*
2. Proceed as needed:
   - Section 4.3, "Writing a TokenIssuanceModule Class"
   - Section 4.4, "Making Custom Classes Available"

4.3 Writing a TokenIssuanceModule Class

This section provides the following topics:

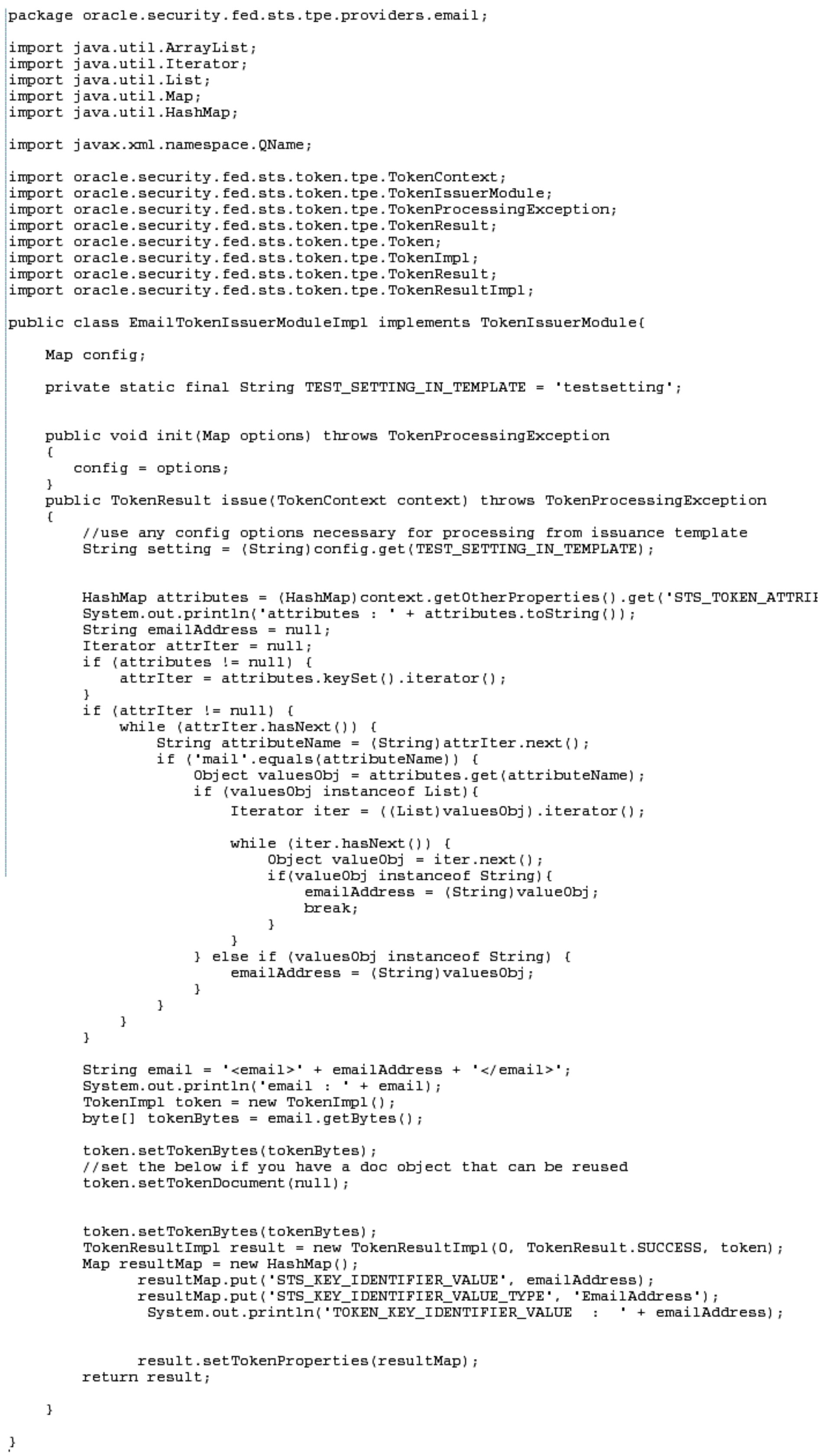
- About Writing a TokenIssuanceModule Class
- Writing a TokenIssuanceModule Class

4.3.1 About Writing a TokenIssuanceModule Class

The `EmailTokenIssuerModuleImpl.java` class should implement the `oracle.security.fed.sts.token.tpe.TokenIssuerModule` interface and attributes in the `TokenContext`.

See Figure 4–3 for an example. The overview that follows outlines development highlights for this module class.

Figure 4–3 EmailTokenIssuerModuleImpl.java



```
package oracle.security.fed.sts.tpe.providers.email;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.HashMap;

import javax.xml.namespace.QName;

import oracle.security.fed.sts.token.tpe.TokenContext;
import oracle.security.fed.sts.token.tpe.TokenIssuerModule;
import oracle.security.fed.sts.token.tpe.TokenProcessingException;
import oracle.security.fed.sts.token.tpe.TokenResult;
import oracle.security.fed.sts.token.tpe.Token;
import oracle.security.fed.sts.token.tpe.TokenImpl;
import oracle.security.fed.sts.token.tpe.TokenResult;
import oracle.security.fed.sts.token.tpe.TokenResultImpl;

public class EmailTokenIssuerModuleImpl implements TokenIssuerModule{

    Map config;

    private static final String TEST_SETTING_IN_TEMPLATE = 'testsetting';


    public void init(Map options) throws TokenProcessingException
    {
       config = options;
    }
    public TokenResult issue(TokenContext context) throws TokenProcessingException
    {
        //use any config options necessary for processing from issuance template
        String setting = (String)config.get(TEST_SETTING_IN_TEMPLATE);


        HashMap attributes = (HashMap)context.getOtherProperties().get('STS_TOKEN_ATTRI
        System.out.println('attributes : ' + attributes.toString());
        String emailAddress = null;
        Iterator attrIter = null;
        if (attributes != null) {
            attrIter = attributes.keySet().iterator();
        }
        if (attrIter != null) {
            while (attrIter.hasNext()) {
                String attributeName = (String)attrIter.next();
                if ('mail'.equals(attributeName)) {
                    Object valuesObj = attributes.get(attributeName);
                    if (valuesObj instanceof List){
                        Iterator iter = ((List)valuesObj).iterator();

                        while (iter.hasNext()) {
                            Object valueObj = iter.next();
                            if(valueObj instanceof String){
                                emailAddress = (String)valueObj;
                                break;
                            }
                        }
                    } else if (valuesObj instanceof String) {
                        emailAddress = (String)valuesObj;
                    }
                }
            }
        }

        String email = '<email>' + emailAddress + '</email>';
        System.out.println('email : ' + email);
        TokenImpl token = new TokenImpl();
        byte[] tokenBytes = email.getBytes();

        token.setTokenBytes(tokenBytes);
        //set the below if you have a doc object that can be reused
        token.setTokenDocument(null);


        token.setTokenBytes(tokenBytes);
        TokenResultImpl result = new TokenResultImpl(0, TokenResult.SUCCESS, token);
        Map resultMap = new HashMap();
              resultMap.put('STS_KEY_IDENTIFIER_VALUE', emailAddress);
              resultMap.put('STS_KEY_IDENTIFIER_VALUE_TYPE', 'EmailAddress');
               System.out.println('TOKEN_KEY_IDENTIFIER_VALUE  :  ' + emailAddress);


              result.setTokenProperties(resultMap);
        return result;

    }

}
```

Development highlights: Writing a TokenIssuanceModule class

1. Implement the `public void init(Map options)throws TokenProcessingException` method.

   The `init()` method is called when the issuer module is initialized. The `init` method is passed a map contain the parameters defined in the issuance template.

2. Implement the `public TokenResult issue(TokenContext context) throws TokenProcessingException` method.

   This method is called when a custom outgoing token must be created.

   a. Create, within the `issue` method, the token using the attributes in the issuance template and the attributes passed in the `TokenContext`. Attributes in the `TokenContext` are accessed in the following way:

   ```
   List attributes =
   (List)context.getOtherProperties().get(TPEConstants.TOKEN_ATTRIBUTES);
   String emailAddress = null;
   HashMap attributes = (HashMap)context.getOtherProperties().get("STS_TOKEN_
   ATTRIBUTES");
   Object valueObj = attributes.get("mail"); //valuesObj will be a list if
   mail has more than 1 value;
   if(attributes != null)
   attrIter = attributes.iterator();
   if(attrIter != null){
   HashMap attributes = (HashMap)context.getOtherProperties().get("STS_TOKEN_
   ATTRIBUTES");
   Object valueObj = attributes.get("mail"); //valuesObj will be a list if
   mail has more than 1 value.
   Map<String, Object> attribute = attrIter.next();
   String attributeName = (String)attribute.get(TPEConstants.SAML_ATTRIBUTE_
   NAME);
   if("mail".equals(attributeName)){
   {Object valuesObj = attribute.get(TPEConstants.SAML_ATTRIBUTE_VALUES);
   if(valuesObj instanceof List) { Iterator iter =
   ((List)valuesObj).iterator(); while(iter.hasNext())
   {Object valueObj = iter.next(); if(valueObj instanceof String)

   }
   }else if(valuesObj instanceof String)
   Unknown macro: { emailAddress = (String)valuesObj; }
   }
   ```

   b. Create a result object and set the bytes of the token and the Document Object Model (DOM) representation of the token (only if the DOM representation was created during the processing in this class):

   ```
   token.setTokenDocument(null);--> if you have a doc object that can be
   reuse.d set it here
   token.setTokenBytes(tokenBytes);
   TokenResult result = new TokenResultImpl(0, TokenResult.SUCCESS, token);
   ```

   c. Set the key identifier information into the token properties, as follows:

   ```
   Map resultMap = new HashMap();
      resultMap.put("STS_KEY_IDENTIFIER_VALUE", emailAddress);
      resultMap.put("STS_KEY_IDENTIFIER_VALUE_TYPE", "EmailAddress");
      result.setTokenProperties(resultMap);
   ```

4.3.2 Writing a TokenIssuanceModule Class

Task overview: Writing an Issuance Module class

1. Write the issuance module class as you refer to Section 4.3.1, "About Writing a TokenIssuanceModule Class" and *Oracle Security Token Service Java API Reference*.
2. Proceed to Section 4.4, "Making Custom Classes Available"

4.4 Making Custom Classes Available

This section describes how to make custom classes available to Oracle Access Manager 11g using the console.

The information here can be applied when you have:

- WS-Security User Name Token
- WS-Trust Custom Token
- Issuing Custom Token

Note: You can also write a script that includes WebLogic Scripting Tool commands for any operation that you can accomplish through the console. For more information, see *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.

This section provides the following topics:

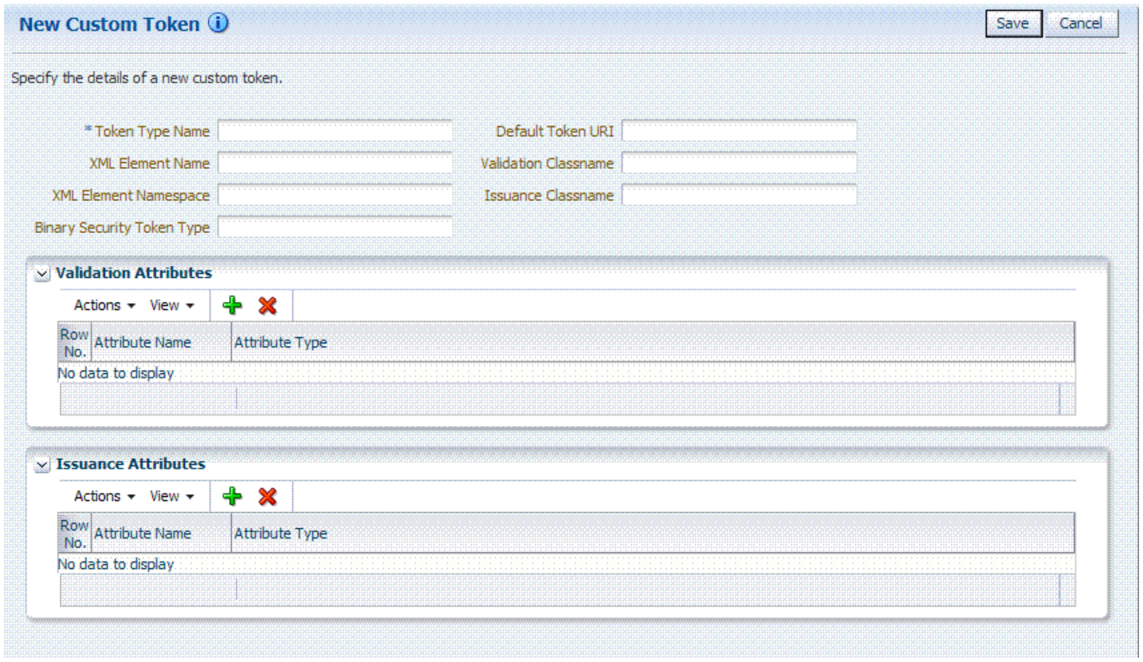
- About Making Classes Available
- About Narrowing a Search for Custom Tokens
- Managing Custom Tokens

4.4.1 About Making Classes Available

After writing the custom token validation and/or issuance classes, you must add Custom Token Configuration to Oracle Security Token Service to indicate when and how these classes should be used.

On the New Custom Token page only the Token Type Name is required (identified with an asterisk, *), as shown in Figure 4–4. Not all elements apply to all custom tokens. However, if you submit information that is incomplete, a dialog box appears to identify what is missing.

Figure 4–4 New Custom Token Page



Specify the details of a new custom token.

* Token Type Name		Default Token URI	
XML Element Name		Validation Classname	
XML Element Namespace		Issuance Classname	
Binary Security Token Type			

Row No.	Attribute Name	Attribute Type

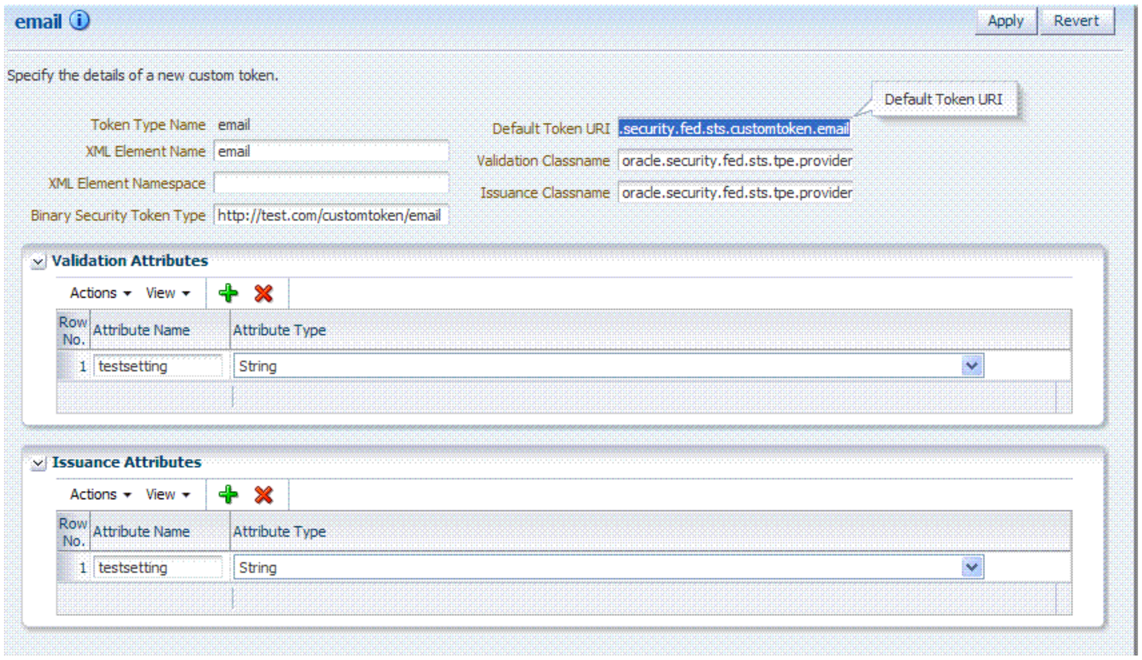
No data to display

Row No.	Attribute Name	Attribute Type

No data to display

After successful submission of new custom token details, the saved page is available for editing as shown in Figure 4–5.

Figure 4–5 Custom Token Definition: email



Specify the details of a new custom token.

Token Type Name	email	Default Token URI	.security.fed.sts.customtoken.email
XML Element Name	email	Validation Classname	oracle.security.fed.sts.tpe.provider
XML Element Namespace		Issuance Classname	oracle.security.fed.sts.tpe.provider
Binary Security Token Type	http://test.com/customtoken/email		

Row No.	Attribute Name	Attribute Type
1	testsetting	String

Row No.	Attribute Name	Attribute Type
1	testsetting	String

For the custom token, you must decide on the XML Element Name, XML Element Namespace, Binary Security Token Type, and so on. Table 4–1 describes the elements on a Custom Token page based on the examples in this chapter.

Table 4–1 New Custom Token Elements

Element	Description
Token Type Name	The unique name you choose for this custom token. For example: `email_token` Note: After you save a new custom token configuration, you cannot edit this name.
Default Token URI	The URI for this custom token. This URI can then be used in the RST to request that a custom token of this type should be issued. For the example in this chapter, the value would be: `oracle.security.fed.sts.customtoken.email`
XML Element Name	The name you decide on, which will be associated with the Token Type Name. For example: `email` If you specify `email` as the XML Element Name, each time the element name, `email`, appears in an incoming token it will be associated with the Token Type Name (in this case `email_token`). Note: Minimally, you need either an XML Element Name or Binary Security Token Type.
Validation Classname	The name of the custom token validation class that you made available to Oracle Security Token Service. For example: `oracle.security.fed.sts.tpe.providers.email.EmailTokenValidatorModuleImpl` Note: Minimally, you need either an issuance class name or validation class name, depending on whether you want to issue or validate a custom token.
XML Element Namespace	The namespace of the custom token element name. For example: `http://email.example.com`
Issuance Classname	The name of the custom token issuance class that you made available to Oracle Security Token Service. For example: `oracle.security.fed.sts.tpe.providers.email.EmailTokenIssuerModuleImpl` Note: Minimally, you need either an Issuance classname or Validation classname, depending on whether you want to issue or validate a custom token.
Binary Security Token Type	Enables the class to validate a custom token sent in as a BinarySecurityToken. The ValueType of the BinarySecurityToken for this custom token. If Oracle Security Token Service receives a Binary Security Token with this valuetype, it will be forwarded to this custom token's Validation class for validation.
Validation Attributes	This section enables you to add (or remove) validation attributes. The table displays existing validation attributes, if any. For this example: ■ Attribute Name: `testsetting` ■ Attribute Type: `String` Note: You will add a value to the attribute when creating a Token Validation Template.

Table 4–1 (Cont.) New Custom Token Elements

Element	Description
Issuance Attributes	This section enables you to add (or remove) issuance attributes. The table displays the following information for existing issuance attributes. ■ Attribute Name: `testsetting` ■ Attribute Type: `String` Note: You will add a value to the attribute when creating a Token Issuance Template.
Save	Click this button on the New Custom Tokens page to save your configuration information.
Cancel	Click this button to dismiss your configuration details.
Apply	Click this button to submit your changes.
Revert	Click this button to dismiss your changes.

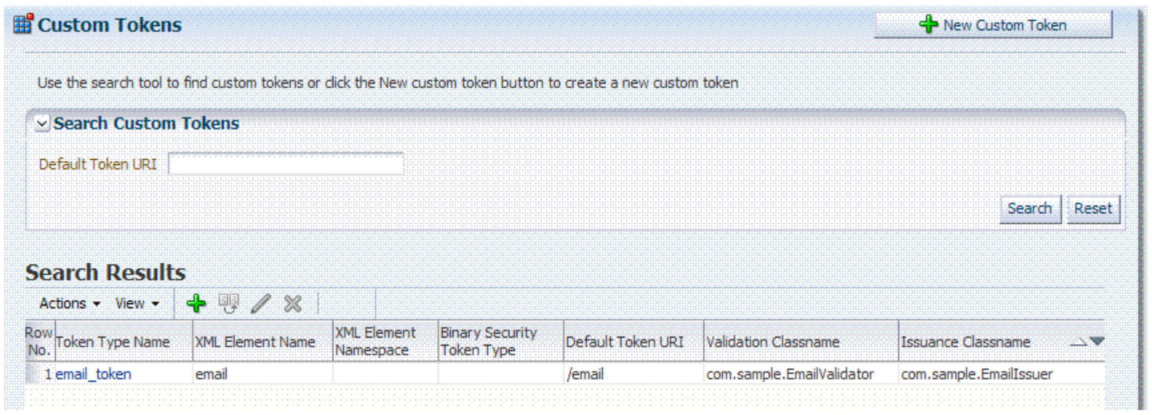
Task overview: Adding custom tokens for custom classes

1. Create a JAR file containing only your custom `TokenIssuerModule or TokenValidatorModule` classes (or both). No XML metadata or manifest is needed.
2. Review information in Figure 4–5 and Table 4–1.
3. Add the JAR to the OAM Server hosting Oracle Security Token Service and create a new custom token, as described in Section 4.4.3, "Managing Custom Tokens".

4.4.2 About Narrowing a Search for Custom Tokens

Figure 4–6 illustrates the Custom Tokens Search controls and Results table. These appear when you double-click the Custom Tokens node in the navigation tree. By default, all currently defined custom tokens are listed when the Search Results table is displayed.

Figure 4–6 Custom Tokens Search Page and Controls



Use the search tool to find custom tokens or click the New custom token button to create a new custom token

Search Results

Row No.	Token Type Name	XML Element Name	XML Element Namespace	Binary Security Token Type	Default Token URI	Validation Classname	Issuance Classname
1	email_token	email			/email	com.sample.EmailValidator	com.sample.EmailIssuer

Table 4–2 describes the Custom Tokens Search elements and controls. No wild cards (*) are allowed in Custom Token searches.

Table 4–2 Custom Tokens Search Elements and Controls

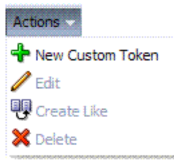
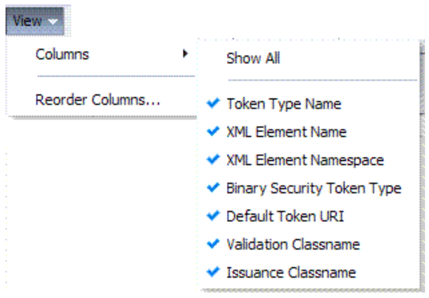
Element	**Description**
Default Token URI	The URI that was defined for the custom token. You can enter the entire URI or only part of it. For instance, if you enter "ai" the Search Results table will display all custom tokens defined with a token URI that includes the letters "ai". Note: Wild cards are not allowed in Custom Token searches.
Search	Initiates the Search function using criteria provided in the form.
Reset	Resets the Search form with defaults only.
Search Results	Provides the results of your search based on your choices in the View menu.
Actions menu	Provides the following functions that can be performed on a selection in the results table:  Actions New Custom Token Edit Create Like Delete Note: Actions menu functions mirror command buttons above the results table. For example: ■ New Custom Token: Click the New Custom Token button at the top of the Search page, or select New Custom Token from the menu, or click the **+** button above the table. ■ Edit: Double-click a name in the Token Type Name column of the Search Results table, or select Edit from the Actions menu, or click the Edit (pencil icon) command button above the Results Table. ■ Create Like: Select the desired row in the table and either select Create Like from the Actions menu, or click the Create Like command button above the table ■ Remove: Select the desired row in the table and either select Delete from the Actions menu, or click the Delete (**X**) command button above the table.
View menu	Provides functions you can use to display various information in the results table:  View Columns Reorder Columns... Show All Token Type Name XML Element Name XML Element Namespace Binary Security Token Type Default Token URI Validation Classname Issuance Classname

Table 4–2 (Cont.) Custom Tokens Search Elements and Controls

Element	Description
△▽	Controls affecting the ordering of items listed in the results table: ■ Ascending ■ Descending

4.4.3 Managing Custom Tokens

Users with valid administrator credentials can use the procedure in this section to manage custom tokens for custom Token Module classes.

The following procedure includes steps to add, edit, and delete custom tokens or attributes of a custom token. Skip any steps that you do not need.

Prerequisites

Writing a TokenValidatorModule Class

Writing a TokenIssuanceModule Class

See Also:

- Section 4.4.1, "About Making Classes Available"
- Section 4.4.2, "About Narrowing a Search for Custom Tokens"

To make custom classes available

1. Create and add the JAR containing your Issuance and Validation classes to the OAM Server hosting Oracle Security Token Service using one of these methods:
   - Add the custom token jar and the sts-common.jar that is available in *DOMAIN_HOME*/config/fmwconfig/mbeans/oam to the Managed Server classpath by editing the startup script.
   - Add the custom token jar and the sts-common.jar that is available in *DOMAIN_HOME*/config/fmwconfig/mbeans/oam to the *DOMAIN_HOME*/lib directory to automatically add these jars to the Managed Server classpath.
   - Restart the OAM Server.
2. **New Custom Token**: From the Oracle Access Suite System Configuration tab, open the Security Token Services section and:
   a. Double-click the Custom Tokens node to open the page.
   b. Click the New Custom Token button.
   c. Fill in the New Custom Token page with details for your custom classes (Table 4–1).
   d. Click Save and dismiss the confirmation window (or click Cancel to dismiss the page without submitting it).
   e. Close the page (or edit as described in Step 4).
   f. Proceed to Step 4, if needed, or to Section 4.5, "Managing a Custom Oracle Security Token Service Configuration".
3. **Find Custom Tokens**: From the Security Token Service section of the System Configuration tab:

   a. **Find All**: Double-click the Custom Tokens node to display a results table with all custom tokens listed.
   b. **Narrow the Search**: Enter some or all characters in the desired Default Token URI, click the **Search** Button, and review the results table.
   c. **Reset the Search Form**: Click the Reset button.

4. **Edit Custom Token Configuration**: Start with the saved page you just created.

   Alternatively: Use Step 3 to find the desired Custom Token, then double-click the name in the Search Results table to open the page.

   a. In the named Custom Token page, click the appropriate field and edit as needed.
   b. **Add Attributes**: Click the Add (+) icon for the Attributes table, enter the Attribute Name and an Attribute Type (Table 4–1).
   c. **Remove Attributes**: From the Attributes table, click the row containing the attribute to remove, click the Delete (X) icon for the table, and dismiss the Confirmation window.
   d. **Apply Changes**: Click the Apply button at the top of the page to submit changes.

5. **Remove a Custom Token**:

   a. Click the desired name in the Search Results table to select the item to remove.
   b. From the Actions menu, click Delete (or click the Delete (X) command button above the table.
   c. Click the Delete button in the Confirmation window (or click No to cancel the operation).

4.5 Managing a Custom Oracle Security Token Service Configuration

This tasks consists of the following procedures:

- Creating the Validation Template
- Creating the Issuance Template for a Custom Token
- Adding the Custom Token to a Requester Profile
- Adding the Custom Token to the Relying Party Profile
- Mapping the Token to a Requestor
- Creating an /wssuser EndPoint

4.5.1 Creating the Validation Template

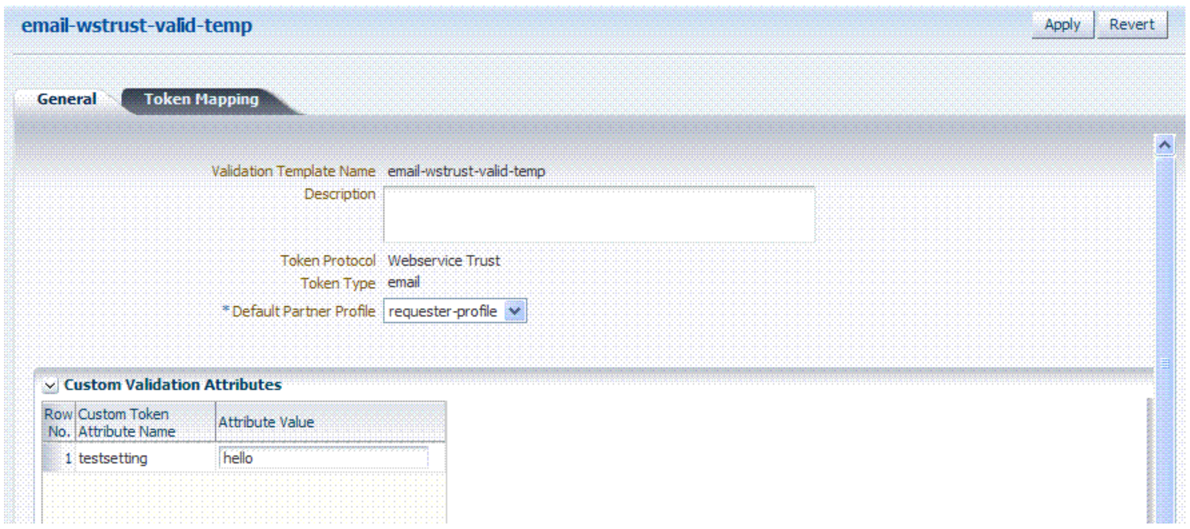
Users with valid Oracle Access Manager administrator credentials can perform the following task to create a Validation Template with a Token Protocol of Webservice Trust to map the token to the requester.

The template in this example can be used for the module classes described earlier in this chapter. Full implementation details are shown in the following figures. As you review these, notice how specifications for this template reference the module class code:

- Figure 4–7, "General Details: email-wstrust-valid-temp"

- Figure 4–8, "Token Mapping: email-wstrust-valid-temp"

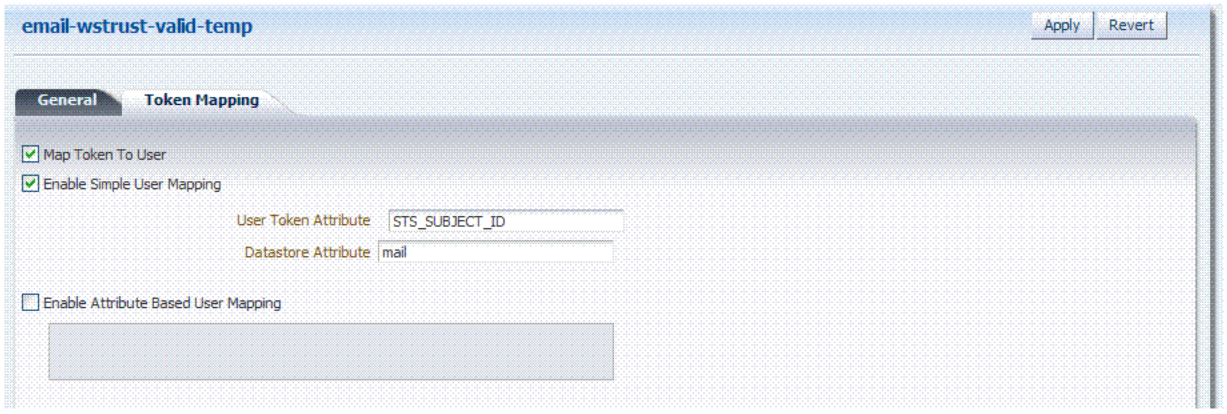
Figure 4–7 General Details: email-wstrust-valid-temp



Validation Template Name	email-wstrust-valid-temp
Description	
Token Protocol	Webservice Trust
Token Type	email
* Default Partner Profile	requester-profile

Row No.	Custom Token Attribute Name	Attribute Value
1	testsetting	hello

Figure 4–8 Token Mapping: email-wstrust-valid-temp



☑ Map Token To User
☑ Enable Simple User Mapping

User Token Attribute	STS_SUBJECT_ID
Datastore Attribute	mail

☐ Enable Attribute Based User Mapping

> **See Also:** *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*

To create the validation template for the custom module classes

1. Display the list of existing Token Validation Templates.

   Oracle Access Suite
     System Configuration
       Security Token Services
         Token Validation Templates

2. Click the New Validation Template button in the upper-right corner (or click the Add (+) command button above the Search Results table).
3. General: Set the following for use with the custom token.

Validation Template Name: email-wstrust-valid-temp

Token Protocol: Webservice Trust

Token Type: email

Default Partner Profile: requester-profile

Custom Validation Attributes: testsetting: hello

4. Token Mapping: Set the following for use with the custom token in this chapter.

   Check the box beside Map Token To User (to enable it).

   Check the box beside Enable Simple User Mapping and enter:

   User Token Attribute: STS_SUBJECT_ID
   Datastore Attribute: mail

5. Click Save and dismiss the confirmation window.
6. Proceed to "Creating the Issuance Template for a Custom Token".

4.5.2 Creating the Issuance Template for a Custom Token

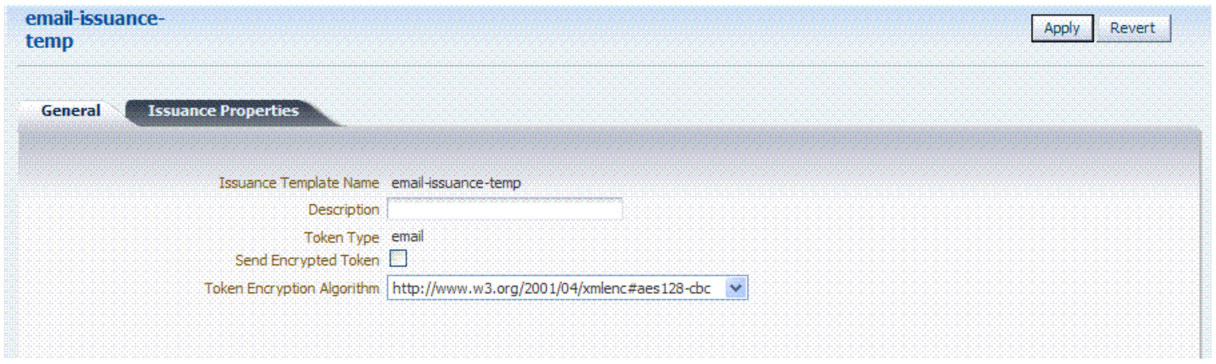
This is a server side configuration. Users with valid Oracle Access Manager administrator credentials can perform the following task to create a Token Issuance Template.

Each Token Issuance Template indicates how to construct a token, and which signing or encryption to use when constructing a token. Each Token Issuance Template also defines the attributes to be sent as part of the outbound token for mapping, and filtering data. However, Issuance Templates do not list mapping or filtering rules, which are defined in the Relying Party Partner Profile.

The template in this example can be used for the email custom token described earlier in this chapter. Implementation details are shown in the following figures, and described in the accompanying procedure. As you review these, notice how specifications for this template reference the module class code:

- Figure 4–9, "General Details: email-issuance-temp"
- Figure 4–10, "Issuance Properties: email-issuance-temp"

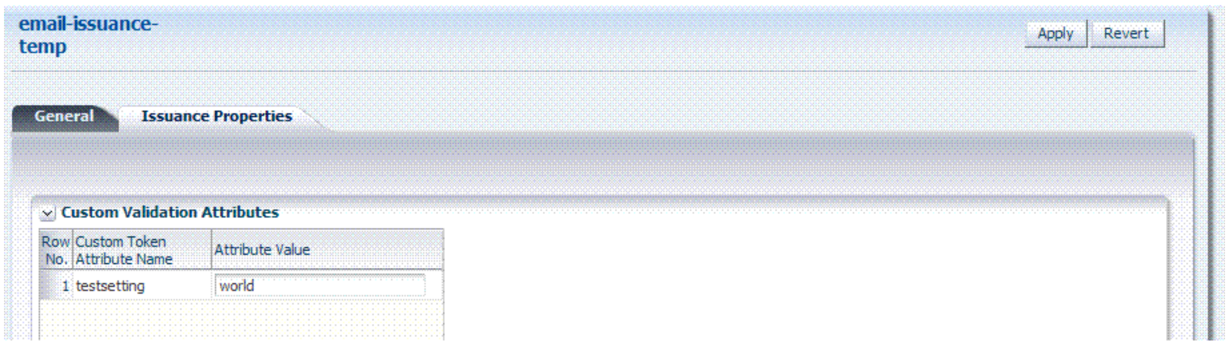
Figure 4–9 General Details: email-issuance-temp



Issuance Template Name email-issuance-temp
Description
Token Type email
Send Encrypted Token ☐
Token Encryption Algorithm http://www.w3.org/2001/04/xmlenc#aes128-cbc

When you have a custom token type deployed, the Issuance Properties are tailored to accommodate the custom token. For instance, the custom email token type was chosen for the issuance template show in Figure 4–10.

Figure 4–10 Issuance Properties: email-issuance-temp



Row No.	Custom Token Attribute Name	Attribute Value
1	testsetting	world

This procedure produces a companion Issuance Template for the custom module classes in this chapter. For the example:

- Ignore the Token Encryption Algorithm, which is not used for the custom token type: email.
- Fill in a value for the Custom Token Attribute, which is populated from the custom token code.

> **See Also:** *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*

To create the Issuance Template for the custom module classes

1. Go to the list of existing Token Issuance Templates.

   Oracle Access Suite
     System Configuration
       Security Token Services
         Token Issuance Templates

2. **New Token Issuance Template**:

   a. Click the New Issuance Template button in the upper-right corner (or click the Add (+) command button above the Search Results table).

   b. General: Set the following for use with the custom token in this chapter.

      Issuance Template Name: email-issuance-temp
      Token Type: email

   c. Click Save and dismiss the confirmation window (or click Cancel without saving).

   d. Issuance Properties: Set the following for use with the custom token in this chapter.

      Custom Token Attribute Value: world

   e. Click Apply and dismiss the confirmation window (or click Revert without saving it).

   f. Close the definition (or edit it as described in Step 4).

3. Edit a Template: Start with the saved page you just created.

   Alternatively: Use Step 3 to find the desired template and click the name in the Search Results table to display the definition.

   a. Edit details as needed.

   b. Click the Apply button at the top of the page to submit changes (or Revert to undo your changes).

4.5.3 Adding the Custom Token to a Requester Profile

You can either edit an existing requester profile to add your custom token to the Token Type Configuration table, or create a new requester profile to use with the custom token. Either way, configure:

- Token Type: email (your custom token)
- Validation Template: email-wstrust-valid-temp

Prerequisites

Your Custom Token and Validation Template must be defined.

To create or edit a requester profile for the custom token

1. From the Oracle Access Suite System Configuration tab, open the Security Token Services section.

2. In the navigation tree, open the Partner Profiles node and double-click the Requestor Profiles node to display a list of existing profiles

3. **Existing Profile**:

   a. In the Search Results table of the Requester Profiles page, click the name of the desired profiles.

   b. Token and Attributes: Fill in the following details for the custom token in this chapter and then click the Save button at the top of the page.

      Token type: `email`
      Validation Template: `email-wstrust-valid-temp`

   c. Click Save, dismiss the confirmation window, and close the page (or click Cancel to dismiss the page without submitting it).

   d. Proceed to Section 4.5.4, "Adding the Custom Token to the Relying Party Profile".

4. **New Profile**: Click the New Requester Profile button to display the New Partner Profile page where you enter details:

   a. **General**: Fill in the following details for the custom token in this chapter and then click the Next button at the top of the page.

      Profile ID: *`unique_requesterprofile_name`*
      Default Relying Party Profile: *`unique_relyingparty_name`*

   b. **Add Token Type Configuration**: Fill in the following details for the custom token in this chapter and then click the **Save** button at the top of the page.

      Token type: `email`
      Validation Template: `email-wstrust-valid-temp`

c. Proceed to Section 4.5.4, "Adding the Custom Token to the Relying Party Profile".

4.5.4 Adding the Custom Token to the Relying Party Profile

You can either edit an existing Relying Party profile, or create a new one to issue the custom token by default, and refer to the Issuance Template and related information. Either way, configure:

- Default token to issue: email (your custom token)
- Issuance Template: `email-issuance-temp`

Prerequisites

Your Custom Token and Issuance Template must be defined.

To edit the requester profile for the custom module classes

1. From the Oracle Access Suite System Configuration tab, open the Security Token Services section.
2. In the navigation tree, open the Partner Profiles node and double-click the Relying Party Profiles node to display a list of existing profiles.
3. **Existing Profile**:
   a. In the Search Results table of the Relying Party Profiles page, click the name of the desired profile.
   b. Click the Token and Attributes tab.
   c. **Token Type Configuration**: Click the Add (+) button above the Token Type Configuration table and enter the following details:

      Token type: `email`
      Issuance Template: `email-issuance-temp`

   d. **Attributes**: Click the Add (+) button above the Attributes table and define the following:

      Attribute name: mail
      Store Type: `Userstore`
      Include in Token: `(check to enable)`
      Encryption (leave blank)
      Value (leave blank)

   e. Click Apply, dismiss the confirmation window, and close the page (or click Cancel to dismiss the page without submitting it).
4. **New Profile**: Click the New Relying Party Profile button to display the New Partner Profile page where you enter details:
   a. **General**: Fill in the following details for the custom token in this chapter and then click the Next button at the top of the page.

      Profile ID: *`unique_relyingparty-name`*
      Default Token: `email`

   b. Click the Token and Attributes tab and perform Steps 2c and 2d, then click Apply.

4.5.5 Mapping the Token to a Requestor

If you don't have a Username Validation Template (username-wss-valid-template), use the Oracle Access Suite to create one to map the token to the requester.

Validation Template Name: username-wss-valid-template

Token Type: Username

Proceed to Section 4.5.6, "Creating an /wssuser EndPoint"

4.5.6 Creating an /wssuser EndPoint

Prerequisites

"Mapping the Token to a Requestor"

To create an endpoint

1. From the Oracle Access Suite System Configuration tab, open the Security Token Services section.
2. Double-click the Endpoints node to display a list of existing Endpoints.
3. **New Endpoint**:
    a. Click the Add (+) button above the table (or choose New Endpoint from the Actions menu).
    b. Enter the new Endpoint URI: /wssuser
    c. Choose the Oracle WSM policy: sts/wss_username_service_policy
    d. Choose the Validation Template: username-wss-validation-template.
    e. Click Apply to submit the definition and dismiss the confirmation window (or click Revert to dismiss the page without submitting it).
    f. Close the page.