Oracle Fusion Middleware

Developing Applications with Oracle Access Management

14c for All Platforms
G10425-01
March 2025

ORACLE

Oracle Fusion Middleware Developing Applications with Oracle Access Management, 14c for All Platforms

G10425-01

Primary Author: Panendra Puttachar

Contributors: Vadim Lander, Vamsi Motukuru, Paresh Raote, Prakash Manwani,Venu Shastri, Sunil Joshi, Narayana Khadri, Prasanna Anupindi, Arvind Kumar Gupta

Contents

3 Developing Custom Authentication Plug-ins

4 Developing Custom Pages

List of Examples

List of Figures

List of Tables

Preface

This guide explains how to write custom applications and plug-ins to programmatically extend access management functionality using the SDKs and APIs provided with Oracle Access Management.

This Preface covers the following topics:

Audience

This document is intended for developers who are familiar with Oracle Access Management.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Fusion Middleware 14*c* Release (14.1.2.1.0) documentation set:

- *Oracle Fusion Middleware Administering Oracle Access Management*
- *Oracle Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*
- *Oracle Fusion Middleware Extensibility Java API Reference for Oracle Access Management Access Manager*
- *Oracle Fusion Middleware User Provisioning Plug-in Java API Reference for Oracle Access Management Identity Federation*
- *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference for Identity and Access Management*

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 This Guide?

This section summarizes the new features and significant changes in *Developing Applications with Oracle Access Management* 14*c* (14.1.2.1.0).

New Features in 14*c* (14.1.2.1.0)

For information about Oracle Access Management 14*c* (14.1.2.1.0), and its features, See the following topics in *Fusion Middleware Administering Oracle Access Management* :

- Updates in Documentation

Part I
Introduction

Oracle Access Management provides multiple converged services with several integrated components. It contains software development kits (SDKs) and application programming interfaces (APIs) with which you can extend functionality or develop applications to customize your environment.

This part introduces the Oracle Access Management components and provides general information about developing with the SDKs and APIs.

Part I contains the following chapter.

1

Developing with Oracle Access Management Components

Oracle Access Management provides multiple converged services with several integrated components. It contains software development kits (SDKs) and application programming interfaces (APIs) with which you can extend functionality or develop applications to customize your environment.
This chapter introduces the Oracle Access Management components.

1.1 About Access Manager

Access Manager is an enterprise level solution that centralizes critical access control services to provide an integrated solution that delivers authentication, authorization, web single sign-on, policy administration, enforcement agent management, session control, systems monitoring, reporting, logging and auditing.

In this release, you can develop your own Access Clients, custom authentication plug-ins, custom login and error pages, administer Access Manager policies programmatically, as well as enable the impersonation feature and develop a custom user interface for managing, using the provided Java Access SDK and Access Manager APIs.

For information about developing applications using Access Manager SDKs and APIs, See Developing with Access Manager.

See Also, *Understanding Oracle Access Management Access Manager* in *Administering Oracle Access Management*.

1.2 About Identity Federation

Identity Federation enables organizations to securely link accounts and identities across security boundaries without a central user repository or the need to synchronize data stores. It provides an interoperable way to implement cross domain single sign-on without the overhead of managing, maintaining, and administering their identities and credentials.

As a result of cloud, Web Services, and business-to-business transactions, federated authentication is now a core element of any Web access management solution. Beginning with this release, SAML-based federation services are not being converged directly into a single access management server. In this release, convergence is limited to Service Provider functionality. In this release any Identity Provider functionality still requires a Oracle Identity Federation installation. However, the linking of Oracle Access Management 14*c* and Oracle Identity Federation is very simple and well integrated.

In this release, you can develop a custom user provisioning plug-in if the out-of-the-box solution does not meet your needs. You can also develop a message processing plug-in. For more information on Identity Federation APIs, See Developing with Identity Federation.

See Also, Managing Oracle Access Management Identity Federation in *Fusion Middleware Administering Oracle Access Management*.

1.3 System Requirements and Certification

System requirements and certification documentation provides information about hardware and software requirements, platforms, databases, and other information.

Both, System requirements and certification documents are available on Oracle Technology Network (OTN).

The system requirements document covers information such as hardware and software requirements, minimum disk space and memory requirements, and required system libraries, packages, or patches:

http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-requirements-100147.html

The certification document covers supported installation types, platforms, operating systems, databases, JDKs, and third-party products:

http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html

Part II

Developing with Access Manager

Oracle Access Management Access Manager provides a software development kits (SDKs) and application programming interfaces (APIs) to extend functionality or develop applications to customize your access environment.

This part discusses developing applications using the Oracle Access Management Access Manager SDK and APIs.

Part 2 contains the following chapters.

2

Developing Access Clients

Oracle Access Management Access Manager (Access Manager) provides a pure Java software developer kit (SDK) and application programming interfaces (APIs) for creating custom Access Clients.
This chapter discusses how to develop a custom Access Client and provides the following sections.

2.1 About Developing Access Clients

With the Access Manager you can develop your own Access Clients, custom authentication plug-ins, custom login and error pages, administer Access Manager policies programmatically, as well as enable the impersonation feature and develop a custom user interface for managing, using the provided Java Access SDK and Access Manager APIs.

A *WebGate* is a Web server plug-in that intercepts HTTP requests for resources and forwards them to the OAM Server for authentication and authorization. A WebGate is a Web server agent that acts as the actual enforcement point for access requests. Several WebGates are provided out-of-the-box and are ready for installation on an Oracle HTTP Server, where it intercepts access requests.

An *Access Client* is a custom WebGate that has been developed using the 14*c* Access SDK and APIs. When a standard WebGate is not suitable, a custom Access Client can be written and deployed for processing requests from users or application for either Web or non-Web resources (non-HTTP).

This section provides the following topics:

2.1.1 About the Access SDK and APIs

The 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.

The 14*c* Access SDK is intended for use by Java application developers in the development of tightly coupled, performant integrations. It is provided on JDK versions that are supported across Oracle Fusion Middleware applications. In addition to this guide, for more information See *Oracle Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*.

> **Note:**
>
> Oracle strongly recommends that developers use the 14*c* Access SDK for all new development.

The following Access API are included:

- **oracle.security.am.asdk**: An authentication and authorization API that provides enhancements to take advantage of 14*c* OAM Server functionality. The 14*c* Access SDK API can be used with Oracle Access Manager 14*c* version of the server.

  > **Note:**
  >
  > The `oracle.security.am.asdk` package provides the 14*c* Java APIs. The 14*c* version is very similar to the previous release APIs, with enhancements for use with the 14*c* OAM Server. From a functional perspective, the 14*c* Access SDK maintains parity with the previous release Access SDK to ensure that you can re-write existing custom code using the 14*c* API layer.

The 14*c* Access SDK includes authentication and authorization functionality. However, it does not include Administrative APIs (for instance, there is no 14*c* Policy Manager API).

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

- Developing a custom Access Client for a Web server or an application server for which Oracle does not provide an out-of-the-box integration.
- Accessing session information that may be stored as part of the Access Manager authentication process.
- Verifying the validity of the Access Manager session cookie rather than trusting an HTTP header for the user principal.

14c Access SDK Features describes the primary features of the 14*c* Access SDK.

Table 2-1 14c Access SDK Features

Feature	Description
Installation	**Client Package**: Is comprised of a single zip file that contains oamasdk-api.jar, as well as other JPS jar files needed for 14*c* agent operations. 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 Access Manager server installation.

Table 2-1 (Cont.) 14c Access SDK Features

Feature	Description
Built In Versioning	Enables you to: • Determine the Access SDK version that is installed. • Validate compatible versions it can operate with. 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.

2.1.2 About Custom Access Clients

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 Access SDK enables development of custom integrations with Access Manager for 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 OAM Server, transparently performs dynamic configuration management, whereby any Access Client configuration changes made using the administration console are automatically reflected in the affected Access SDK runtimes.

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

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

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 2-2.

Table 2-2 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 14*c*, Java is the only development language for custom Access Clients.
Resource Type	Protect both HTTP and non-HTTP resources.
Credential Retrieval	Enable HTTP FORM-based input, the use of session tokens, and command-line input, among other methods.

After it has been written and deployed, a custom Access Client is managed by an Oracle Access Management administrator the same as a standard WebGate. For information about managing a custom Access Client using the administration console, See Resource Types and Their Use .

See Also,

- When to Create a Custom Access Client
- Types of Resources in the Access Client Architecture

2.1.2.1 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.

2.1.2.2 Types of Resources in the Access Client Architecture

Each Access Client is built from the following three types of resources:

- Custom Access Client code.

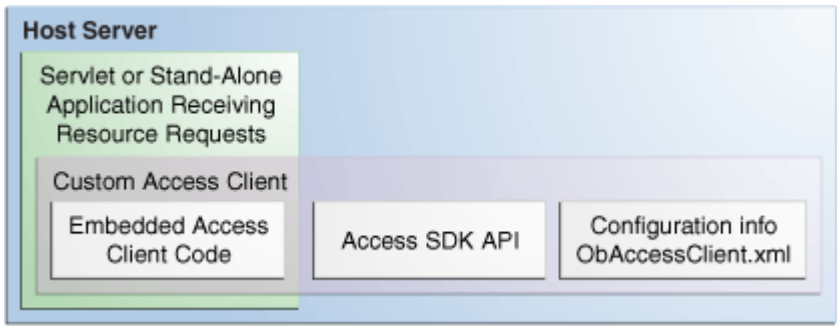
  Built into a servlet or standalone application. For the 14*c* release, you write Access Client code using the Java language platform.

- Configuration information.
  - **ObAccessClient.xml file**: Primary configuration file, which contains configuration information that constitutes an Access Client profile.
  - **cwallet.sso** and **jps-config.xml** files: For an 14*c* agent only.
  - If the transportation 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 issued for the Access Client.
    - **password.xml** – An XML file that holds the value of global pass phrase. Same password is also used to protect private key file.
- Access Manager API libraries.

  Facilitates interaction between the Access Client and OAM Server.

Figure 2-1 shows the relationship between the Access Client components installed on a host server.

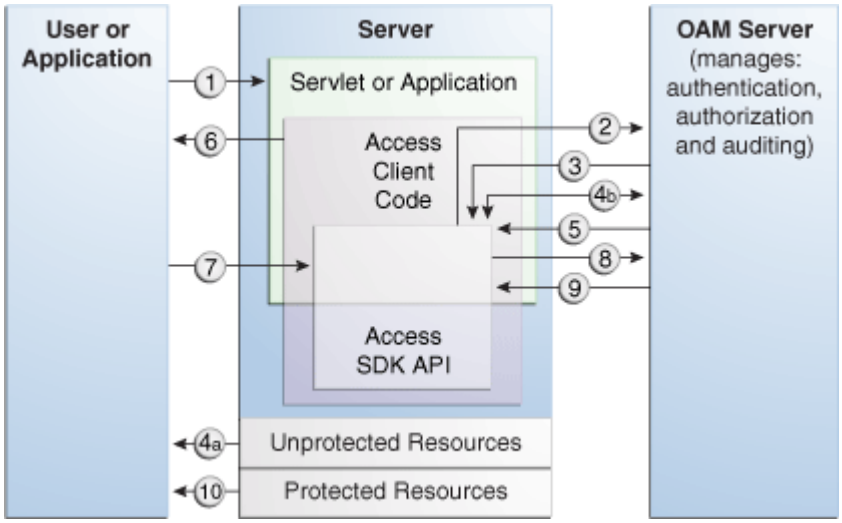
Figure 2-1 Architectural Detail of an Access Client



2.1.3 About Access Client Request Processing

Regardless of the variability introduced by the types of resources discussed in Types of Resources in the 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 Figure 2-2. Details of the process overview are explained below the figure.

Figure 2-2 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 a `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 or denies access to the resource depending on the value of the `DenyOnNotProtected` flag. Default value is true.

     For Access Manager 14*c* agent, `DenyOnNotProtected` flag in always true and cannot be changed.
   - 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.

   For an Access Client developed using the Access SDK, a SSO token is issued as a string type with no name. Use `getSessionToken()` on an existing `UserSession` object to return that session's token. If you have an existing token, it can be used to construct a user session object. The token is encrypted and opaque to a user, but internally, can be 14*c* format.

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.

The flow illustrated in Figure 2-2 represents 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 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.

When writing a custom Access Client, it is possible to authenticate users over the backchannel.

2.2 Installing Access SDK

To install the Java Access SDK Client for Access Manager 14*c*, perform the following steps:

1. Download the `oam-java-asdk.zip` file from Oracle Technology Network.
2. Extract the contents of the file `oam-java-asdk.zip` to a local directory.
3. Add `oamasdk-api.jar` to your CLASSPATH to configure the build path with the dependent jars and execute the sample asdk client
4. Execute the command:

```
java -Dopss.tenant.mode=JPS_API -cp
java -cp oamasdk-api.jar:nap-api.jar:./:mportcert.jar:opss_standalone/
modules/oracle.jps/jps-manifest.jar SampleASDK
ndalone/modules/oracle.odl/ojdl.jar:opss_standalone/modules/oracle.jps/jps-
man
ifest.jar:oracle_common/modules/oracle.igf/igf-manifest.jar:oracle_common/
modules/oracle.idm/identitystore.jar SampleASDK
```

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.

2.3 Developing Access Clients

With the Access Manager you can develop your own Access Clients, custom authentication plug-ins, custom login and error pages, administer Access Manager policies programmatically,

as well as enable the impersonation feature and develop a custom user interface for managing, using the provided Java Access SDK and Access Manager APIs.

The following topics are discussed in this section:

2.3.1 Understanding the Structure of an Access Client

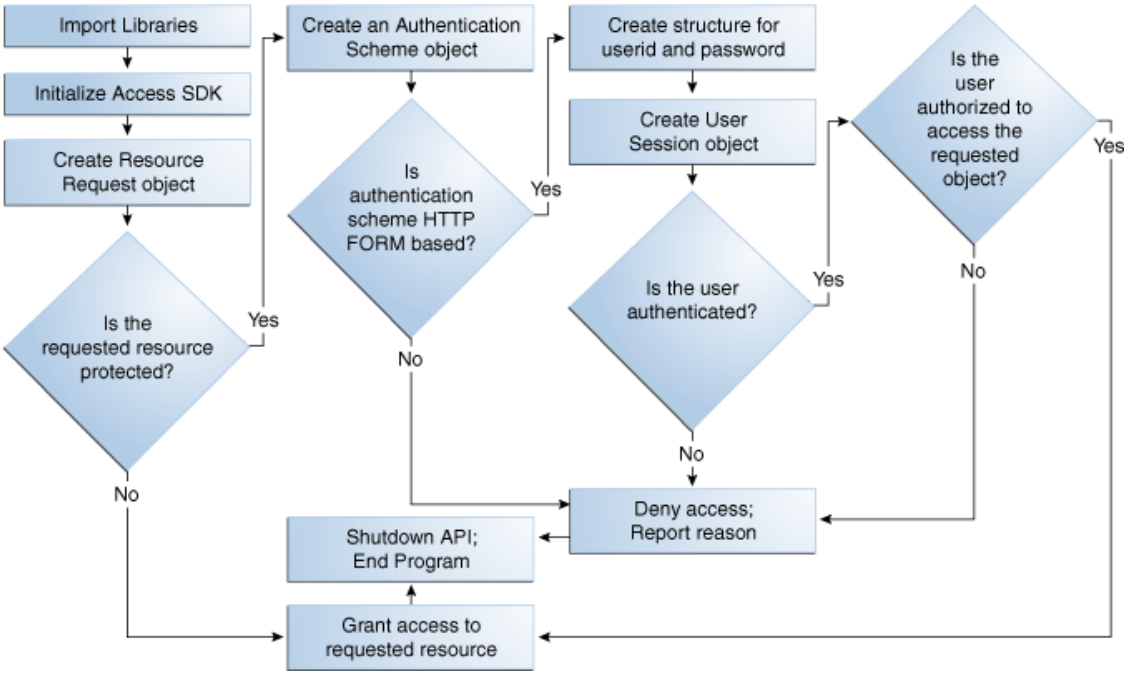
The structure of a typical Access Client application roughly mirrors the sequence of events required to set up an Access Client session.

- Include or import requisite libraries.
- Get resource.
- Get authentication scheme.
- Gather user credentials required by authentication scheme.
- Create user session.
- Check user authorization for resource.
- Clean up (Java uses automatic garbage collection).
- Shut down.

2.3.2 Understanding a Typical Access Client Execution Flow

All HTTP FORM-based Access Client applications and plug-ins follow the same basic pattern. Figure 2-3 shows a process flow for form-based applications. Details are described in the following figure.

Figure 2-3 Process Flow 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: If the resource is not protected, the Access Client grants or denies access to the resource depending on the value of the `DenyOnNotProtected` flag. Default value is `true`. For Access Manager agent, `DenyOnNotProtected` flag in always `true` and cannot be changed.
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 *Creating a New Application Domain*.

2.3.3 Sample Code: Simple Access Client

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

- 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 `JAccessClient.java` appears in Example 2-1. You can copy this code verbatim into the text file `JAccessClient.java` and execute it on the computer where your Access Manager SDK is installed.

See Annotated Sample Code: Simple Access Client for an annotated version of this example to help you become familiar with 14*c* Java Access Manager API calls.

Example 2-1 JAccessClient.java

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

public class JAccessClient {
   public static final String  _resource = "//Example.com:80/secrets/
         index.html";
   public static final String  _protocol = "http";
   public static final String  _method = "GET";
   public static final String  _login = "jsmith";
   public static final String  _passwd = "j5m1th";
   public static final String m_configLocation = "/myfolder";
   public static void main(String argv[]) {
      AccessClient ac = null;
      try {
          ac = AccessClient.createDefaultInstance(m_configLocation,
 AccessClient.CompatibilityMode.OAM_11G);

         ResourceRequest rrq = new ResourceRequest( _protocol,  _resource,
                _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",  _login);
                creds.put("password",  _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();
       }
    }
```

2.3.4 Annotated Sample Code: Simple Access Client

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 _resource = "//Example.com:80/secrets/index.html";
   public static final String _protocol = "http";
   public static final String _method = "GET";
   public static final String _login = "jsmith";
   public static final String _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 *Resources in an Application Domain*.

```
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 {
```

To initialize the Access SDK, create an `AccessClient` instance by providing the directory location of the `ObAccessClient.xml` configuration file. There are multiple ways to provide configuration location to initialize the Access SDK. For more information refer to *Oracle Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*.

The instance of `AccessClient` initializes the Access SDK API. When the AccessClient instance is created in OAM_11g mode, you must use an 14c agent profile. By default, if this compatibility mode is not provided, then default `OAM_111g` is used, and the agent will be operating in 14*c* agent mode and can only talk with 14*c* OAM Servers.

```
ac = AccessClient.createDefaultInstance(m_configLocation ,
AccessClient.CompatibilityMode.OAM_11G);
```

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

- **_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 *Creating a Custom Resource Type*.
- **_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
  ```

- **_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 *Managing Resource Types*.

```
ResourceRequest rrq = new ResourceRequest(_protocol,
   _resource, _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 `_login` and `_passwd` into the hashtable.

```
System.out.println("Form Authentication Scheme.");
Hashtable creds = new Hashtable();
creds.put("userid", _login);
creds.put("password", _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 *Credential Challenge Methods*.

```
} 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 needs 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.

2.3.5 Sample Code: Java Login Servlet

This example follows the basic pattern of API calls that define an Access Client, as described in Sample Code: Simple Access Client. 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 *Understanding SSO Cookies*.

> **Note:**
>
> This example Java servlet does not provide SSO to resources protected by Access Manager WebGates.

This sample login servlet accepts userid/password parameters from a form on a custom login page, and attempts to log the user in to 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. Annotated Sample Code: Java Login Servlet provides 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");
      } 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>");
         }
}
```

2.3.6 Annotated Sample Code: Java Login Servlet

Import standard Java packages to support input, 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 Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*.

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

```
public void init() {
      AccessClient ac =
```

```
 AccessClient.createDefaultInstance("myfolder");
   } 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.

```
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);
```

> **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.

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>");
  }
}
```

2.3.7 Sample Code: Additional Methods

Building on the basic pattern established in the sample application `JAccessClient.java`, discussed in Sample Code: Simple Access Client, the following sample invokes several additional OAM Server methods. For instance, it inspects the session object to determine which actions and 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 *Testing User Authentication from the Access Tester Console*.

The complete listing for this sample application appears in Example 2-3. An annotated version of the code is provided in Annotated Sample Code: Additional Methods.

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);

   } 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();
  }
 }
```

2.3.8 Annotated Sample Code: Additional Methods

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 Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*.

```
try {
 ac = AccessClient.createDefaultInstance(configDir ,
   AccessClient.CompatibilityMode.OAM_11G);
}
```

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");
```

If 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();
  }
}
```

2.3.9 Sample Code: 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[] encodedCert = 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<String, String> creds = new Hashtable<String, String>();

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

// Create ResourceResource request object including all information about the //
// resource being accessed including Resource type (for example http, ejb etc.
// If null, defaults to http), and operation for the resource object
// (for example GET, POST, PUT, HEAD, DELETE, TRACE, OPTIONS, CONNECT, OTHER)
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;
```

2.4 Understanding Access SDK Logs

The Access SDK uses Java logging APIs for producing logs. The `oracle.security.am.asdk` package contains the `AccessLogger` class, which produces the Access SDK log. The log generated by the Access SDK provides information about operations performed. For example, operation status, any errors or exceptions that occur, and any general information that is helpful for troubleshooting can be logged.

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

> **Note:**
>
> 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.

The following types of exceptions are used to indicate error conditions in an Access SDK log.

- **OperationNotPermittedException**

  The Access SDK provides a set of session management APIs. Only privileged Access Clients can perform these session management operations. `AllowManagementOperations` flag must be set for the specified agent profile to initialize Access SDK.

  If the Access Client is not allowed to perform these operations, the OAM Server returns an error. When the server returns an error, the Access SDK will throw this exception.

- **AccessException**

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

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
```

2.5 Building an Access Client Program

The following topics are discussed in this section:

- Setting the Development Environment
- Compiling a New Access Client Program

2.5.1 Setting the Development Environment

Setting up your development environment involves installing JDK and Access SDK software and setting appropriate environment variables. The development environment has the following requirements:

1. Install JDK 17/21.
2. Install 14*c* Access SDK.
3. 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
   ```

4. 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
   ```

5. 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
   ```

   For a list of all jar files required in the CLASSPATH variable, see Installing Access SDK.

2.5.2 Compiling a New Access Client Program

After configuring the development environment as documented in 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 CLASSPATH and Access Client program name as needed. For more information about the jar files to add to CLASSPATH, see Installing Access SDK.

2.6 Deploying Access Clients

After development, the Access Client must be deployed in a live Access Manager environment in order to test and use it. It is assumed that the Access Client program is already developed and compiled

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

1. Retrieve the Access SDK jar file and copy this to the computer you will use to build the Access Client. .

   See Installing Access SDK
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 CLASSPATH. If in a non-JRF environment, verify that the necessary JPS jar files are in the CLASSPATH.

   See, Installing Access SDK.
6. Deploy the Access Client.

   To deploy the access client See Registering an OAM Agent Using the Console in *Fusion Middleware Administering Oracle Access Management*.

2.7 Configuring Access Clients

This section describes the configuration steps required before deploying an Access Client developed using the Access SDK. The Access Client deployment process is similar to that of other Access Manager agents. This section provides the following details.

- Understanding Configuration Requirements for Access SDK
- Generating the Required Configuration Files
- SSL Certificate and Key File Requirements

2.7.1 Understanding Configuration Requirements for Access SDK

An Access SDK configuration consists of the following files:

- **ObAccessClient.xml**

  This configuration file (ObAccessClient.xml) holds various details, such as Access Manager server host, port, and other configuration items, that decide behavior of the Access Client. For example, idle session time.

An alternative to using ObAccessClient.xml is to initialize the Access SDK by providing a bootstrap configuration. An access client or application can use a bootstrap configuration from its own configuration store or other method. Configuration details such as host and port number of the OAM Server can be invoked using `AccessClient.createDefaultInstance`. For more information about programmatic initialization, See *Oracle Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*.

- **cwallet.sso**

  This Oracle wallet file is an artifact created when an 12*c*/14*c* agent is registered with Access Manager. The cwallet.sso file contains the secret key that is used by the OAM Server when encrypting a token issued for the agent.

  The cwallet.sso file can be stored in the same location as other files or elsewhere. The path must be declared in jps-config.xml and is relative to the jps-config.xml location.

  In a JRF environment, there is a system jps-config.xml located under the <DOMAIN_HOME>/config/fmwconfig directory. This file specifies the use of the system cwallet.sso located in the same directory; the system wallet contains keys and credentials for all components in the system. Because of this, you must merge your agent registration cwallet.sso with the system cwallet.sso using the following procedure:

  1. Prepare a merge-cred.xml, specifying the directory for the source cwallet.sso (agent registration artifacts) and the destination cwallet.sso (system artifacts). The file contents are like those defined in Example 2-4.
  2. Run the following WLST command to merge the wallets.

     ```
     <MW_HOME>/common/bin/wlst.sh
     wls:/offline> connect("<username>", "<password>", "<host>:<admin_port>")
     wls:/base_domain/serverConfig>
     migrateSecurityStore(type="credStore",configFile="merge-creds.xml",
      src="FileSourceContext",dst="FileDestinationContext")
     ```

  3. Run the following command to verify that the agent cwallet.sso has been successfully merged into the system cwallet.sso.

     ```
     <MW_HOME>/oracle_common/bin/orapki wallet display
          -wallet <destination cwallet.sso dir>
     ```

- **jps-config.xml**

  This file is required by the libraries used to read the cwallet.sso file. It can reside in either of the following locations:

  – default under <*current working dir*>/config/jps-config.xml (template is extracted from unzipping the client install zip file), where <*current working dir*> is the directory where the client install zip file was unzipped. Or,

  – can be specified through `-Doracle.security.jps.config=`*`jps-config.xml file location`*. You must pass the location as a property in the Java command.

  > **Note:**
  >
  > In a JRF environment, as previously stated, a system jps-config.xml file located in the <DOMAIN_HOME>/config/fmwconfig directory is used by default. There is no need to prepare another jps-config.xml.

- Java Security Grants

When Java Security Manager is enabled, you need to add additional grants for the application to the system-jazn-data.xml file in order to access credentials in the wallet. Choose one of the following based on your environment.

- In a JRF environment with deployed applications, add the following grants to the system-jazn-data.xml file.

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

- In a non-JRF environment with a standalone application, if Java Security Manager is not enabled (which is generally the case for standalone applications) no policy file is needed.
- In a non-JRF environment with deployed applications, when Java Security Manager is enabled, find the corresponding Java security policy file being used (for example, weblogic.policy for Weblogic Server) and add the following security grants to it.

```
grant codeBase "<url>"
  {
    permission
     oracle.security.jps.service.credstore.CredentialAccessPermission
     "context=SYSTEM,mapName=OAMAgent,keyName=*", "read";
  };
```

  <url> specifies the code source location for the deployed application; for example, `file:/scratch/install/WLS_HOME/user_projects/domains/base_domain/servers/AdminServer/tmp/_WL_user/ASDKServlet/-`

- **JKS Keystores for SSL**

  This file is required only if the transport security mode is Simple or Cert. Both the 11*g* OAM Server and 14*c* OAM Server supports transport security modes Open, Simple and Cert to communicate with agents. Credentials are passed using the Oracle Access Protocol (OAP). When OAP is used in Open mode the communication is vulnerable to eavesdropping, so Open mode is discouraged in production environments. Open mode is recommended in testing environments only.

  An Access Client developed using Access SDK is called an *agent*. Depending on the mode in which OAM Server is configured, an Access Client will have to be configured to communicate in the same mode.

  Each 14*c* agent has its own agent key. The 14*c* agent key is stored in cwallet.sso. This key is used to encrypt the SSO token, the `accessClientPasswd`, and the global passphrase (stored in password.xml) used in Simple or Cert transport security mode. The SSO token issued for one agent cannot be used directly for another agent, unless you obtain a scoped session token from a master token. See *Managing the Access Protocol for OAM Proxy Simple and Cert Mode Security* in *Fusion Middleware Administrator's Guide for Oracle Access Management*.

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

- oamclient-truststore.jks
- oamclient-keystore.jks
- password.xml

See Also,

Types of Resources in the Access Client Architecture.

- **password.xml**

  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.

  See Generating the Required Configuration Files.

- **Log Configuration**

  Is required in order to generate a log file. For more information, see Understanding Access SDK Logs.

Example 2-4 merge-cred.xml Sample

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
  <jpsConfig
xmlns="http://xmlns.oracle.com/oracleas/schema/11/jps-config-11_1.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/oracleas/schema/11/
  jps-config-11_1.xsd" schema-major-version="11" schema-minor-version="1">

   <serviceProviders>
    <serviceProvider
class="oracle.security.jps.internal.credstore.ssp.SspCredentialStoreProvider"
name="credstoressp" type="CREDENTIAL_STORE">
     <description>File-based credential provider</description>
    </serviceProvider>
   </serviceProviders>

   <serviceInstances>
        <!-- Source file-based credential store instance -->
    <serviceInstance location="<source cwallet.sso dir>"
      provider="credstoressp" name="credential.file.source">
    </serviceInstance>

        <!-- Destination file-based credential store instance -->
    <serviceInstance location="<destination cwallet.sso dir>"
      provider="credstoressp" name="credential.file.destination">
    </serviceInstance>
   </serviceInstances>

   <jpsContexts>
    <jpsContext name="FileSourceContext">
     <serviceInstanceRef ref="credential.file.source"/>
    </jpsContext>

    <jpsContext name="FileDestinationContext">
     <serviceInstanceRef ref="credential.file.destination"/>
    </jpsContext>
   </jpsContexts>
</jpsConfig>
```

2.7.2 Generating the Required Configuration Files

The ObAccessClient.xml configuration file can be obtained by registering an Access Client as 12*c*/14*c* agent with the OAM 14*c* Server, using the administration console or a remote registration tool. When registering 12*c*/14*c* agents the cwallet.sso file is also created. For more information, See *Introduction to Agent Registration* in *Fusion Middleware Administrator's Guide for Oracle Access Management*.

The Oracle Access Management 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 Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*.

2.7.3 SSL Certificate and Key File Requirements

The 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. The following sections have more information.

- About Simple Transport Security Mode
- Working in the Cert Transport Security Mode

2.7.3.1 About Simple Transport Security Mode

In Simple transport mode, the JKS keystores are auto-generated by the OAM Server. The generated keystores are located in *WLS_OAM_DOMAIN_HOME*/output/webgate-ssl/.

2.7.3.2 Working in the Cert Transport Security Mode

In Cert transport security mode, the certificates for the server and agent should be requested from a certifying authority. 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. Instead of cacert.pem or cacert.der, substitute the CA certificate file of the issuing authority.
2. If an earlier version of 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 *Generating a Certificate Request and Private Key for OAM Server*in *Fusion Middleware Administrator's Guide for Oracle Access Management*.

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

2.7.3.2.1 Importing the CA Certificate

This step is not required when using the 14*c* Java Access SDK.

The CA certificate must be imported to the trust store when using earlier versions of JNI SDK. The Access SDK provides a self-signed CA certificate that can used for issuing certificates to the Access Client. 14*c* OAM Server provides a self-signed CA certificate.

- **OAM 14c Server**: The CA certificate (cacert.der) is located in $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 cacert.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.

2.7.3.2.2 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 so the Access Client can communicate with OAM Server.

- **OAM Server**: Use the tool Remote Registration and administration console for generating a certificate file (aaa_cert.pem) and key file (aaa_key.pem) in PEM format for the Access Client.

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.

2.8 Best Practices

This section presents a number of ways to avoid problems and to resolve the most common problems that occur during development. The following topics are discussed in this section:

- Avoiding Problems with Custom Access Clients
- Identifying and Resolving Access Client Problems
- Environment Problems using 11g Java Access SDK with Containers
- Tuning for High Load Environment

2.8.1 Avoiding Problems with Custom Access Clients

Here are some suggestions for avoiding problems with custom Access Clients.

- 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 Management Administration Console. For details, see Registering and Managing OAM Agents in *Fusion Middleware Administering Oracle Access Management*.
- To ensure clean connect and disconnect from the OAM Server, use the `initialize` and `shutdown` methods in the `AccessClient` class.
- The OBACCESS_INSTALL_DIR environment variable *must* be set on your Windows or UNIX-type 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.
- 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 for developing thread-safe code:
  - 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.

2.8.2 Identifying and Resolving Access Client 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.
- Read the Release Notes.
- 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 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 *Validating Connectivity and Policies Using the Access Tester*.

2.8.3 Environment Problems using Java Access SDK with Containers

This section provides information about resolving environment conflicts that can develop when using the Java Access SDK. It contains information regarding the following containers.

- Resolving Environment Problems with Java EE Containers
- Resolving Environment Problems with Oracle WebLogic Server
- Resolving Environment Problems with Other Application Servers

2.8.3.1 Resolving Environment Problems with Java EE Containers

Use this procedure to resolve Java class version conflicts when a web application is using the Access SDK.

A conflict can occur when a version of the library, different from the one used by the Access SDK is loaded by another application hosted on the same Java EE container. The following is a sample error message that may display:

```
oracle/security/am/common/aaaclient/ObAAAServiceClient.&lt;init&gt;(Ljava/lang
/String;[CILjava/lang/String;Ljava/lang/String;[C[CZIJJLjava/lang/Integer;Ljava/u
til/List;Ljava/util/List;)V
at oracle.security.am.asdk.AccessClient.createClient(AccessClient.java:798)
at oracle.security.am.asdk.AccessClient.initialize(AccessClient.java:610)
at oracle.security.am.asdk.AccessClient.&lt;init&gt;(AccessClient.java:527)
at
oracle.security.am.asdk.AccessClient.createDefaultInstance(AccessClient.java:234)
at com.newco.authenticateIdentity.AuthenticateIdentityAccessClient.authenticateUser(
AuthenticateIdentityAccessClient.java:52)
```

This issue is related to how classes are loaded into the Java EE container. For more information, see your container's documentation discussing class loading.

To solve this problem, configure class loader filtering for the web application that needs a specific library version. For more information and steps, see the documentation for your application server.

2.8.3.2 Resolving Environment Problems with Oracle WebLogic Server

Use WebLogic Server `FilteringClassLoader` to specify packages that are always loaded from the application, rather than loaded using the system class loader.

To resolve this issue, perform these steps:

1. Verify the weblogic.xml file exists in the META-INF folder of your application. If it does not, create this file and add the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-application/1.0/we
blogic-application.xsd"
xmlns="http://www.bea.com/ns/weblogic/weblogic-application">

<prefer-application-packages>
  <?xml version="1.0" encoding="UTF-8"?>
<weblogic-application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-application/1.0/we
blogic-application.xsd"
xmlns="http://www.bea.com/ns/weblogic/weblogic-application">

<prefer-application-packages>
<package-name>Package to be loaded</package-name>
<package-name>Package to be loaded</package-name>
</prefer-application-packages>
</weblogic-application>
```

   where *Package to be loaded* is the corresponding package from the log file. For example, assume the problem is `ObAAAServiceClient`, then the corresponding package name is *oracle.security.am.common.aaaclient*. Add as follows:

```
<package-name>oracle.security.am.common.aaaclient.*<package-name>
```

   All classes associated with this package will be loaded by the application loader, even if identical classes having a different version are specified in the CLASSPATH of the System class loader.

2. Stop the application.
3. Delete the previously deployed version of the application.
4. Install the application.
5. Access the resource.

   The error should be gone and the application is running smoothly.

2.8.3.3 Resolving Environment Problems with Other Application Servers

All application servers have a configuration file where class loading related options are configured. In general, the key is to identify the configuration file and tags that are required to enable a specific class loader to load a set of classes.

1. Locate the configuration file for the application server.
2. Use the application class loader to prevent classes from being loaded by the parent class loader, even if they are specified in the CLASSPATH.
3. Change the default class loading behavior so the parent class loader is called only if the current class loader fails to load the class.
4. Alternately, as in WebLogic Server, there may be a method that enables loading of classes using the designated class loader.
5. In some application servers, you may need to define a separate domain for your application, for a parent domain, and set class loading behavior to load the parent last.

2.8.4 Tuning for High Load Environment

In a high load, high stress environment, the Access SDK configuration must be tuned as follows:

- Configure `poolTimeout` as a user defined parameter. You must increase the number of clients for `poolTimeout`.
- Tune the maximum (max) number of connections. For high performance, the max number of connections of primary server should be in the agent profile.

3

Developing 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 to 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 the custom plug-ins created by your own Java developers.
This chapter provides the following sections regarding authentication plug-ins.

- Introduction to Authentication Plug-ins
- Introduction to Multi-Step Authentication Framework
- Introduction to Plug-in Interfaces
- Sample Code: Custom Database User Authentication Plug-in
- Developing an Authentication Plug-in

See Also, Deploying and Managing Individual Plug-ins for Authentication in *Fusion Middleware Administering Oracle Access Management*.

3.1 Introduction to Authentication Plug-ins

The release provides authentication modules for immediate use out-of-the-box as well as the following:

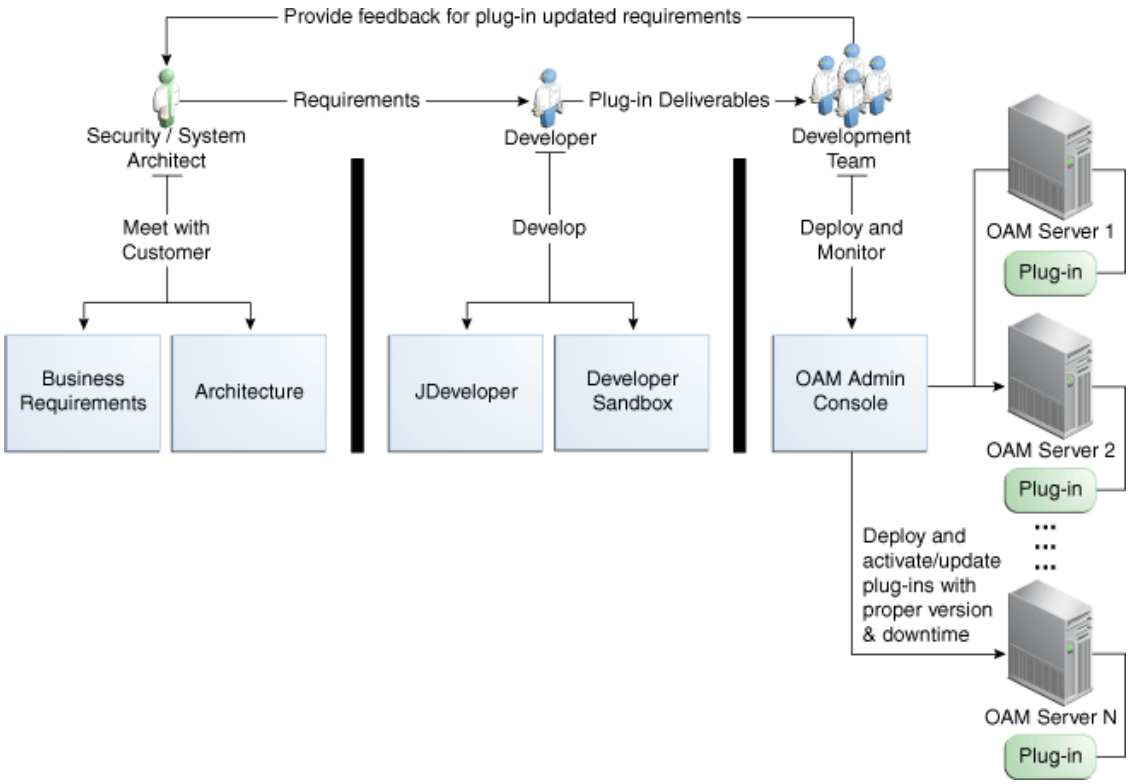
- Authentication plug-in interfaces and SDK tooling to build customized authentication modules (plug-ins) to bridge the out-of-the-box features with individual requirements. The new interfaces and SDK tooling:
  - Provide backward compatibility to support custom Oracle Access Manager previous release plug-ins.
  - Include a deterministic method to orchestrate custom plug-ins within an authentication module.
- A mechanism that enables quick deployment of customized authentication plug-ins.
- A mechanism to maintain the complete plug-in State lifecycle.

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

See Also, About the Plug-in Interfaces

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.

1. **Planning**:

   Identify the business requirements for this plug-in and consider the authentication flow when a user requests a resource, as described in About Planning, the Authentication Model, and Plug-ins.

   The security architect knows how Access Manager 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 file.

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

3. **Deployment**:

   Oracle Access Management administrators deploy and orchestrate multiple plug-ins to work together in an authentication module and also tests and monitors plug-ins. Common deployment tasks include the following:

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. Testing the plug-in using the Access Tester as described in *Testing User Authentication from the Access Tester Console*

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

For information about deploying authentication plug-ins using the Oracle Access Management Administration Console, See *Deploying and Managing Individual Plug-ins for Authentication* in *Fusion Middleware Administering Oracle Access Management*.

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 typical plug-in life cycle is as follows:

- 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 Access Manager and use it without restarting servers
  - Distribute: Propagate the plug-in jar file from one local OAM Server 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 OAM Server
- 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 file is available, the deployer can import it to a Weblogic Server DOMAIN_HOME/oam/plugins from the administration console's Import action.

Table 3-1 describes the states of a plug-in life cycle that are controlled by Oracle Access Management administrators. For more information, See *Deploying and Managing Individual Plug-ins for Authentication* in *Fusion Middleware Administering Oracle Access Management*.

Table 3-1 Plug-in Life Cycle States

State	Description
Import	Adds the plug-in JAR file to an Weblogic Server 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 Weblogic Server, 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 to add the following functionality. This is not a complete listing; other types of plug-ins are supported.

- User Identity Mapping

  Plug-ins can add functionality to handle 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

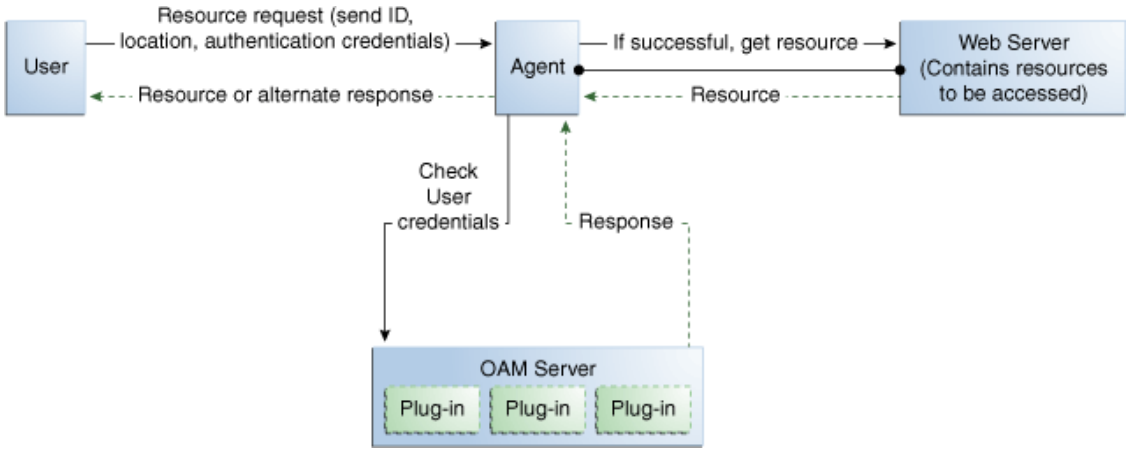
  Responses (not provided out-of-the-box) might be needed when authenticating the user. Custom plug-ins can fulfill this need.

- Custom Responses

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

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 dotted line arrows are custom responses generated by plug-ins that are deployed on the OAM Server.

Figure 3-2 Authentication Model and Plug-ins



Before designing and developing custom authentication plug-ins, Oracle recommends that developers analyze the 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 handle it. One is to have specific schemes 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 second option is to have every scheme be hard-coded to handle the 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). Each request approach has its own advantages and disadvantages as documented in Table 3-2.

Table 3-2 Request 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.

Suppose a user wants to log in to his online bank account using his home computer, at midnight. The differences between the two approaches are simple but important and developers must decide which approach best meets their requirements. The following process overviews outline the differences between the decision engine approach and the hard-coded approach.

- About the Decision Engine Approach Process
- About the Hard-Coded Approach Process

3.1.2.1 About the Decision Engine Approach Process

In the decision engine approach to handling authorization requests:

1. The request comes from the user with a certain IP address at midnight.
2. The decision engine determines it has previously handled 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.

3.1.2.2 About the Hard-Coded Approach Process

In the hard-coded approach to handling authorization requests:

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.

3.2 Introduction to Multi-Step Authentication Framework

This section provides the following topics:

- About the Multi-Step Framework
- Process Overview: Multi-Step Authentication
- About the PAUSE State
- Information types shared with the credential collector page

3.2.1 About the Multi-Step Framework

The Multi-Step Authentication Framework requires a custom authentication plug-in to transmit information to the backend authentication scheme several times during the login process. All information collected by the plug-in and saved in the context will be available to the plug-in through the authentication process. Context data also can be used to set cookies or headers in the login page.

Events are the building blocks of the authentication flow. Events are created using exposed methods of the authentication module plug-in implementation. These events can be combined with the rules to build a deterministic workflow for the authentication. The Workflow controller is the module responsible for orchestrating the authentication workflow. Workflow configuration is defined in the Workflow definition language.

Multi-Factor Authentication is a business term that refers to the collection of multiple credentials necessary to authenticate a user. The Multi-Step Authentication Framework can implement Multi-Factor Authentication requirements. It can also implement Single Factor Authentication requirements using multiple steps as necessary. For example, the username and password can be collected on separate pages. Multi-step authentication relies on:

- WebGate using a credential collector (DCC or ECC) for dynamic credential collection with multi-step authentication flows. This enables greater flexibility for interactions with users or programmatic entities when collecting authentication-related information that involves several methods to establish the identity of the user.
- Authentication module chaining, where modules of a similar challenge mechanism are grouped and the credentials are collected in one pass, then validated against each module. You can chain multiple authentication modules in a new authentication scheme, and define a new scheme plug-in containing the flows.

  The challenge mechanism defines how to collect the credentials. The following mechanisms are available: FORM, BASIC, X509, WNA, OAM, TAP, and NONE. The challenge mechanism controls the way in which the required credentials are collected. Currently, this is tied to the authentication scheme.

> **Note:**
>
> About WebGate Configured as a Detached Credential Collector in *Fusion Middleware Administering Oracle Access Management*

3.2.2 Process Overview: Multi-Step Authentication

1. **Process Request**: The Master Controller processes the authentication request and passes it to the plug-in.
2. **Process Event**: The authentication scheme is executed and the plug-in determines whether any input is needed to continue the authentication. If input is required, the plug-in returns an execution status of `PAUSE` which suspends the event flow.

   `PAUSE` indicates that the authentication processing cannot proceed until additional information is obtained from user. As such, redirection is allowed. When the requested information is supplied, processing continues from the point it was paused. The request is updated with details of the associated `ACTION` that must be performed. The `ActionContext` has all the information to execute the `ACTION`.

   For example, if `PAUSE` is associated with `CREDCOLLECT_ACTION`, the Master Controller saves the plug-in execution state and begins executing events corresponding to the `ACTION` by mapping this `CREDCOLLECT_ACTION` to the `CRED_COLLECT` event and proceeding with collection as specified by the plug-in's `CredentialParameter` object.
3. The saved plug-in state is revived and plug-in execution resumes until either a state of `SUCESS` or `FAILURE` is reached. `FAILURE` indicates that the authentication attempt has failed. If so, OAM Server will take attempt to reauthenticate the user once again. For example, the user is presented with a login form.
   - If a valid subject is available, a session is created for the user, which is used to save the execution state. Otherwise, the execution state is stored in the request object. This session has the lowest Authentication Level (configured through global (Common) System Configuration).

- When user authentication is finished, the session is updated to a fully valid session with the authentication level defined in the authentication scheme and the session timeout configured for the OAM Server.

4. When the events in the dynamic flow controller finish executing, control is merged back to the parent controller and the execution state is updated.
5. When authentication completes, access is granted to the requested resource.

3.2.3 About the PAUSE State

In multi-step authentication mode, the plug-in can either collect the credentials from start or use the credentials obtained from the default login page and collect extra credentials if required. If the challenge parameter `initial_command=NONE` is set in the authentication scheme, control comes to the plug-in directly and the plug-in controls the credentials to be collected.

The plug-in can employ the `PAUSE` status to pass the `UserAction` parameter for user interaction to collect credentials. All the credentials required by the module can be collected in one or more passes to the client. During a `PAUSE` execution, the plug-in execution state and the context data will be saved. Once control returns back to the plug-in, the paused execution resumes and all the collected data is available to the plug-in.

When the plug-in is set to a `PAUSE` state, the plug-in can:

- Specify the data to be collected
- Specify the URL to redirect or forward to
- Specify the query string, if any

3.2.4 Information types shared with the credential collector page

The following types of information can be conveyed to the credential collector page.

- UserContextData
- UserActionContext
- UserAction
- UserActionMetaData

3.2.4.1 UserContextData

- `UserContextData` specifies metadata: name, display name and type of parameter to be collected by the login page. For example, to collect a user name from the login application:

  ```
  final UserContextData userNameContext = new UserContextData(form_username,
  form_username, new CredentialMetaData(PluginConstants.TEXT));
  ```

  where name of the attribute is `form_username`.

- `UserContextData` specifies the login page URL to direct a user to for collecting credentials. `CredentialMetaData` with `URL` type specifies the login page URL. For example:

  ```
  final UserContextData urlContext = new UserContextData (loginPageURL, new
  CredentialMetaData("URL"))
  ```

  where `loginPageURL` specifies the URL to be directed to.

- `UserContextData` is used to pass query parameters to the login page URL. `CredentialMetaData` with `QUERY_STRING` type specifies the query parameters to be sent with the `loginPageURL`. This can be processed by the login page. For example:

```
String queryString = "queryParam1=testParameter";
       final UserContextData queryStringContext =new UserContextData
 (queryString, new CredentialMetaData("QUERY_STRING"));
```

3.2.4.2 UserActionContext

`UserActionContext` holds the `UserContextData` metadata collected from the login page.

3.2.4.3 UserAction

`UserAction` class is used to collect the credentials. The action forwards or redirects (based on the `UserActionMetaData` parameter) to the login page to collect more credentials.

The following example shows how the classes can be used to specify information to the login page:

```
//create a user name context data.
   UserContextData userNameContext =
      new UserContextData("form_username", "form_username",
                              new CredentialMetaData(PluginConstants.TEXT));
   //create a password context data
   // Any form parameter containing the words  "password", "passcode" and "_pin"  will
be treated as sensitive values for debug logging

   UserContextData passwordContext =
      new UserContextData("form_password", "form_password",
                     new CredentialMetaData(PluginConstants.PASSWORD));

   // create URl context data for login page
   UserContextData urlContext = new UserContextData (loginPageURL,
            new CredentialMetaData ("URL"));

   UserActionContext actionContext = new UserActionContext ();

   //add the UserContextData to the CredentialActionContext
   actionContext.getContextData().add(userNameContext);
   actionContext.getContextData().add(passwordContext);
   actionContext.getContextData().add(urlContext);

   //specify if we FORWARD or REDIRECT with a GET/POST to the login page
   UserActionMetaData userAction = UserActionMetaData. FORWARD;

   // create a UserAction object and set it to the authentication context.
   UserAction action = new UserAction (actionContext, userAction);
   authContext.setAction(action);
```

3.2.4.4 UserActionMetaData

`UserActionMetaData` specifies the action type to be used with `UserAction`. The `UserAction` performs a forward or a redirect (with a `GET` or `POST`) to the login page based on the `UserActionMetaData` value. Possible values for `UserActionMetaData` are: `FORWARD`, `REDIRECT_GET`, and `REDIRECT_POST`.

3.3 Introduction to Plug-in Interfaces

This section provides the following topics:

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

3.3.1 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. The new plug-in name must be included in the manifest file. A period (.) is not a valid character in the plug-in name.

This OAM release provides a generic plug-in interface and a more specific authentication interface as described in the following topics:

- About the GenericPluginService
- About the AuthnPluginService

3.3.1.1 About the 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, and start and stop the plug-in.

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 Manager plug-ins. This provides base implementations for plug-ins start and stop methods

> **See Also:**
>
> *Oracle Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*

3.3.1.2 About the 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.

3.3.2 About Plug-in Hierarchies

This topic provides a look at the hierarchies:

- Figure 3-3
- Figure 3-4
- Figure 3-5
- Figure 3-6
- Figure 3-7

> **See Also:**
>
> *Oracle Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*

Figure 3-3 Plug-in Package Hierarchy

Figure 3-4 Plug-in Class Hierarchy

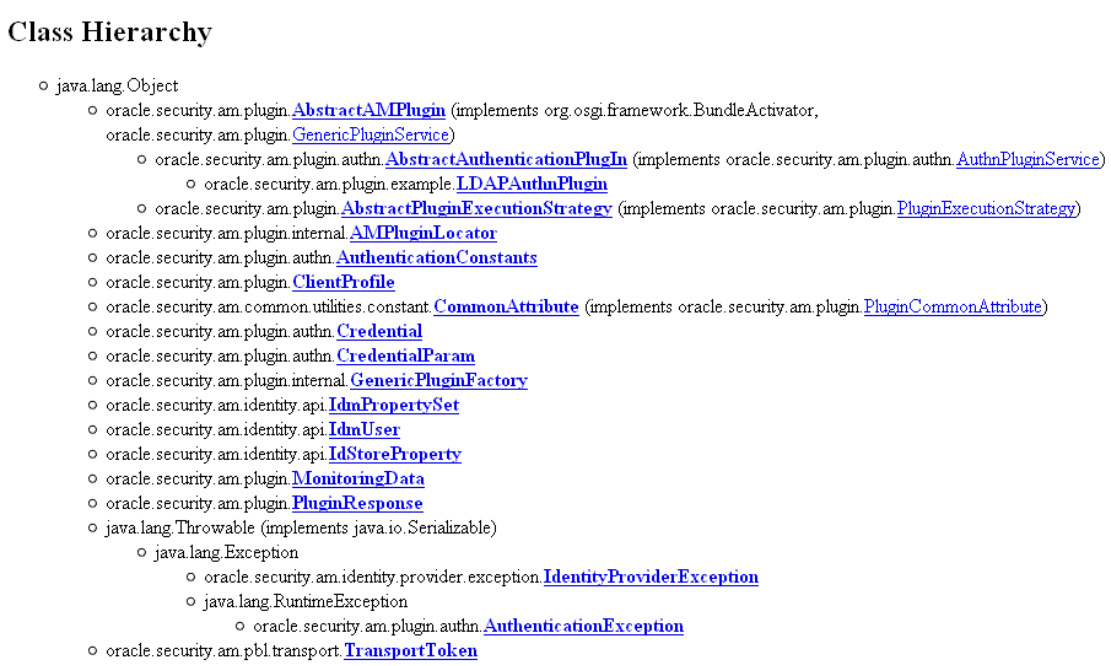
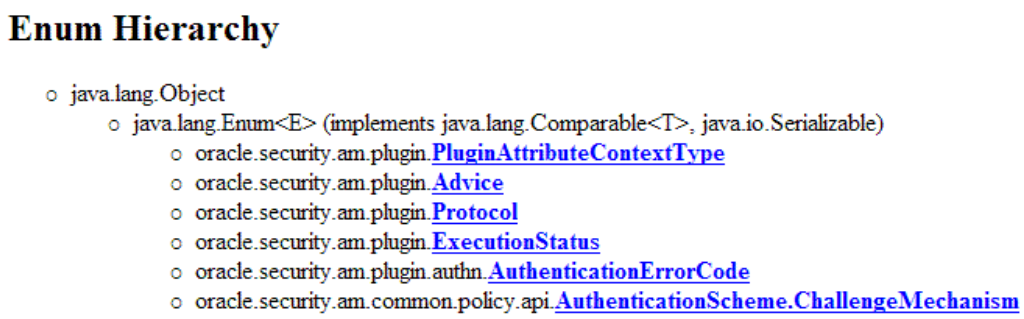


Figure 3-5 Plug-in Interface Hierarchy

Figure 3-6 Plug-in Annotation Type Hierarchy

Figure 3-7 Plug-in Enum Hierarchy



3.4 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 File for the Plug-in
- Understanding the Plug-in JAR File Structure

3.4.1 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
- Figure 3-9
- Figure 3-10

> **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;
  }
```

3.4.2 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>
```

Example 3-1 shows the XML metadata for the sample: Database User Authentication Plug-in.

Example 3-1 XML Metadata: Database User Authentication Plug-in

```
<Plugin type="Authentication">
<author>uid=User1</author>
<email>User1@example.com</email>
<creationDate>09:32:20, 2010-12-02</creationDate>
<description>Custom User Authentication Plugin Validation Against Domain Name</
description>
<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>
```

3.4.3 Sample Manifest File for the Plug-in

Beginning with the 11.1.2 release, the plug-in manifest file contains the following information:

- The plug-in version is taken from the `Bundle-Version` field. This field must be an integer
- The `name=attribute` parameter, which used to be read from the XML file in earlier releases, is now read from the `Bundle-SymbolicName` or the `Bundle-Name` field. This parameter does not need to be in the XML file.
- The `implementation` parameter, which used to be read from the XML file in earlier releases, is now read from the `Bundle-Activator` field. This parameter does not need to be in the XML file.
- The following can be removed from the XML file: `<interface>oracle.security.am.plugin.authn.AbstractAuthenticationPlugIn</interface>`.

Example 3-2 Sample Manifest File

```
Manifest-Version: 1.0
Bundle-Version: 10-->Note this to be an integer.
Bundle-Name: MFASamplePlugin
Bundle-Activator: mfasampleplugin.MFASamplePlugin
Bundle-ManifestVersion: 2
Import-Package:
 org.osgi.framework;version="1.3.0",oracle.security.am.plugin,oracle.security.a
 m.plugin.authn,oracle.security.am.plugin.impl,oracle.security.am.plugin.api,or
 acle.security.am.common.utilities.principal,oracle.security.idm,javax.security
 .auth
Bundle-SymbolicName: MFASamplePlugin
.
A corresponding sample modified XML file is :
<Plugin type="Authentication">
<author>uid=User2</author>
<email>User2@example.com</email>
<creationDate>09:32:20 2010-12-02</creationDate>
<description>Custom MFA Sample Auth Plugin</description>
<configuration>
<!-- Attribute "actiontype" indicates if the plugin wants to REDIRECT or
FORWARD to the login page to collect credentials-->
<AttributeValuePair>
<Attribute type="string" length="20">actiontype</Attribute>
<mandatory>false</mandatory>
<instanceOverride>false</instanceOverride>
<globalUIOverride>true</globalUIOverride>
<value>FORWARD</value>
</AttributeValuePair>
.
</configuration>
.
</Plugin>
```

3.4.4 Understanding the 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 Introduction to Plug-in Interfaces)

- META-INF (MANIFEST.MF)

3.4.5 Error Codes Supported by Plugin Framework

The plugin framework (`oracle.security.am.plugin.authn.AuthenticationErrorCode`) exposes authentication error codes.

Table 3-3 Error Codes supported by `oracle.security.am.plugin.authn.AuthenticationErrorCode` plugin framework

Error Codes	Error Messages
OAM-1001	INVALID_USER
OAM-1002	AUTHENTICATION_FAILED
OAM-1003	INVALID_PASSWORD
OAM-1004	INTERNAL_ERROR
OAM-1005	UNSPECIFIED_ERROR
OAM-1006	USER_LOCKED_ERROR / DISABLED_USER
OAM-1007	USER_PASSWORD_EXPIRED
OAM-1009	IDSTORE_UNREACHABLE_ERROR
OAM-10015	INVALID_TENANT_NAME
OAM-10016	INVALID_OPERATION
OAM-10017	INVALID_REMEMBER_ME_TOKEN / CHALLENGES_NOT_DEFINED
OAM-10018	USER_PWD_CANNOT_CHANGE
OAM-10019	INVALID_CHALLENGE_RESPONSE
OAM-10020	CHALLENGES_DISABLED
OAM-10021	PASSWORD_MUST_CHANGE_WARNING
OAM-10022	PASSWORD_EXPIRE_WARNING

These error codes can be used by other custom plugins to perform any appropriate customized actions.

3.5 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.

The following topics are discussed:

- About Writing a Custom Authentication Plug-in
- Writing a Custom Authentication Plug-in
- Error Codes in an Authentication Plug-In
- JAR Files Required for Compiling a Custom Authentication Plug-in

3.5.1 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 About the Plug-in Interfaces)
- Implement `initialize` method

- Implement `process` method

Table 3-4 describes the methods required for the plug-in's functionality.

Table 3-4 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.

The following tips will help in developing custom plug-ins.

- An external JAR is required in both the Weblogic class path and inside the plug-in as there is no visibility of the external JAR inside the plug-in.
- The plug-in does not use the Weblogic class path and must have its own class path for the external JAR defined in the manifest file; for example, Bundle-ClassPath:

```
,jndi-1.2.1.jar,ldap.jar,providerutil.jar,oaam_soap_client.jar,oaam_core.jar,oaam_uio
.jar
```

- The package name must be individually specified in the manifest file. Wildcards are not supported, and even nested packages must be specified in the Import-Package: section. For example:

```
javax.naming;resolution:=optional,javax.naming.spi;resolution:=optional
```

- To avoid bundle constraint exceptions during plug-in activation, put `;resolution:=optional` in the packages.
- If the JAR file is not present inside the plug-in AND its classpath (even though available in the Weblogic class path) it will throw a classnotfound exception.
- If the JAR file is not available in the Weblogic classpath, a type cast exception is thrown.

3.5.2 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 About Planning, the Authentication Model, and Plug-ins.

1. Extend `AbstractAuthenticationPlugIn` class and implement the following methods (see also About Writing a Custom Authentication Plug-in):
   - Implement `initialize` method
   - Implement `process` method
2. Develop plug-in code using appropriate Access Manager interfaces and packages. See:
   - Introduction to Authentication Plug-ins
   - Sample Code: Custom Database User Authentication Plug-in
3. Prepare Metadata for the Custom Plug-in. See:
   - Sample Plug-in Configuration Metadata Requirements
4. Prepare the Plug-in Jar file and manifest and turn these over to your deployment team. See:
   - Sample Manifest File for the Plug-in
   - Understanding the Plug-in JAR File Structure
5. Proceed to:
   - JAR Files Required for Compiling a Custom Authentication Plug-in
   - For information about deploying and managing custom authentication plug-ins, see *Deploying and Managing Individual Plug-ins for Authentication*.

3.5.3 Error Codes in an Authentication Plug-In

In the case where a plug-in needs to exchange data to the login page, error page, or client application pages, this data can be sent as `PluginResponses`. The response is in the format as a `Name=Value` pair that provides details about the data. The OAM Server sends these responses to the custom page as HTTP request parameters. The following response types facilitate the exchange:

- `CLIENT`: Enables the plug-in to communicate data about the authentication process or about the user to the client application. The request parameter is `PLUGIN_CLIENT_RESPONSE`.
- `ERROR`: Enables a plug-in to communicate any error about the authentication process. The request parameter is `PLUGIN_ERROR_RESPONSE`.

Example 3-3 Error Code in a Custom Authentication Plug-in

```
//Setting responses
PluginResponse rsp = new PluginResponse();
rsp = new PluginResponse();
rsp.setName("PluginClientCode");
rsp.setType(PluginAttributeContextType.CLIENT);
rsp.setValue("Err-100");
context.addResponse(rsp);
rsp = new PluginResponse();
```

```
rsp.setName("PluginErrorCode");
rsp.setType(PluginAttributeContextType.ERROR);
rsp.setValue("Card Expired");
context.addResponse(rsp);

String errorResponse = request.getParameter(GenericConstants.PLUGIN_ERROR_RESPONSE);
String clientResponse = request.getParameter(GenericConstants.PLUGIN_CLIENT_RESPONSE);
```

3.5.4 JAR Files Required for Compiling a Custom Authentication Plug-in

Several JAR files are required to compile a custom authentication plug-in:

- felix.jar
- oam-plugin.jar
- utilities.jar
- identity-provider.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

4

Developing Custom Pages

Oracle Access Manager provides default login, logout, error and password pages and an extensible framework for creating a custom page tailored to the look and feel of your company's brand.
This chapter explains how to develop custom pages and how to deploy them in your environment. It provides the following sections.

4.1 About the Custom Pages Framework

Access Manager provides a framework that includes a set of static HTML pages displayed to the user. These default pages can be customized to reflect your company's look and feel, or replaced entirely with new pages. You can create custom interactive pages for authentication during login, logout, and error conditioning processing.

The custom pages framework is generally dynamic which requires that it be implemented as a script or an application that can perform the required logic. Thus, you can design, implement, and deploy a custom HTML page that, for example, displays a different version of the login form depending on whether the user is accessing via a mobile browser or a desktop browser.

Login, logout, error and password pages are packaged as part of the custom pages framework. An out-of-the-box Web application archive (WAR) file is provided that can be used as a starting point to develop customized pages. This WAR is named oamcustompages.war and located in the *MW_HOME*`/oam/server/tools/custompages/` directory. After creating a custom HTML page, add it to the WAR.

> **Note:**
>
> A custom page can be used in combination with existing Access Manager authentication modules or a custom authentication plug-in. For information about developing a plug-in, See Understanding Form-Based Login Page authentication.

Any user facing custom pages must return the OAM_REQ token and the authentication endpoint.

See Also,

- Returning the OAM_REQ Token

- Returning the End Point

4.1.1 Returning the OAM_REQ Token

OAM_REQ is a transient cookie that is set or cleared by Access Manager if the authentication request context cookie is enabled. This cookie is protected with keys known to Access Manager only. OAM_REQ must be retrieved from the query string and sent back as a hidden form variable.

When a resource is requested, the OAM Server redirects or forwards it to the credential collector page to collect credentials. The OAM Server also sends an OAM_REQ token to the login page. In the case of a customized login application, the login application should ensure that the OAM_REQ token is retrieved from the request and posted back to the OAM Server along with the credentials. OAM_REQ must be retrieved from the query string and sent back as a hidden form variable. For example:

```
String reqToken = request.getParameter(GenericConstants.AM_REQUEST_TOKEN_IDENTIFIER);

<%
if(reqToken != null && reqToken.length() > 0) { %>
<input type="hidden" name="<%=GenericConstants.AM_REQUEST_TOKEN_IDENTIFIER%>"
value="<%=reqToken%>">
<%
}
%>
```

See Also, *Understanding SSO Cookies* in *Fusion Middleware Administering Oracle Access Management*.

4.1.2 Returning the End Point

The end point, `/oam/server/auth_cred_submit`, must be returned to the OAM Server. For example:

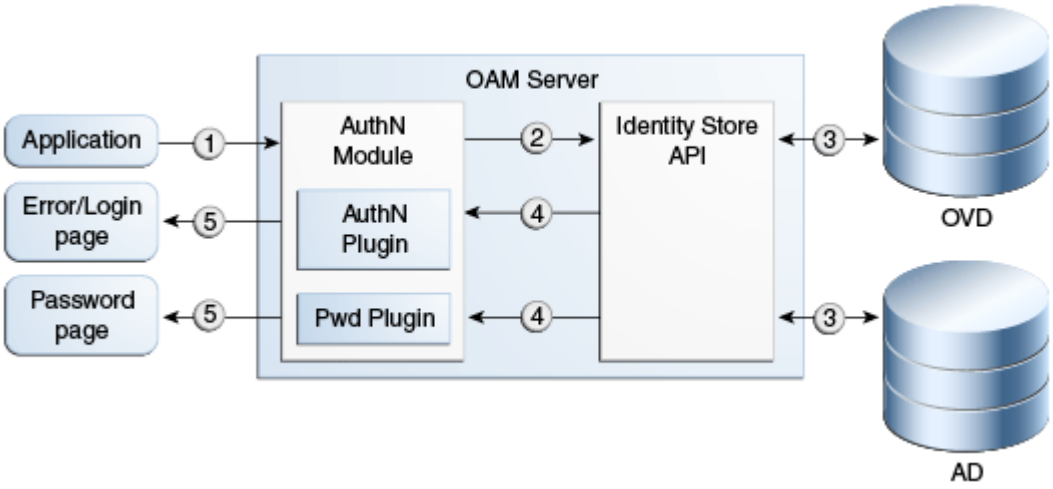
```
<form action="/oam/server/auth_cred
_submit"> or "http://oamserverhost:port/oam/server/auth
_cred_submit".
```

4.2 Authenticating with Custom Pages

The authentication process involves determining what credentials a user must supply when requesting access to a resource, gathering the credentials over HTTP, and returning an HTTP response that reflects the results of credential validation.

Figure 4-1 shows the end-to-end request flow for authenticating a user accessing an Access Manager protected resource that results in an error page.

Figure 4-1 Authentication Request Flow



In this process, the resource is protected by a specific authentication scheme that uses the authentication (AuthN) plug-in. The AuthN plug-in uses the Identity Store API to authenticate the credentials. The AuthN plug-in can also call a third-party API (not shown).

1. When a user requests a protected resource, the authentication flow is triggered and a login page is displayed. The user enters the required credentials which are then submitted to the authentication engine.
2. The Identity Store API establishes a connection to the backend store to complete authentication. AD and OVD are shown as example identity stores.
3. The Identity Store layer returns the results to the plug-in and the authentication engine layer. The authentication layer maps the error codes from the backend to the corresponding Access Manager error codes. For information on standard error codes, See Standard Error Codes.

   The results include any authentication error codes and native error codes acquired from the identity stores. Native error codes are returned to the login page as unmapped secondary error codes (`p_sec_error_msg`). For information about secondary error codes, See Secondary Error Message Propagation.

   Error codes are returned as query parameters to the error or login page. Error codes are transmitted as HTTP Request parameters to the error page. The query parameter is named `p_error_code`.
4. The primary error code message is a localized string containing the detailed text for the error code. This can be obtained from the appropriate resource bundle file using the error code.

The following sections contain more information.

- Authentication Using an Agent
- Authentication Using Unsolicited POST
- Authentication Using Unsolicited Login With DCC WebGates

4.2.1 Authentication Using an Agent

Programmatic authentication using HTTP client APIs is supported for the OAM Server. The following sections have more information.

- Program based authentication using OAM Server
- Process Overview: Developing Programmatic Clients

4.2.1.1 Program based authentication using OAM Server

In this release, the OAM Server uses Javascript to transmit the login form for credential input to the client. In the case where the client or device cannot understand a script-based redirect, the user-agent the client uses must be configured as a programmatic client. OAM Server is then recognized the client as a programmatic one and not as a browser based one. In this case, OAM Server will not use javascript based redirect.

The following is an example of the configuration setting in oam-config.xml. Multiple entries can be added under the `userAgentType` setting.

```
<Setting Name="userAgentType" Type="htf:map">
   <Setting Name="Mozilla/4.0 (compatible; Windows NT 5.1)
 OracleEMAgentURLTiming/3.0" Type="xsd:string">PROGRAMATIC</Setting>
 </Setting>
```

4.2.1.2 Process Overview: Developing Programmatic Clients

Use the following process when developing Access Manager programmatic clients.

- The application invokes a protected resource via HTTP channel using the client API.
- OAM Server redirects to the login page with the request parameters: `site2pstoretoken` and `p_submit_url`.

  `p_submit_url` contains the programmatic authentication endpoint. For example: `http(s)://host:port/sso/auth`. The default browser action URL creates a session on the server side and is not present in the programmatic authentication endpoint. The programmatic authentication endpoint will not create a session for every authentication, rather it will use a global session for a user. This session will be the same for all authentication performed programmatically for a specific user.
- Programmatic clients are expected to submit credentials to the programmatic endpoint.
- Clients must submit the following information to `p_submit_url`: `ssousername`, `password`, `s2pstoretoken` (obtained in Step 3).
- OAM Server validates credentials, and if valid, redirects the request to the initial protected application.
- In case of credential validation error, `p_error_code` is returned.

4.2.2 Authentication Using Unsolicited POST

Use the following process for programmatic authentication using unsolicited POST.

1. Enable Direct Authentication.

In oam-config.xml, ensure that `ServiceStatus` under `DirectAuthenticationServiceDescriptor` is set to `true`. (`DirectAuthenticationServiceDescriptor` is under `OAMServicesDescriptor`).

2. Submit the following information to the endpoint `https://oam_host:oam_port/oam/server/authentication`:
   - **username**
   - **password**
   - **successurl**, for example, `http://machinename.example.com:7778/sample-web/headers.jsp`.
3. Once the credentials are validated, OAM Server redirects to the success URL after setting `OAM_ID` cookie as part of HTTP redirect (HTTP response code 302).

To allow direct authentication only for POST, or vice-versa:

1. Login to Oracle Access Management administration console and navigate to **Policy Configuration**, then **Application Domains**.
2. Select edit **IAMSuite**. Navigate to **Resources**, then search and edit resource **/ oamDirectAuthentication**.
3. Under **Operations**, de-select all operations that are not to be supported, except POST. For example, GET, DELETE.

To change how username and password credentials are authenticated for different success URLs:

1. During the direct authentication request, along with `username`, `password` and `successurl`, pass another parameter `type` with a value specifying the authentication mechanism.
2. Go to **IAMSuite** application domain. Create a new resource with the resource URL `/oamDirectAuthentication`, and query string with name `type` and value specified in Step 1.
3. Associate this resource to the authentication scheme that supports the `type` selected.
4. Create multiple resources with the URL `/oamDirectAuthentication` and different values for the query string `type` (for example, `type=FORM`, `type=CREDS`) and associate it to corresponding authentication schemes.

4.2.3 Authentication Using Unsolicited Login With DCC WebGates

The following procedure will configure an unsolicited login using DCC WebGates.

1. Configure the DCC WebGate.

   See Configuring a DCC WebGate for X509 Authentication in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.
2. Enable Direct Authentication.

   Ensure that ServiceStatus under DirectAuthenticationServiceDescriptor in the `oam-config.xml` file is set to true. The DirectAuthenticationServiceDescriptor parameter is under OAMServicesDescriptor.
3. Add `TunneledUrls=/oam/server/authentication` to the User-defined parameters of the DCC WebGate profile.

   `/oam/server/authentication` is a Direct Authentication URL.

> **Note:**
>
> Add globalHMACFlag=true to the User-defined parameters of the DCC WebGate profile.

4. Restart the DCC WebGate.
5. Ensure that the host/port variation is added to the resource's WebGate.

   For example, if the resource's WebGate is `http://<RWG-Host>:<port>/`, add the host variation <RWG-Host> and the port variation <port> to the Host Identifier.

> **Note:**
>
> DCC WebGates require no configuration in the Host Identifier so add the host variation to the resource's WebGate host identifier only. The resource WebGate and DCC WebGate have different profiles and sets of host identifiers.

6. Protect `/pages/login.jsp` and `/oam/**` and `/oamsso-bin/login.pl` with the Public Resource Authentication and Authorization Policy of the DCC WebGate application domain.

   `/oam/server/authentication` should be protected by the Public Resource Authentication and Authorization Policy scheme in the DCC Application Domain. `/oamsso-bin/login.pl` and `/pages/login.jsp` are login pages and should be marked as public resources in the DCC. The successurl value needs to be protected and is provided as the post parameter.

> **Note:**
>
> See Authentication Using Unsolicited POST to configure the Authentication Scheme for the resources protected using the Direct Authentication URL.

7. The DCC authentication end point can be accessed here at `http://<dcc-host>:<dcc-port>/oam/server/authentication`.

   There is a TYPE parameter which can be mapped to a specific application. Post the username, password and successurl parameters to this DCC end point to access the resource.

4.3 About Custom Login Pages

The custom login page can be created as a WAR file and packaged with the necessary resource bundle files. The WAR file can then be deployed on an application behind a DCC, or if DCC is not used, the page can be deployed on the same server where ECC is running.

When using ECC, the following settings must be specified for the Authentication scheme using the custom WAR file.

- **Context Type** = CustomWar
- **Challenge URL** = Relative path for the URL of the login page inside the WAR file
- **Context Value** = Custom WAR's root path. If a customized error page is included as part of the Custom WAR file, you must specify the absolute URL in the authentication policy-

failure redirect URI. For example: `http://host:port/SampleLoginWar/pages/MFAError.jsp`.

See Also,

- Authentication Schemes and Pages in *Fusion Middleware Administering Oracle Access Management*
- Understanding Form-Based Login Page authentication
- What is Page Redirection Process

4.3.1 Understanding Form-Based Login Page authentication

Form-based authentication enables the development of customized Web forms that process login credentials using Access Manager's authentication mechanisms. These forms are HTML pages that enable you to present login information in different languages, to display user interface elements that comply with your company's presentation standards, and to add functions required for password management.

The form or login application can be written to process the redirect from the user and render the HTML using your preferred technology (JSP, ASP.net, Perl, PHP, and so on). This allows you to customize the look and feel of the page so it reflects company standards, and it enables pre-processing of the user's submission (POST) before their credentials are sent to the OAM Server, if desired.

Three modes of request cache are supported: `basic`, `cookie`, and `form`. When working in `basic` mode, the `request_id` is a mandatory parameter and should be sent back. In `form` mode, the OAM_REQ token is mandatory and should be sent back if available. In `cookie` mode, OAM_REQ is set as a cookie.

A custom login form page has the following requirements:

- The page must be built to support the desired authentication module.
- The page must support retrieval of the OAM_REQ token. See Returning the OAM_REQ Token.
- The page must retrieve the end point. See Returning the End Point.

4.3.2 What is Page Redirection Process

When writing a custom login page for authentication by Access Manager, a common method is to redirect a user to a login page that is hosted outside of the OAM Server. The user is redirected to the custom page or application you have written.

When a form-based authentication scheme has been created with an external challenge type, the OAM agent redirects the user first to the `obrareq.cgi` URL, which in turn redirects the user to the login page specified as the Challenge URL for the authentication scheme. The Challenge Redirect URL declares the DCC or ECC endpoint. The Challenge URL is the URL associated with the Challenge method such as FORM.

On the redirect page, `request_id` and `redirect_url` are added to the query string. For example:

```
?
request_id=5092769420627701289&redirect_url=http%3A%2F%2Fexample.com%3A7777%2Fscripta%2Fprintenv
```

When using the x509 authentication scheme there is no separate page for login credentials as authentication occurs transparently. However, page redirection also applies to non-form based authentication methods. For example, when using an x509 Authentication Scheme, you can direct users to a confidentiality disclaimer statement, or similar, before a protected resource is displayed. In this case, enter the path to the disclaimer page and have that page redirect to the `/oam/CredCollectServlet/X509` page. Be sure to present the original query scheme.

4.4 Understanding Custom Error Pages

This section contains information specific to the development of custom error pages. It contains the following information.

- Enabling Error Page Customization
- Standard Error Codes
- Security Level Configuration
- Secondary Error Message Propagation
- Sample Code: Retrieving Error Codes
- Error Data Sources Summary

For information on how error code query parameters `p_error_code` and `p_sec_error_msg` will map to the custom error codes in your environment, see Sample Code: Retrieving Error Codes.

> **Note:**
>
> Error Codes in an Authentication Plug-In and Sample Code: Retrieving Password Policy Error Codes

4.4.1 Enabling Error Page Customization

Once a custom Error page is packaged and deployed as an out-of-the-box Web application archive (WAR) file, use the `updateCustomPages` WLST command to enable and disable it. The custom Error page has a specific location that must be respected when deploying the custom web application. The location is:

```
http(s)://<host>:<port>/<context>/pages/Error.<pageExtension>
```

See Also,

Oracle Fusion Middleware WebLogic Scripting Tool Command Reference for Identity and Access Management. For information on `updateCustomPages` and other WLST commands.

Specifying the Custom Error and Logout Page Deployment Paths

4.4.2 Standard Error Codes

Access Manager provides standard error codes that indicate the reasons for failure. Common reasons include an invalid username and password combination, a locked or disabled user account, or an internal processing error. The reason for the authentication error is received from the backed identity store and mapped to a specific error code maintained in the Access

Manager Server. This error code is then propagated to the login or error page. (The custom error page also displays any internal server errors as well.)

Table 4-1 summarizes the standard error information available in login and error pages.

Table 4-1 Types of Error Information

Message Type	Description
Error code	A string containing a specific number. The error codes are managed solely by Access Manager. Query string parameter is named `p_error_code`.
Primary error message	A localized string containing the detailed text for the error code. Is based on the client locale, namely, the user's browser language setting.
Secondary error message	A non localized string containing the real cause for the failure. Secondary error message can be provided by a custom authentication plug-in or be returned by an identity store. Query string parameter is named `p_sec_error_msg`.

Table 4-2 lists all the error message codes sent by the OAM Server and the corresponding primary error message. If a primary error message has been customized for an application, the application must map this custom message to the corresponding standard error message maintained by OAM Server. There is no difference between OAM-1 and OAM-2 error codes.

Table 4-2 Standard Error Codes and Message

Error Code	Primary Error Message
OAM-1	An incorrect Username or Password was specified.
OAM-2	An incorrect Username or Password was specified.
OAM-3	Unexpected Error occurred while processing credentials. Please retry your action again!
OAM-4	System error. Please contact the System Administrator.
OAM-5	The user account is locked or disabled. Please contact the System Administrator.
OAM-6	The user has already reached the maximum allowed number of sessions. Please close one of the existing sessions before trying to login again.
OAM-7	System error. Please re-try your action. If you continue to get this error, please contact the Administrator.
OAM-8	Authentication failed.
OAM-9	System error. Please re-try your action. If you continue to get this error, please contact the Administrator.
OAM-10	The password has expired. Please contact the System Administrator.

4.4.3 Security Level Configuration

An error code's security level determines the error code that is returned by OAM Server. The security level is configured by an administrator using the Access Manager Configuration panel in the administration console. The following security level settings are available when configuring error codes for custom login pages:

- SECURE: Most secure level. Provides a generic primary error message that gives little information about the internal reason for the error.
- EXTERNAL: The recommended level and is the default.
- INTERNAL: The least secure level. Enables propagation of error code to login or error page.

Table 4-3 lists the standard error codes (see Table 4-2) that are propagated to the login or error page according to security level.

Table 4-3 Error Condition Mapping by Security Level

Error Condition	Internal Mode	External Mode	Secure Mode
Invalid login attempt.	OAM-1	OAM-2	OAM-8
Processing submitted credentials failed for a reason. For example, in WNA mode the `spnego` token is not received.	OAM-3	OAM-3	OAM-8
An authentication exception is raised for a reason.	OAM-4	OAM-4	OAM-9
User account is locked due to certain conditions. For example, the invalid attempt limit is exceeded.	OAM-5	OAM-5	OAM-8, or OAM-9 if OIM is integrated
User account is disabled.	OAM-5	OAM-5	OAM-9
User exceeded the maximum number of allowed sessions. This is a configurable attribute.	OAM-6	OAM-6	OAM-9
Can be due to multiple reasons. The exact reason is not propagated to the user level for security reasons. Is the default error message displayed when no specific error messages are propagated up.	OAM-7	OAM-7	OAM-9

When an error condition occurs, the OAM Server will forward to the default error page, unless the default page has been overridden as a `failure-redirect-url` in the authentication policy. When using a custom error page, the absolute error page URL must be set as the `failure_redirect_url` in the authentication policy so that the server will redirect to the custom page. The custom login page typically has the logic to serve as the error page.

In the case of error conditions OAM-1 and OAM-8, which enable the credentials to be collected again, the user is returned to the login page.

4.4.4 Secondary Error Message Propagation

The authentication engine layer maps exceptions from the backend identity store to error codes specific to OAM Server. These codes are then propagated. Plug-ins can retrieve the secondary error code and then propagate so that appropriate action can be taken.

> **Note:**
>
> The primary error codes are propagated to the error or login page in all modes. The secondary error message is propagated only when the security level is configured to be INTERNAL. For more information, see Security Level Configuration.

Secondary error messages are sent as HTTP Request parameters to the error page. The query parameter is named `p_sec_error_msg`. This message is a concatenated string of code and message text from the backend and is not translated.

For example, in the case where OVD is the backend and invalid credentials are entered, authentication fails and the cause is returned from the backend as `LDAP:error code 49-Invalid Credentials` and the OAM Server error code is returned as `OAM-1`. In this case the following data will appear in the log in page:

4.4.5 Sample Code: Retrieving Error Codes

An error code is sent as HTTP Request parameters to the error page. The query parameter is named `p_error_code`. This parameter value will contain error code values returned by the OAM Server, such as OAM-1.

> **Note:**
>
> See Table 4-2 for standard Access Manager error codes and corresponding message. These error codes do not have supplementary information.

A custom login page can be associated with a custom resource bundle to transform the error codes to meaningful messages that can be displayed to the end user. However, if the custom login page does not require meaningful error messages or translations, then the custom resource bundle is not required.

A local resource bundle file must be created with the error condition mapped to Access Manager error codes as summarized in Table 4-3. The file can be consumed in the login or error page. Example 4-1 provides a resource bundle code sample and Example 4-2 provides an error code page sample.

Example 4-1 Resource Bundle Code

```
package mytest.error;
     Import java.util.ListResourceBundle;
  public class ExampleErrorMsg extends ListResourceBundle {
    /* (non-Javadoc)
     * @see java.util.ListResourceBundle#getContents()
     */
    public Object[][] getContents()
    {
        return m_contents;
    }
    /** The Constant m_contents. */
    private static final Object[][] m_contents =
     {
        {"OAM-1", " An incorrect Username or Password was
 specified "},
```

```
        {"OAM-2", " An incorrect Username or Password was
 specified "},
        {"OAM-3", "Unexpected Error occurred while processing
 credentials. Please retry your action again!"},
        {"", ……};
             }
    }
```

Example 4-2 Error Code Page

```
<%@page import="mytest.error.ExampleErrorMsg"%>
  //initializing the messageBundle first
  String defaultResourceBundle = "mytest.error.ExampleErrorMsg";
  java.util.Locale myLocale = request.getLocale();
  ResourceBundle msgBundle=
 ResourceBundle.getBundle(defaultResourceBundle,myLocale);
String errCode = request.getParameter("p_error_code");
String secondaryErrMessage = request.getParameter("p_sec_error_msg");
  <%
        if(errCode != null && errCode.length() > 0) {
          try {
             simpleMessage = msgBundle.getString(errCode);
          } catch(Exception e) {
             //get the default authn failed message
             simpleMessage = msgBundle.getString("OAM-8");
          }
  %>
  <div class="message-row">
      <p class="loginFailed"> <%=simpleMessage%> </p>
  </div>
```

4.4.6 Error Data Sources Summary

Table 4-4 summarizes the error data sources available to an authentication plug-in.

Table 4-4 Authentication Plug-In Error Data Sources

Source	Parameter
Error code	HTTP Request parameter: `p_error_code`
Primary error message	Is obtained from the resource bundle, using the error code
Secondary Error Message (sent only in INTERNAL error mode)	HTTP Request parameter: `p_sec_error_msg`
Concatenated list of server error codes	HTTP Request parameter: `p_error_codes_list`
Password Plugin error message	HTTP Request parameter: `rejectedRulesDesc`
Plugin client responses	HTTP Request parameter: PLUGIN_CLIENT_RESPONSE
Plugin error responses	HTTP Request parameter: PLUGIN_ERROR_RESPONSE

4.5 Understanding Custom Password Pages

When writing a custom password page for authenticating users, a common method to follow is to redirect the user to a password page hosted outside Access Manager; custom pages are written to process a redirect from the user and to render the HTML.

Access Manager processes user-entered passwords and password changes with form-based Java Server Pages (JSP). These pages can be customized in several languages or to display a company logo. Custom password pages can be kept as a JSP, or written using ASP.net, Perl, PHP, and other similar technologies. The pages enable pre-processing of the user's submission (POST) before their credentials are sent to Access Manager, if desired. The following sections contain more details.

- Customizing the Password Page WAR
- Using the Request Cache
- Specifying the Password Service URL
- Sample Code: Retrieving Warning Messages
- Sample Code: Retrieving Password Policy Error Codes
- Sample Code: Obtaining Password Policy Rules

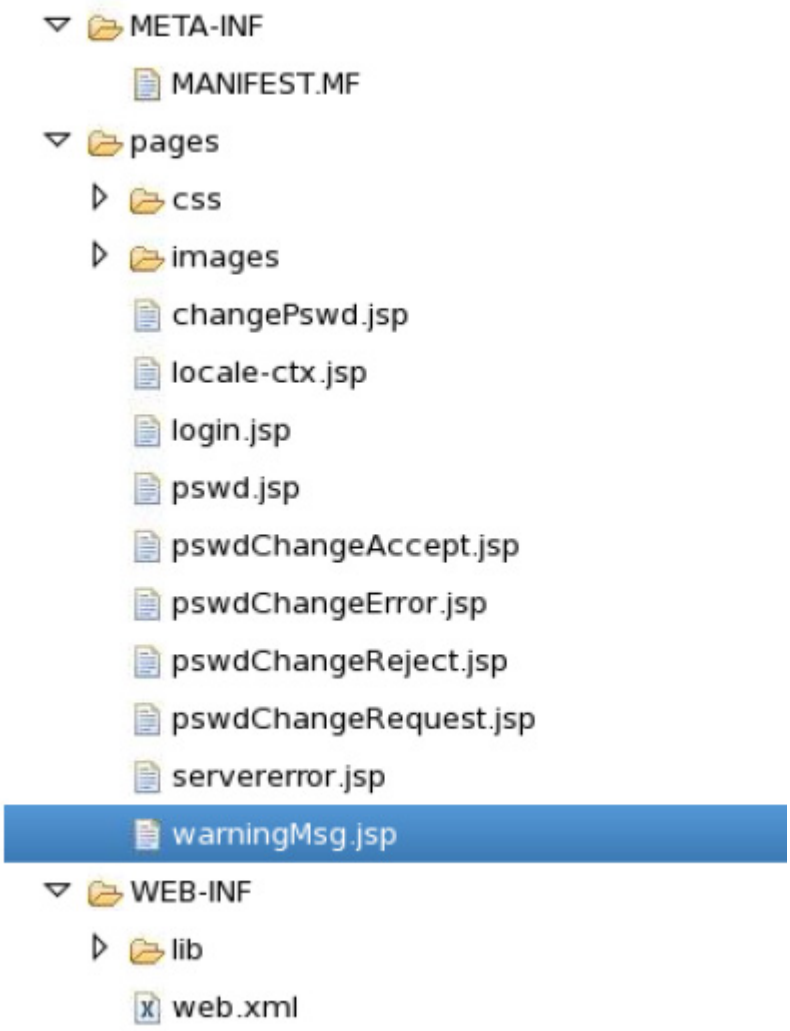
4.5.1 Customizing the Password Page WAR

A basic Web archive (WAR) file that includes the required translation bundles for login and password pages is provided when Access Manager is installed. This WAR is called `oamcustompages.war` and is the starting point for customizing password pages. A custom password service form page has the following requirements:

- The page must support retrieval of the OAM_REQ token as documented in Returning the OAM_REQ Token.
- The page must retrieve the end point as documented in Returning the End Point.

The WAR is located in the $IDM_HOME//oam/server/tools/custompages/ directory. The advantage of using the WAR is that the basic structure is already in place. Figure 4-2 displays the structure of the WAR; the CSS and images can be customized as per your requirements.

Figure 4-2 Unarchived WAR



4.5.2 Using the Request Cache

Three modes of request cache are supported: basic, form and cookie.

- Basic mode defines the request_id as a mandatory parameter that must be sent back.
- Form mode defines the OAM_REQ token as mandatory and should be sent back, if available.
- Cookie mode sets OAM_REQ as a cookie.

4.5.3 Specifying the Password Service URL

The starting point for password pages is `pswd.jsp`. The location of this page is configured under Password Policy using the Oracle Access Management Administration Console. See *Password Policy Management*.

1. Log in to the Access Manager Administration Console as administrator.
2. Click the Password Policy link under Access Manager.

3. Define or modify the value of the Password Service URL attribute.
4. Click Apply.

4.5.4 Sample Code: Retrieving Warning Messages

A user-facing page has access to the number of days before which the password will expire. The following code snippet illustrates how to obtain the number of days or hours before the password expires.

```
String message = "";
if(errCode != null && errCode.equals("1")) {
message = msgBundle.getString("USER_PSWD_WARNING") + errCode +
msgBundle.getString("USER_PSWD_DAY");
} else if (errCode != null && errCode.equals("0")) { message =
msgBundle.getString("USER_PSWD_WARNING_HOURS"); } else {
message = msgBundle.getString("USER_PSWD_WARNING")+ " " + errCode + " " +
msgBundle.getString("USER_PSWD_DAYS");
}
```

4.5.5 Sample Code: Retrieving Password Policy Error Codes

A user-facing page has access to the password policy result context and has the ability to obtain applicable messages. Each message may include supplementary information, depending on the message. The following code snippet shows how a page can obtain the message and supplementary information from the password policy result context:

```
String simpleMessage = "";
  String result = request.getParameter("rejectedRuleDesc");
  if(result.indexOf('~') != -1) {
   String[] results = result.split("~");
   for(String eachResult : results) {
      if(eachResult.indexOf(":") != -1) {
        String messageKey = eachResult.substring(0, eachResult.indexOf(":"));
        String resourceBundleKey = UrlSubstitutionMessages.ERRORCODEMAP.get(messageKey);
        String placeHolderValue = eachResult.substring(eachResult.indexOf(":") + 1,
eachResult.length());
        String displayValue = Localizer.localize(resourceBundleKey, placeHolderValue,
myLocale);
        simpleMessage += displayValue + "<br>";
      }
      else {
        String resourceBundleKey = UrlSubstitutionMessages.ERRORCODEMAP.get(eachResult);
        String displayValue = Localizer.localize(resourceBundleKey, null, myLocale);
        simpleMessage += displayValue + "<br>";

      }
   }
  }
```

For example, if the password doesn't have enough characters, the following will be the result in context:

- `PasswordRuleDescription.getResourceBundleKey()` returns `"passwordPolicy.error.minLength"`
- `PasswordRuleDescription.getPlaceHolderValue()` returns minimum number of characters
- `PassswordRuleDescription.eachDesc.getDisplayValue()` returns fully translated message

The password plug-in redirects to the password pages and the corresponding message is received from the password policy. These error codes are sent to the password policy pages as HTTP Request parameter named `ruleDes`. Table 4-5 lists the available error codes, message key, and the corresponding message that can be returned during password validation.

Table 4-5 Password Validation Error Codes

Message Key in URL	Message Key for Resource Bundle	Message Text
PSWD-1	passwordPolicy.message.minLength	Password must be at least {0} characters long
PSWD-2	passwordPolicy.message.maxLength	Password must not be longer than {0} characters
PSWD-3	passwordPolicy.message.minAlpha	Password must contain at least {0} alphabetic characters
PSWD-4	passwordPolicy.message.minNumber	Password must contain at least {0} numeric characters
PSWD-5	passwordPolicy.message.minAlphaNumeric	Password must contain at least {0} alphanumeric characters
PSWD-6	passwordPolicy.message.minSpecialChars	Password must contain at least {0} special characters
PSWD-7	passwordPolicy.message.maxSpecialChars	Password must not contain more than {0} special characters
PSWD-8	passwordPolicy.message.maxRepeated	Any particular character in the password must not be repeated more than {0} times
PSWD-9	passwordPolicy.message.minUnique	Password must contain at least {0} unique characters
PSWD-10	passwordPolicy.message.minUpperCase	Password must contain at least {0} uppercase letters
PSWD-11	passwordPolicy.message.minLowerCase	Password must contain at least {0} lowercase letters
PSWD-12	passwordPolicy.message.maxAge	Password will expire {0} days after the last password change
PSWD-13	passwordPolicy.message.warnAfter	Password change reminder will be sent {0} days after the last password change
PSWD-14	passwordPolicy.message.reqdChars	Password must contain the following characters: {0}
PSWD-15	passwordPolicy.message.invalidChars	Password must not contain the following characters: {0}
PSWD-16	passwordPolicy.message.validChars	Password can contain the following characters: {0}
PSWD-17	passwordPolicy.message.invalidStrings	Password must not contain the following strings: {0}
PSWD-18	passwordPolicy.message.startsWithChar	Password must start with an alphabetic character
PSWD-19	passwordPolicy.message.disAllowUserId	Password must not match or contain user ID
PSWD-20	passwordPolicy.message.disAllowFirstName	Password must not match or contain first name

Table 4-5 (Cont.) Password Validation Error Codes

Message Key in URL	Message Key for Resource Bundle	Message Text
PSWD-21	passwordPolicy.message.disAllowLastName	Password must not match or contain last name
PSWD-22	passwordPolicy.message.dictMessage	Password must not be a dictionary word
PSWD-23	passwordPolicy.message.enforceHistory	Password must not be one of {0} previous passwords
PSWD-24	passwordPolicy.message.minAge	Password cannot be changed for {0} days after the last password change
PSWD-25	passwordPolicy.message.minUnicode	Password must contain at least {0} Unicode characters
PSWD-26	passwordPolicy.message.maxUnicode	Password must not contain more than {0} Unicode characters

4.5.6 Sample Code: Obtaining Password Policy Rules

A user-facing page has access to the password policy rules applicable for the user. Each message may include supplementary information, depending on the message. The following code snippet shows how a page can obtain the rules and supplementary information from the password policy result context:

```
String simpleMessage = "";
String result = request.getParameter("ruleDesc");
if(result.indexOf('~') != -1) {
String[] results = result.split("~");
for(String eachResult : results) {
if(eachResult.indexOf(":") != -1) {
String messageKey = eachResult.substring(0, eachResult.indexOf(":"));
String resourceBundleKey = UrlSubstitutionMessages.ERRORCODEMAP.get(messageKey);
String placeHolderValue = eachResult.substring(eachResult.indexOf(":") + 1,
eachResult.length());
String displayValue = Localizer.localize(resourceBundleKey, placeHolderValue, myLocale);
simpleMessage += displayValue + "<br>";
}
else {
String resourceBundleKey = UrlSubstitutionMessages.ERRORCODEMAP.get(eachResult);
String displayValue = Localizer.localize(resourceBundleKey, null, myLocale);
simpleMessage += displayValue + "<br>";

}
}
}
```

For example, if the password does not have enough characters, the following will be the result in context:

- `PasswordRuleDescription.getResourceBundleKey()` returns `"passwordPolicy.error.minLength"`
- `PasswordRuleDescription.getPlaceHolderValue()` returns minimum number of characters

- `PasswordRuleDescription.eachDesc.getDisplayValue()` returns fully translated message

4.6 Using the Credential Collectors with Custom Pages

A credential collection component is enabled to serve as communication endpoint for custom pages. The credential collector facilitates interaction with the customized user interface.

Either one of two Access Manager credential collection components can be enabled to serve as the communication endpoint for custom pages, and to facilitate interaction with the customized user interface. The Access Manager credential collection components are:

- The Embedded Credential Collector (ECC) which can be used out-of-the-box with no additional installation and setup. In cases where the ECC is used, the default pages are accessed from the following locations:
  - Login page: http(s)://*host:port*/oam/pages/login.jsp
  - Error page: http(s)://*host:port*/oam/pages/servererror.jsp
- The Detached Credential Collector (DCC) which is recommended for greater scalability and security isolation in production deployments. In cases where the DCC is used, the default pages are accessed from the following locations:
  - Login page: /oamsso-bin/login.pl
  - Login action URL: /oam/server/auth_cred_submit. This is the default action URL if no `action` is configured in the authentication scheme parameters. No corresponding physical page is located with the default URL. A physical page is needed at the URL location only when an `action` has been configured in the authentication scheme and a runtime `action type` results in a pass through on the action URL.
  - Error page: /oberr.cgi. This is a URL pattern recognized by DCC and is not a physical location.

Regardless of which credential collection component is enabled for communicating with users, the design and implementation of custom pages in your environment is almost identical.

> **Note:**
>
> For more information about the Access Manager Server credential collectors, See *Configuring WebGate and Authentication Policy for DCC*.

This section contains details regarding the DCC.

- About the Detached Credential Collector with Custom Pages
- Creating a Form-Based Login Page Using DCC
- About Custom Login and Error Pages for DCC Tunneling

4.6.1 About the Detached Credential Collector with Custom Pages

The primary differences when using the DCC to collect credentials from a custom page include the following:

- The DCC is installed with default pages implemented as Perl scripts using HTML templates located in the following Webgate Oracle Home (*WebgateOH*) directories:

  - *WebgateOH*/webgate/ohs/oamsso
  - *WebgateOH*/webgate/ohs/oamsso-bin
- In addition to customizing login pages for supported authentication mechanisms, the default error and logout pages can be customized.
  - The default error page is triggered when an error condition occurs outside of the authentication flow, or if the failure redirect URL is not specified in the authentication scheme. The default error page template and associated error messages are located in a language and locale specific subdirectory within the Webgate Oracle Home. For example, the exact location for `en-us` is: *WebgateOH*/webgate/ohs/lang/en-us/WebGate.xml.
  - The default logout page is located in *WebgateOH*/webgate/ohs/oamsso-bin/logout.pl.
- Custom pages can be deployed on the Oracle HTTP Server hosting the DCC or, in the case of JSP or Servlets, on a web container fronted by it.
- Use a configurable URL to allow HTML forms to post collected data to the DCC. The `action` challenge parameter in the authentication scheme specifies the URL where the credentials are expected.
- `requestid` query parameter handling is not required.

4.6.2 Creating a Form-Based Login Page Using DCC

To create a a form based login page using DCC:

1. Create an HTML form from which the user's credentials (user name and password) can be submitted.

   For more information, see Understanding Form-Based Login Page authentication.
2. Place the form in an unprotected directory, or in a directory protected by an Anonymous authentication scheme, on your Web server with DCC.
3. Create a form-based authentication scheme and specify the path to the login form as the Challenge URL.
4. Call the form action using HTTP GET or POST.
5. Protect the target URL in the action of the login form with a policy.
6. Configure the challenge parameters in the authentication scheme.
7. Specify the authentication module to use to process the credentials.

4.6.3 About Custom Login and Error Pages for DCC Tunneling

Custom login and error pages can be created when using DCC tunneling. The default pages are accessed from the following locations:

- Login page: http(s)://*DCChost:DCCport*/oam/pages/login.jsp
- Error page: http(s)://*DCChost:DCCport*/oam/pages/servererror.jsp

A procedure (similar to Creating a Form-Based Login Page Using DCC) should be followed to create and use these custom pages. For more details on credential collection and DCC tunneling, See Understanding Credential Collection and Login.

4.7 Specifying the Custom Error and Logout Page Deployment Paths

Custom Error and logout pages have specific paths that must be respected when deploying the custom application. The error page path is:

```
http(s)://<host>:<port>/<context>/pages/Error.<pageExtension>
```

The logout page path is:

```
http(s)://<host>:<port>/<context>/pages/Logout.<pageExtension>
```

context and pageExtension are variables that can be configured using the `updateCustomPages` WLST command. pageExtension has a default value of jsp but can be left blank while running the command. `updateCustomPages` will add a context path and page extension to the configuration.

```
<Setting Name="ssoengine" Type="htf:map">
  <Setting Name="ErrorConfig" Type="htf:map">
  <Setting Name="ErrorMode" Type="xsd:string">EXTERNAL</Setting>
  <Setting Name="CustomPageExtension" Type="xsd:string">html</Setting>
  <Setting Name="CustomPageContext" Type="xsd:string">SampleApp</Setting>
</Setting>
</Setting>
```

> **Note:**
>
> See updateCustomPages in *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference for Identity and Access Management* for details.

5
Managing Policy Objects

Access Manager provides a Policy Administration API that enables Create, Read, Update, and Delete (CRUD) operations on its policy objects. This chapter describes the API and provides examples for using a RESTful Web service for Access Manager policy administration.
The following sections contain details regarding the Policy Administration API.

5.1 About the Policy Administration API

Oracle Policy Administration API supports representational state transfer (REST) interfaces for administering Access Manager policy objects as RESTful resources

Oracle Policy Administration API conforms to the Java Specification Request (JSR) 311: JAX-RS 1.1 specifications: Java API for RESTful Web Services 1.1. For more information, See: http://download.oracle.com/otndocs/jcp/jaxrs-1.1-mrel-eval-oth-JSpec/.

This section provides the following topics:

5.1.1 Access Manager Policy Model

The Policy Administration API exposes Access Manager policy model objects (also known as artifacts) to RESTful clients, modeling operations on these objects to HTTP requests containing specific URLs and operations. Operations are subject to Access Manager policy administration rules that enforce policy validation and consistency.

Figure 5-1 shows the policy model and the relationship of the policy objects that can be managed.

Figure 5-1 Policy Model

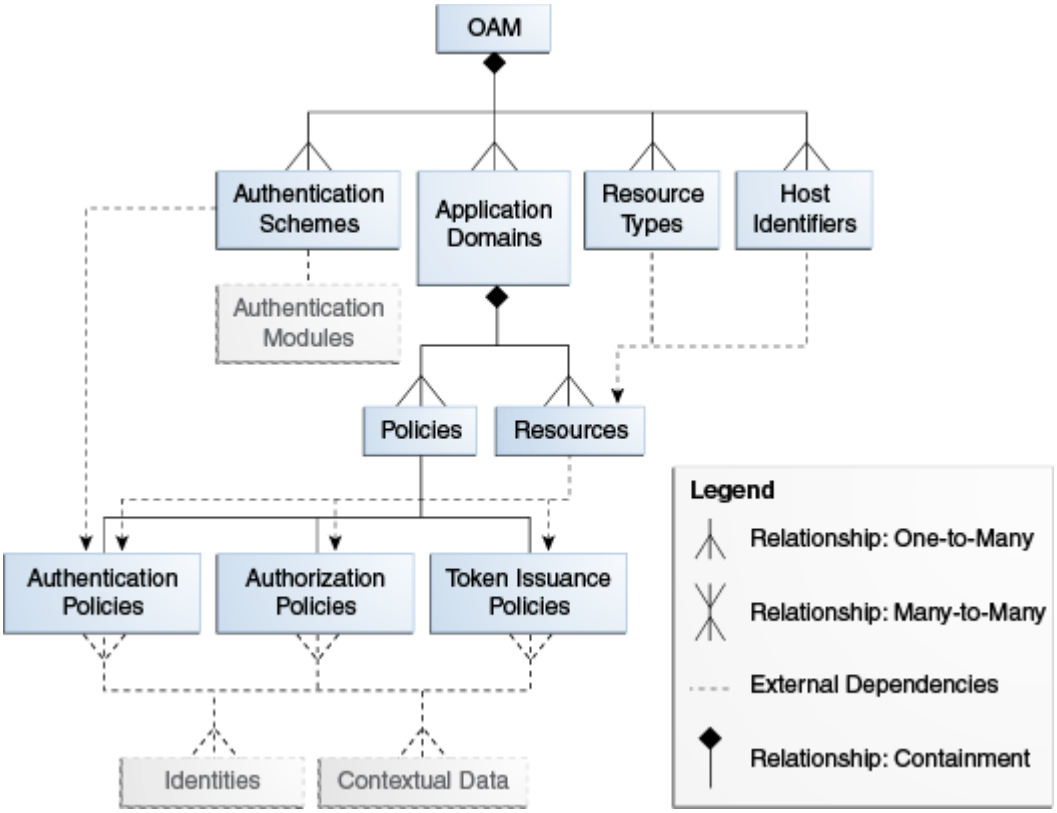


Table 5-1 provides details about the policy objects that can be managed using the RESTful interfaces. Each policy object is represented as an HTTP resource that is accessible through an HTTP uniform resource locator (URL).

Table 5-1 Policy Objects

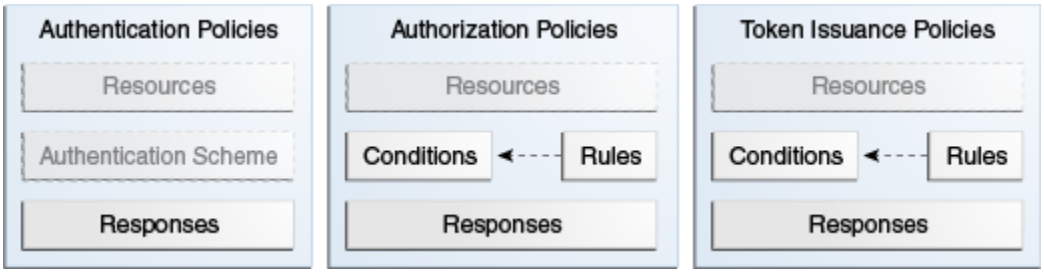
Object Name	Description
Application Domain	The top-level construct of the policy model. Each application domain provides a logical container for resources, and the associated authentication and authorization policies that dictate who can access these.
Host Identifier	A host can be known by multiple names. To ensure that Access Manager recognizes the URL for a resource, Access Manager must know the various ways used to refer to that resource's host computer.
Resource	Resources represent a document, or entity, or pieces of content stored on a server and available for access by a large audience. Clients communicate with the server and request the resource (using HTTP methods) that is defined by an existing Resource Type.
Resource Type	A resource type describes the kind of resource to be protected.
Authentication Policy	Authentication policies specify the authentication methodology to be used for authenticating the user. Policies define the way in which the resource access is to be protected.
Authorization Policy	Authorization policies specify the conditions under which a subject or identity has access to a resource.

Table 5-1 (Cont.) Policy Objects

Object Name	Description
Token Issuance Policy	A Token Issuance Policy defines the rules under which a token can be issued for a resource (Relying Party Partner) based on the client's identity, with the client either being a Requester Partner or an end user.
Authentication Scheme	A named component that defines the challenge mechanism, level of trust, and the underlying authentication module required to authenticate a user.

Figure 5-2 shows the contents of the Access Manager policies.

Figure 5-2 Policy Contents



You can access the OAM Server RESTful interfaces through client applications such as:

- Web browsers
- cURL
- GNU Wget

5.1.2 Security Model

REST services are protected by the container security that enforces the required roles. The Policy Administration REST API is protected by administrative roles.

The enforcement policy configuration for the API is similar to the policy enforcement for Policy Administration actions performed in the administration console. For example, client invocations are expected to supply credentials in the Authorization Header of the HTTP request, ensuring that the client invocations remain stateless as seen in the following sample request:

The following is an example of the response content returned from the sample HTTP request, which contains a list of application domains:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><ApplicationDomains>
<ApplicationDomain>
    <name>Demo Application Domain</name>
    <description>Policy objects enabling OAM Agent to protect deployed Demo
applications</description>
</ApplicationDomain>

<ApplicationDomain>
    <name>Clear Vision Domain</name>
    <description>Policy objects enabling OAM Agents to protect Clear Vision
applications</description>
</ApplicationDomain>
</ApplicationDomains>
```

The authentication provider for user authentication is based on Access Manager application configuration. When the REST service is protected by a Webgate, the Webgate decides about the access request based on the Authentication Scheme associated with the URL. These URLs have Cookieless Basic as the authentication method. The Cookieless Basic scheme should not be changed, instead have it protected with more than one scheme. In such a case, the Webgate treats an access request to these resources as pass-through, preserving the Authorization headers of the request. Access Manager process the request based on the Authorization header provided.

5.1.3 Resource URLs

Resource URLs are structured to include the Access Manager product version, the component exposed by the REST service, and the resources being invoked. The basic structure of a resource URL is as follows:

`http(s)://`*`host`*`:`*`port`*`/oam/services/rest/`*`path`*

where:

- **host** is the host where the OAM Server is running.
- **port** is the HTTP or HTTPS port.
- **path** is the relative path that identifies a particular resource. *path* is constructed as /*version*/*component*/*service*/ where:
  - *version* - is the Access Manager product version, such as 11.1.2.0.0
  - *component* - is the component exposed by the RESTful service, such as ssa or sso
  - *service* - is the root resource for that given API, such as hostidentifier

  An example of a *path* value is: `/oam/services/rest/11.1.2.0.0/ssa/policyadmin/hostidentifier/`*`host_identifier_name`*.

The Policy Administration REST Web Application Description Language (WADL) file lists the supported policy resources and methods. The Policy Administration REST WADL document is available at `http://`*`adminserver.example.com`*`:`*`adminport`*`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/application.wadl`.

Additional parameters are required to process the request query parameters. All resource URLs support the OPTIONS method.

Policy objects can be identified by name or id. If both are provided, the id is used.

Table 5-2 summarizes the resource URLs that are exposed to enable administration of the policy objects shown in Figure 5-1. In the following table:

- IDENTIFER refers to the name or id of the object the request refers to.
- APPDOM_IDENTIFER uniquely identifies an existing Application Domain type object by appid or appname.

Table 5-2 Resource URLs

Policy Object	URL	Artifact Mandatory Parameter
Application Domain	`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/appdomain`	IDENTIFIER
Host Identifier	`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/hostidentifier`	IDENTIFIER
Resource Type	`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/resourcetype`	IDENTIFIER
Resource	`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/resource`	IDENTIFIER, APPDOM_IDENTIFIER
Authentication Policy	`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/authnpolicy`	IDENTIFIER, APPDOM_IDENTIFIER
Authorization Policy	`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/authzpolicy`	IDENTIFIER, APPDOM_IDENTIFIER
Token Issuance Policy	`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/tokenpolicy`	IDENTIFIER, APPDOM_IDENTIFIER

5.1.4 URL Resources and Supported HTTP Methods

Access Manager policy object services are mapped to URL resources. Each resource is referenced by a global identifier (URI).

Access to URL resources is based on user role. The RESTful service expects user credentials to be present in the Authentication header of the HTTP request in BASIC mode. If the authenticated user has the policy administration role, the requested policy administration action is performed.

5.1.5 Error Handling

A service request can result in various error conditions ranging from invalid service invocation to server side failures. Failures and error code conditions are reported back to the clients as HTTP return codes with an explanatory message.

Table 5-3 contains the mapping between HTTP return codes and message.

Table 5-3 Error Conditions and HTTP Return Codes

Error Condition	HTTP Return Code	Content
Unable to parse input, or input does not match required entities.	400	Bad request
Service not located	404	Not found

Table 5-3 (Cont.) Error Conditions and HTTP Return Codes

Error Condition	HTTP Return Code	Content
Requested object not found	404	Not found <additional information indicating the not found object)
User not authorized to execute service	401	Unauthorized
Requested method not supported	405	Method not allowed
Client does not accept produced content type	406	Not acceptable
Request parameters semantics incorrect	422	Unprocessable entity <additional information on nature of error>
Client media type unsupported	415	Unsupported media type. Note: The supported media types are text/xml (or application/xml) and application/json.
Failed dependency	424	Failed dependency <additional information on failed dependency>
Generic server failure	500	Internal server error

5.2 Compatibility

The release version number is embedded as part of the REST service URLs exposed by OAM Server. There is no support for forward compatibility. Clients of newer versions cannot expect to send a request to an older version of OAM Server and receive back newer versions of objects. There is backward compatibility support for older clients.

5.3 Managing Policy Objects

Access Manager provides a Policy Administration API that enables Create, Read, Update, and Delete (CRUD) operations on its policy objects. Use these sections to understand the API and refer to examples for using a RESTful Web service for Access Manager policy administration.

This section provides the following topics:

- HTTP Methods
- Media Types
- Resources Summary

5.3.1 HTTP Methods

Table 5-4 describes the supported HTTP methods. A successful HTTP method acts upon a representation of the policy object (resource), which is an xml file. A JavaScript Object Notation (JSON) object is returned.

Table 5-4 Methods For Managing Policy Objects

Method	Action
GET	Retrieves the policy objects.
POST	Creates the policy object.
PUT	Modify a policy object.
DELETE	Delete a policy object.

5.3.2 Media Types

The supported media types are:

- application/xml
- application/json
- text/xml

5.3.3 Resources Summary

Table 5-5 provides detail about each policy resource, the supported HTTP methods, and the results of each action.

Table 5-5 Access Manager Policy Resources Summary

Resource	Method	Description
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/appdomain`	GET	All matching Application Domain resources are returned. If no query parameter is provided, all Application Domain resources are returned. If an ID or NAME query parameter is specified, all matching Application Domain resources are returned.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/appdomain`	POST	The Application Domain object is created by this method. The request body must contain an Application Domain. An Application Domain object matching the request is created. All the policy child objects are also created.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/appdomain`	PUT	An Application Domain object is modified by this method. The request body must contain the Application Domain resource that represents the object. The Application Domain resource matching the specified ID or NAME query parameter is modified. If query parameters are not matched, the Application Domain object matching ID or NAME query parameters will be modified. If both ID and NAME are present, the ID value will be used.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/appdomain`	DELETE	An Application Domain object is deleted by this method. The Application Domain matching NAME or ID query parameter is deleted.

Table 5-5 (Cont.) Access Manager Policy Resources Summary

Resource	Method	Description
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/tokenissuancepolicy`	GET	A Token Issuance Policy object is retrieved by this method. The resource that represents the Token Issuance Policy object is returned. This representation contains the matching Token Issuance Policy resource attributes and their values. Valid query parameters are ID or NAME, and APPDOMAINID or APPDOMAIN. If an APPDOMAINID or APPDOMAIN parameter is not specified, a status code 424 is returned with the appropriate message. If an ID or NAME query parameter is not specified, all Token Issuance Policy resources in the Application Domain are returned. If the ID or NAME parameter matches, all Token Issuance Policy resources in that Application Domain are returned. In all cases, if both ID and NAME are present, ID will be used.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/tokenissuancepolicy`	POST	A Token Issuance Policy object is created by this method. The request is performed on the resource that is the parent of the object. The request body must contain a Token Issuance Policy resource that represents the object. A Token Issuance Policy object matching the request is created in the corresponding Application Domain.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/tokenissuancepolicy`	PUT	A Token Issuance Policy object is modified by this method. The request body must contain the Token Issuance Policy resource that represents the object. The Token Issuance Policy resource matching the ID or NAME query parameters is modified. The Token Issuance Policy object should belong to an Application Domain matching the APPDOMAINID or APPDOMAIN query parameter. If query parameters are not specified, the Token Issuance Policy matching the ID or NAME parameter will be modified. The Token Issuance Policy should belong to the Application Domain specified in the Application Domain Name attribute. If both ID and NAME are present, ID value will be used.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/tokenissuancepolicy`	DELETE	A Token Issuance Policy object is deleted by this method. The Token Issuance Policy matching the ID or NAME query parameter, in the Application Domain specified in the APPDOMAINID or APPDOMAIN query parameter, is deleted.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/resource`	GET	A Resource object is retrieved by this method. The resources that represents the Resource object is returned. This representation contains the matching Resource resource attributes and their values. Valid query parameters ID or NAME, and APPDOMAINID or APPDOMAIN. If an APPDOMAINID or APPDOMAIN parameter is not specified, a status code 424 is returned with the appropriate message. If an ID or NAME query parameter is not specified, all Resource resources in that Application Domain are returned. If the ID or NAME parameter matches, the matching Resource resource in the Application Domain is returned. In all cases, if both ID and NAME are present, ID will be used.

Table 5-5 (Cont.) Access Manager Policy Resources Summary

Resource	Method	Description
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/resource`	POST	A Resource object is created by this method. The request is performed on the resource that is a parent of the object. A Resource object matching the request is created in the corresponding Application Domain.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/resource`	PUT	A Resource object is modified by this method. The request body must contain the Resource resource that represents the object. The Resource matching ID or NAME query parameters is modified. The Resource should belong to an Application Domain matching the APPDOMAINID or APPDOMAIN query parameter. If query parameters are not specified, the Resource object matching the ID or NAME specified will be modified. The Resource should belong to the Application Domain specified in the Application Domain Name attribute. If both ID and NAME are present, the ID value will be used.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/resource`	DELETE	A Resource object is deleted by this method. The Resource object matching the ID or NAME query parameters, in the Application Domain in the APPDOMAINID or APPDOMAIN query parameters, is deleted.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/authzpolicy`	GET	An Authorization Policy object is retrieved by this method. The resource that represents the Authorization Policy object is returned. This representation contains the matching Authorization Policy resource attributes and their values. Valid query parameters are ID or NAME, and APPDOMAINID or "appdomain". If an appdomainid or APPDOMAIN parameter is not specified, a status code 424 is returned with the appropriate message. If an ID or NAME parameter is not specified, all Authorization Policy resources in that Application Domain are returned. If the ID or NAME parameter matches, the matching Authorization Policy resource in the Application Domain is returned. In all cases, if both ID and NAME are present, ID will be used.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/authzpolicy`	POST	An Authorization Policy object is created by this method. The request is performed on the resource that is the parent of the object. An Authorization Policy object matching the request is created in the corresponding Application Domain.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/authzpolicy`	PUT	An Authorization Policy object is modified by this method. The request body must contain the Authorization Policy resource that represents the object. The Authorization Policy resources that matching the ID or NAME query parameter is modified. The Authorization Policy should belong to an Application Domain matching the APPDOMAINID or APPDOMAIN query parameter. If query parameters are not specified, the Authorization Policy matching the ID or NAME parameter will be modified. The Authorization Policy should belong to the Application Domain specified in the Application Domain Name attribute. If both ID and NAME are present, ID value will be used.

Table 5-5 (Cont.) Access Manager Policy Resources Summary

Resource	Method	Description
oam/services/rest/11.1.2.0.0/ssa/policyadmin/authzpolicy	DELETE	An Authorization Policy object is deleted by this method. The Authorization Policy matching the ID or NAME query parameters, in the Application Domain specified in APPDOMAINID or APPDOMAIN query parameters, is deleted.
oam/services/rest/11.1.2.0.0/ssa/policyadmin/hostidentifier	GET	A Host Identifier object is retrieved by this method. The resource that represents the Host Identifier object is returned. This representation contains the matching Host Id en tier resource attributes and their values. Valid query parameters are ID or NAME. If a query parameter is not specified, all the Host Identifier resources are returned. If the ID or NAME parameter matches, the matching Host Identifier resource is returned.
oam/services/rest/11.1.2.0.0/ssa/policyadmin/hostidentifier	POST	A Host Identifier object is created by this method. The request is performed on the resource that is the parent of the object. A Host Identifier object matching the request is created.
oam/services/rest/11.1.2.0.0/ssa/policyadmin/hostidentifier	PUT	A Host Identifier object is modified by this method. The request body must contain the Host Identifier resource that represents the object. The Host Identifier resource matching the ID or NAME query parameters is modified. If query parameters are not specified, the Host Identifier matching the ID or NAME parameter will be modified. If both ID and NAME are present, ID value will be used.
oam/services/rest/11.1.2.0.0/ssa/policyadmin/hostidentifier	DELETE	A Host Identifier object is deleted by this method. The Host Identifier matching the ID or NAME query parameter is deleted.
oam/services/rest/11.1.2.0.0/ssa/policyadmin/resourcetype	GET	A Resource Type object is retrieved by this method. The resource that represents the Resource Types object is returned. This representation contains the matching Resource Type resource attributes and their values. Valid query parameters ID or NAME. If a query parameter is not provided, all Resource Type resources are returned. If the query parameter id or name matches, the matching Resource Type is returned.
oam/services/rest/11.1.2.0.0/ssa/policyadmin/resourcetype	POST	A Resource Type object is created by this method. The request body is performed on the parent of the object. A Resource Type object matching this request is created.
oam/services/rest/11.1.2.0.0/ssa/policyadmin/resourcetype	PUT	A Resource Type object is modified by this REST method. The request body must contain the Resource Type resource that represents the object. The Resource Type resource matching ID or NAME query parameter is modified. If query parameters are not specified, the Resource Type matching the ID or NAME parameter will be modified. If both ID and NAME are present, ID value will be used.
oam/services/rest/11.1.2.0.0/ssa/policyadmin/resourcetype	DELETE	A Resource Type object is deleted by this method. The Resource Type matching the NAME or ID query parameter is deleted.

Table 5-5 (Cont.) Access Manager Policy Resources Summary

Resource	Method	Description
`oam/services/rest/ 11.1.2.0.0/ssa/ policyadmin/authnscheme`	GET	An Authentication Scheme object is retrieved by this method. The resource that represents the Authentication Schemes object is returned. This representation contains the matching Authentication Scheme resource attributes and their values. Valid query parameters are ID or NAME. If a query parameter is not specified, all Authentication Scheme resources are returned. If the query parameter ID or NAME matches, the matching Authentication Scheme is returned.
`oam/services/rest/ 11.1.2.0.0/ssa/ policyadmin/authnscheme`	POST	An Authentication Scheme object is created by this method. The request is performed on the resource that is the parent of the object. An Authentication Scheme object matching the request is created.
`oam/services/rest/ 11.1.2.0.0/ssa/ policyadmin/authnscheme`	PUT	An Authentication Scheme object is modified by this method. The request body must contain the Authentication Scheme resource that represents the object. The Authentication Scheme resource matching the ID or NAME query parameter is modified. If query parameters are not specified, the Authentication Scheme matching ID or NAME parameter will be modified. If both ID and NAME are present, ID value will be used.
`oam/services/rest/ 11.1.2.0.0/ssa/ policyadmin/authnscheme`	DELETE	An Authentication Scheme object is deleted by this method. The Authentication Scheme matching the NAME or ID query parameter is deleted.
`/oam/services/rest/ 11.1.2.0.0/ssa/ policyadmin/ application.wadl`	GET	Web Application Definition Document is generated. It describes the REST services provided. The document contains a stylesheet reference that renders HTML content.

5.4 Client Tooling

Two XML schemas are available for generating client side POJOs, which represent the RESTful service resources:

- For the policyadmin service, the schema is oam-policyadmin-11.1.2.0.0.xsd.
- For the token service, the schema is oam-token-11.1.2.0.0.xsd.

To generate the client side object, run the JAXB command `xjc` (part of the JDK) as follows:

```
xjc  [-p package-name] oam-policyadmin-11.1.2.0.0.xsd
```

This command generates the Java POJO objects for the RESTful resources, which can be used in the client side Java code. These objects can be converted back to XML using JAXB and can then be sent to the REST server over HTTP.

For more information about JAXB, see `http://jaxb.java.net`. For more information about building clients for Jersey-based REST server, see `http://jersey.java.net/`.

5.5 cURL Command Examples

The following examples are provided as reference.

- Retrieve Application Domains cURL Command

5.6 Retrieve Application Domains cURL Command

```
$ curl -u USER:PASSWORD http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/
policyadmin/appdomain
```

The following is sample output from this cURL command.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><ApplicationDomains>
<ApplicationDomain>
    <id>759463e3-2b63-4e38-893c-00d5da479719</id>
    <name>IAM Suite</name>
    <description>Policy objects enabling OAM Agent to protect deployed IAM Suite
applications</description>
</ApplicationDomain>

<ApplicationDomain>
    <id>69f6be9b-f000-48db-9b6d-df4724cc0bd9</id>
    <name>Fusion Apps Integration</name>
    <description>Policy objects enabling integration with Oracle Fusion Applications</
description>
</ApplicationDomain>
```

5.7 Create a New Application Domain cURL Command

```
curl -u weblogic:welcome1 -H "Content-Type: application/xml" --request POST --data
"@/tmp/cr.appdomain.xml" http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/
policyadmin/appdomain
```

The following is a sample input file for this cURL command.

```
<ApplicationDomain>
    <name>Appdomain1</name>
    <description>test application domain</description>
</ApplicationDomain>
```

The following is sample output from this cURL command.

```
http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/policyadmin/appdomain?
id=fa60e312-fe65-4aa8-aace-1735a39c4058
```

5.8 Retrieve All Authentication Schemes cURL Command

```
curl -u USER:PASSWORD http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/
policyadmin/authnscheme
```

The following is sample output from this cURL command.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AuthenticationSchemes>
  <AuthenticationScheme>
      <id>aa84b589-7f16-4b3a-942c-ba51b3ab6de5</id>
      <name>KerberosScheme</name>
      <description>Kerberos Scheme</description>
      <authnModuleName>Kerberos</authnModuleName>
      <authnSchemeLevel>2</authnSchemeLevel>
      <challengeMechanism>WNA</challengeMechanism>
      <ChallengeParameters>
          <challengeParameter>
              <key>spnegotoken</key>
              <value>string</value>
          </challengeParameter>
          <challengeParameter>
              <key>challenge_url</key>
              <value>/oam/CredCollectServlet/WNA</value>
          </challengeParameter>
      </ChallengeParameters>
      <challengeRedirectURL>/oam/server/</challengeRedirectURL>
  </AuthenticationScheme>
</AuthenticationSchemes>
```

5.9 Create an Authentication Scheme cURL Command

```
curl -u weblogic:welcome1 -H "Content-Type: application/xml" --request POST --data
"@/tmp/cr.authnscheme.xml" http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/
policyadmin/authnscheme
```

The following is a sample input file.

```
<AuthenticationScheme>
    <name>TestAuthnScheme</name>
    <description>test authn scheme</description>
    <authnModuleName>TestModule1</authnModuleName>
    <authnSchemeLevel>2</authnSchemeLevel>
    <challengeMechanism>WNA</challengeMechanism>
    <ChallengeParameters>
        <challengeParameter>
            <key>spnegotoken</key>
            <value>string</value>
        </challengeParameter>
        <challengeParameter>
            <key>challenge_url</key>
            <value>/oam/CredCollectServlet/WNA</value>
        </challengeParameter>
    </ChallengeParameters>
    <challengeRedirectURL>/oam/server/</challengeRedirectURL>
</AuthenticationScheme>
~
```

The following is sample output from this cURL command.

```
http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/policyadmin/authnscheme?
id=acb1fa95-f780-4091-be88-2e96cf5bbd49
```

5.10 Retrieve a Specific Authentication Scheme cURL Command

```
curl -u USER:PASSWORD http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/
policyadmin/authnscheme?name=KerberosScheme
```

The following is sample output from this cURL command.

```
<AuthenticationScheme>
      <id>aa84b589-7f16-4b3a-942c-ba51b3ab6de5</id>
      <name>KerberosScheme</name>
      <description>Kerberos Scheme</description>
      <authnModuleName>Kerberos</authnModuleName>
      <authnSchemeLevel>2</authnSchemeLevel>
      <challengeMechanism>WNA</challengeMechanism>
      <ChallengeParameters>
          <challengeParameter>
              <key>spnegotoken</key>
              <value>string</value>
          </challengeParameter>
          <challengeParameter>
              <key>challenge_url</key>
              <value>/oam/CredCollectServlet/WNA</value>
          </challengeParameter>
      </ChallengeParameters>
      <challengeRedirectURL>/oam/server/</challengeRedirectURL>
  </AuthenticationScheme>
```

5.11 Retrieve All Resources in an Application Domain cURL Command

```
curl -u USER:PASSWORD http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/
policyadmin/resource?appdomain="IAM Suite"5
```

5.12 Create a Resource in an Application Domain cURL Command

```
curl -u weblogic:welcome1 -H "Content-Type: application/xml" --request POST --data
"@/tmp/cr.resource.xml" http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/
policyadmin/resource?appdomain="AppDomain1"
```

5.13 Retrieve All Policies in an Application Domain cURL Command

```
curl -u USER:PASSWORD http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/
policyadmin/authnpolicy?appdomain="IAM Suite"
```

6

Developing an Application to Manage Impersonation

Access Manager impersonation support enables a user to designate other users to act on their behalf within a constrained time frame. While impersonation grants are natively supported by Access Manager, you will need to develop a custom user interface or modify an existing interface in order to manage impersonation grants. This chapter provides information about enabling impersonation and developing a custom user interface. It includes the following sections:

- About the Impersonation feature in Access Manager
- Configuring Impersonation Support
- Testing SSO Login and Impersonation

> **Note:**
>
> See *Integrating Oracle ADF Applications with Access Manager SSO*.

6.1 About the Impersonation feature in Access Manager

The Access Manager user impersonation feature enables a user to perform operations and access resources on behalf of another user. Impersonation grants, specified with a user identifier and start and end time, are required for a user to be able to impersonate another.

The following topics are discussed:

- About Impersonation Terminology
- Understanding Impersonation Concepts
- About Impersonation Grant Syntax
- Understanding Impersonation Trigger Invocation Using the SSO Service
- Triggering Impersonation Without API Abstraction
- Impersonator Identity Communication During Impersonation Sessions

6.1.1 About Impersonation Terminology

Table 6-1 introduces common Access Manager impersonation concepts and terms.

Table 6-1 Impersonation Terminology

Term	Definition
Impersonator	A user who acts on another user's behalf.
Impersonatee	The user who is being impersonated by another.

Table 6-1 (Cont.) Impersonation Terminology

Term	Definition
Impersonation grant	Security metadata created by the impersonatee to designate a particular impersonator to impersonate her within a specified time window.
Impersonation trigger	An act of an impersonator choosing to initiate an impersonation session on behalf of another user.
Access Manager impersonation session	A distinct type of Access Manager session that can be distinguished from regular user session by the target application.
Impersonation consent	A consent given by the impersonator to acknowledge the awareness that Access Manager impersonation session is in effect.

6.1.2 Understanding Impersonation Concepts

Access Manager user impersonation support allows an end user (the impersonatee) to designate one or more users (impersonators) to act on their behalf within a constrained window of time. This information is collected using a custom user interface you develop, and persisted as a set of impersonation grants in the user directory.

The impersonator, while holding an authenticated session and interacting with a custom user interface, may choose to initiate an impersonation session on behalf of another named user. Access Manager performs required authorization checks to ascertain that the impersonator is allowed to impersonate the impersonatee. If allowed, the impersonation session is created.

The Access Manager-protected application behaves as if the impersonated user was accessing it. The application can determine whether the user is the impersonator or the impersonatee.

The impersonation session terminates when the impersonator chooses to do so through the application user interface. The impersonator will return to their regular user session and be able to access the application as himself once again. The impersonator is not allowed to switch the impersonatee user during his impersonation session (that is, nested or recursive impersonation is not allowed).

Access Manager provides the runtime enforcement of the impersonation semantics as described above, while all of the user interface aspects and associated metadata (impersonation grant) lifecycle are provided by your custom interface. The integration between Access Manager and a custom user interface can be codified in terms of the following three interfaces (touch points):

- Impersonation grant syntax, persistence, and lifecycle
- Impersonation trigger invocation
- Impersonator identity communication during Access Manager impersonation session

6.1.3 About Impersonation Grant Syntax

The following two impersonation grants are part of the `orclIDXPerson` object class:

- `orclImpersonationGrantee`: If this attribute contains grants for a user, then that user can impersonate the current user. This is the attribute checked by OAM Server during an impersonation request.

- `orclImpersonationGranter`: If this attribute contains grants for a user, then that user can be impersonated by the current user. This attribute is not used to enforce impersonation, it is used to start the impersonation session from the application.

Impersonation grants of an impersonatee are persisted in the user's record in the LDAP directory as a multi-valued attribute. Each of the values represents a specific grant to a named impersonator and a specified time window. Each value of the multi-valued attribute is a composite string, with individual fields delineated by a separator character. For this release, Oracle Identity Directory is also supported when using impersonation feature.

You can create or modify a custom user interface to enable users to create, view, update, or delete impersonation grants within their user profile. The user interface must be constructed to persist impersonation grants in the designated LDAP directory in the multi-valued attribute named `orclImpersonationGrantee`. The format of individual values is *<Impersonator orclGUID> | <begin LDAP timestamp> | <end LDAP timestamp>* . For example:

```
orclImpersonationGrantee: xyz123abcd|20100604224517Z|20100604234517Z;
klmn980nopr|20100604224517Z|20100604234517Z
```

In the following example, assume:

- Impersonator: *jdoe*
- Impersonatee: *lsmith*

jdoe is trying to impersonate *lsmith*. The following command can be used to obtain the `OrclGuid` of the impersonator (*jdoe*):

```
ldapsearch -h <hostname> -w <password>  -p <port> -D"cn=orcladmin" -
b"dc=us,dc=example,dc=com"  "cn=jdoe" orclguid
```

For example, LDAP search for `orclguid`:

```
ldapsearch -h myhost1.us.example.com -w welcome1 -p 16890 -b"dc=us,dc=example,dc=com" -
D"cn=orcladmin"  "cn=jdoe" orclguid
version: 1
```

where:

- `dn: cn=jdoe,cn=Users,dc=us,dc=example,dc=com`
- `orclguid: A14BEB42E822D605E040E50AB29327E7`

For example, LDAP search for `orclImpersonationGrantee`:

```
ldapsearch -h host1.us.example.com -w welcome1 -p 16890 -b"dc=us,dc=example,dc=com" -
D"cn=orcladmin"  "cn=lsmith" orclImpersonationGrantee
version: 1
```

where:

- `dn: cn=lsmith,cn=Users,dc=us,dc=example,dc=com`
- `orclImpersonationGrantee: A14BEB42E822D605E040E50AB29327E7|20100324163000Z|
  20120524172000Z`

Add this value to the `orclImpersonationGrantee` entry to impersonatee user in OID as follows:

```
A14BEB42E822D605E040E50AB29327E7|20100324163000Z|20120524172000Z
```

> **Note:**
>
> No spaces are permitted in the list of individual values.

Object class and attribute definition for this attribute must be bootstrapped in the LDAP server's schema. OID 11.1.1.3 and later contains the necessary object class.

Access Manager retrieves impersonation grants of a given impersonatee when an impersonator attempts to create an impersonation session. However, if the grant doesn't exist for the given impersonator or if the current time is not within the time window of any such grants, impersonation session creation fails. Access Manager does not otherwise read or modify the grants within user profiles.

Subsequent revocation of the impersonation grant (for example, by modifying the `orclImpersonationGrantee` attribute) that authorized the impersonation session will not affect the impersonation sessions still in progress.

6.1.4 Understanding Impersonation Trigger Invocation Using the SSO Service

An authenticated user can select to impersonate another user. The user interface to select which user to impersonate is provided by an application. After the information has been collected, the application invokes the impersonation trigger.

This can be done by invoking one of the methods in the SSO Service as shown in Required Method to Abstract Triggering Mechanism Using SsoService API or directly by redirecting the user's browser to Access Manager trigger URLs.

For more information about the SSO Service, See Configuring the Identity Provider, Property Sets, and SSOin *Oracle Fusion Middleware Application Security Guide*.

Required Method to Abstract Triggering Mechanism Using SsoService API illustrates the methods required to use the SSO Service to abstract the specifics of the triggering mechanism (preferred).

In this example `props` contains `IMP_USER_ID` of the impersonatee, `SUCCESS_URL`, `FAILURE_URL`, and `TARGET_URL` similar to login/logout/auto-login API of the SSOService. Abbreviated SsoService API Triggering Example shows an abbreviated example.

jps-config.xml With Changes For imp.begin.url and imp.end.url provides a snippet from jps-config.xml showing the configuration changes needed (`imp.begin.url` and `imp.end.url` properties):

Example 6-1 Required Method to Abstract Triggering Mechanism Using SsoService API

```
void beginImpersonation(HttpServletRequest request, HttpServletResponse response,
Map<String, ?> props) throws SsoServiceException
void endImpersonation(HttpServletRequest request, HttpServletResponse response,
Map<String, ?> props) throws SsoServiceException
```

Example 6-2 Abbreviated SsoService API Triggering Example

```
import oracle.security.jps.JpsException;
import oracle.security.jps.service.JpsServiceLocator;
import oracle.security.jps.service.ServiceLocator;
```

```
import oracle.security.jps.service.sso.SsoService;

public void doGet (HttpServletRequest req, HttpServletResponse res)
 throws ServletException, IOException

{

  try {
   ServiceLocator serviceLocator = JpsServiceLocator.getServiceLocator();
   SsoService sso = (SsoService)serviceLocator.lookup(SsoService.class);

   Map m = new HashMap();

   m.put(SsoService.SUCCESS_URL, "https://login01.example.com:7777/appl2.html");
   m.put(SsoService.FAILURE_URL, "https://login01.example.com:7777/fail.html");
   m.put(SsoService.IMP_USER_ID, "mcooper");

    sso.beginImpersonation(req, res, m);


    [....]

   m.put(SsoService.TARGET_URL, "https://login02.example.com:8080/
   normalSession.html");
   sso.endImpersonation(req, res, m);

  } catch(JpsException jpse) {
    jpse.printStackTrace();
  }
}
```

Example 6-3 jps-config.xml With Changes For imp.begin.url and imp.end.url

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jpsConfig xmlns="http://xmlns.example.com/oracleas/schema/11/jps-config-11_1.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.example.com/oracleas/schema/11/jps-config-11_1
1.xsd">
    <property value="off" name="oracle.security.jps.jaas.mode"/>
    <propertySets>
        <propertySet name="saml.trusted.issuers.1">
            <property value="www.example.com" name="name"/>
        </propertySet>

[...]

        <propertySet name="props.auth.uri.0">
            <property value="/oamsso/logout.html" name="logout.url"/>
            <property value="https://login01.example.com:7777/oam/server/impersonate/end
              name="imp.end.url"/>
            <property value="https://login01.example.com:7777/oam/server/impersonate/
start
              name="imp.begin.url"/>
            <property value="/${app.context}/adfAuthentication" name="login.url
             .BASIC"/>
            <property value="/${app.context}/adfAuthentication" name="login.
             url.ANONYMOUS"/>
            <property value="/${app.context}/adfAuthentication" name="login.
             url.FORM"/>
        </propertySet>
        <propertySet name="props.auth.level.0">
            <property value="0" name="type-level:ANONYMOUS"/>
```

```
            <property value="1" name="type-level:BASIC"/>
            <property value="2" name="type-level:FORM"/>
        </propertySet>
    </propertySets>

[...]
```

6.1.5 Triggering Impersonation Without API Abstraction

To invoke the Access Manager impersonation triggers directly, without the use of an API abstraction, the redirection to Access Manager maintained trigger end point has to contain a specification of query parameters for `userid`, `success_url`, and `failure_url`.

The `userid` field carries the Impersonatee's userid, the success_url/failure_url is where the impersonator's browser should be pointed to after the impersonation session has been created or failed to be created, respectively. The URLs provided must include protocol and host:port information, as shown in Triggering Impersonation Without API Abstraction.

To terminate the impersonation session and restore the original impersonator's Access Manager session, the user interface must force a browser redirect to an Access Manager maintained end point, and provides the target URL for the impersonator to come back to shown in Restore Original Impersonator's Session. Use of `failure_url` is optional.

Example 6-4 Triggering Impersonation Without API Abstraction

```
https://login.example.com/oam/server/impersonate/start?userid=impersonatee
userid&success_url=SuccessRedirect URL&failure_url=FailureRedirect URL
```

Example 6-5 Restore Original Impersonator's Session

```
https://login.example.com/oam/server/impersonate/end?end_url=TargetRedirect
URL&failure_url=FailureRedirect URL
```

6.1.6 Impersonator Identity Communication During Impersonation Sessions

Use the header names provided for communicating the identity of the impersonator to the downstream application.

Headers For Identity Information provides the header names for communicating the identity of the impersonator to the downstream application. The WebGate uses an additional HTTP header injected into the request. The interested application may detect that the Access Manager impersonation session is in progress by inspecting the HTTP headers of inbound requests.

Table 6-2 Headers For Identity Information

Header Name	Description
OAM_IMPERSONATOR_USER	The header name that carries the impersonator userID.
OAM_REMOTE_USER	The header that carries the end userID, which is the same as with a standard Access Manager user session.

6.2 Configuring Impersonation Support

Use the information in the topics provided to enable the impersonation feature.

The impersonation feature is not enabled by default. You enable the impersonation feature by either configuring `oam-config.xml` or by using the `idmConfigTool` command. The following sections contain details.

- Configuring Impersonation Using oam-config.xml
- Configuring Impersonation Using idmConfigTool
- Configuring the Authentication Scheme

6.2.1 Configuring Impersonation Using oam-config.xml

The impersonation feature for the OAM Server is enabled by configuring the `oam-config.xml` file. Example 6-6 shows the relevant section of the file and the parameters that can be set. `EnableImpersonation` must be set to 'true' to enable impersonation. The default setting is 'false'.

Example 6-6 Enabling Impersonation Feature in oam-config.xml

```
<Setting Name="ImpersonationConfig" Type="htf:map">
<Setting Name="EnableImpersonation" Type="xsd:boolean">true</Setting>
    <Setting Name="UserAttributeName"
    Type="xsd:string">orclImpersonationGrantee</Setting>
    <Setting Name="ErrorPage" Type="xsd:string">/pages/servererror.jsp</Setting>
</Setting>
```

Impersonation requires that the login request context be preserved either with a `serverRequestCacheType` setting of COOKIE or BASIC. The default setting is COOKIE. This OAM Server parameter can be set in the `oam-config.xml` file or using the `configRequestCacheType` WLST command. For more information about the `configRequestCacheType` command, see *configRequestCacheType* in *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference for Identity and Access Management*.

6.2.2 Configuring Impersonation Using idmConfigTool

Use the procedure to configure the impersonation feature.

Follow this procedure to configure the impersonation feature using `idmConfigTool`. For information about the `idmConfigTool` command and input parameters, See Using the idmConfigTool Command in *Oracle Fusion Middleware Integration Guide for Oracle Identity Management Suite*.

1. Use `idmConfigTool` with the `-prepareIDStore` command to seed the Identity Store with the users required by Access Manager.

   The command syntax is `./idmConfigTool.sh -prepareIDStore mode=OAM input_file=`*`input_parameters`*.

2. Configure the impersonation feature using `idmConfigTool` with the `configOAM` command with the parameter `OAM11G_IMPERSONATION_FLAG:true`.

   The command syntax is `./idmConfigTool.sh -configOAM input_file=`*`input_parameters`*.

3. Define the impersonator grant permissions by providing the session timestamps for the impersonation session duration.

The format of individual values is *<Impersonator orclGUID> | <begin LDAP timestamp> | <end LDAP timestamp>*. No spaces are permitted. For example, in OID add similar timestamp values to `orclImpersonationGrantee` entry as shown in the following:

```
83295E092B2F9FD4E040E50AEBB91998|20100604224517Z|
20110604224517Z;90FE8C8083CEBC1FE040E50AEBB9176A|20100704224517Z|20110604224517Z
```

where:

- The first block is the GUID of the impersonator. As shown here, `83295E092B2F9FD4E040E50AEBB91998` is the GUID of the impersonator.
- The second block is the timestamp start date.
- The third block is the timestamp end date.

4. Submit the data to the LDAP server.

6.2.3 Configuring the Authentication Scheme

Use these steps to set the authentication scheme to LDAPScheme.

For impersonation support the authentication scheme for the protected application must be set to LDAPScheme. This must be done before initiating an impersonation session. To set the authentication scheme to LDAPScheme:

1. In the Oracle Access Management Administration Console, go to the **Policy Configuration** tab, **App Domain**, **Authentication Policies**, **Protected Resource Policy**.
2. From the Authentication Scheme list, select **LDAPScheme**.

For more information about the LDAPScheme, See *Managing Authentication Schemes*.

6.3 Testing SSO Login and Impersonation

Use these steps to test the impersonation set up according to your environment and your custom user interface.

The steps to test impersonation set up will vary according to your environment and your custom user interface. The following general advice is provided as an example of the steps to take. Adjust the steps as needed for your environment.

1. Log in to Oracle Access Management using your own userID and credentials.
2. Access a resource for which you have authorization to verify that Access Manager is working with your credentials as expected.
3. Start your impersonation session.
4. In the impersonation confirmation form that appears, enter your own (that is, impersonator's) password and click Submit to provide impersonation consent.
5. In the same browser, access a resource for which the impersonated user has authorization.
6. Confirm the Impersonating column in the Access Manager Session Management Page displays true.

   For more information, See Session Management Controls in *Oracle Fusion Middleware Administering Oracle Access Management*.

7. Confirm that HTTP header variables (OAM_REMOTE_USER and OAM_IMPERSONATOR_USER) are set in the impersonation session by using a script or Perl program that will print header variable.

   For more information, See Impersonator Identity Communication During Impersonation Sessions.

8. Terminate your impersonation session.

9. Confirm that OAM_REMOTE_USER is set to user before impersonation trigger, and OAM_IMPERSONATOR_USER HTTP header variable is empty or blank, by using a script or Perl program that will print header var.

7

Customizing Oracle Mobile Authenticator

The Oracle Mobile Authenticator (OMA) is a mobile device application that uses One Time Password (OTP) and push notifications to authenticate users without incurring the cost of hardware tokens or SMS charges. This application provides password less authentication. The supported platforms are iOS, Android and Windows universal app.
This chapter contains procedures that you can use to brand the Oracle Mobile Authenticator to represent your company's logo and colors. It contains the following sections.

7.1 About Oracle Mobile Authenticator and Customization

The Oracle Access Management Adaptive Authentication Service offers the ability to add multiple steps to the user authentication process. This additional security may be enforced by adding a OTP step, or an Access Request (Push) Notification step after initial user authentication. In certain cases, the enforcement involves the use of the Oracle Mobile Authenticator (OMA), a mobile device app that uses Time-based One Time Password and push notifications to authenticate users within the additional second factor authentication scheme. For more details on the Adaptive Authentication Service and how it works with the OMA, See Managing the Adaptive Authentication Service and Oracle Mobile Authenticator in *Fusion Middleware Administering Oracle Access Management*.

As the Administrator, you can customize the following features in the OMA application and distribute the application for internal use.

- Application logo and images
- String resources
- End User License Agreement (EULA), Private Policy and Help of the app
- Name, Identity and Version information of the app

7.2 Customizing Oracle Mobile Authenticator on iOS

The Oracle Mobile Authenticator (OMA) is distributed as a ZIP of xcarchive which can be used to customize the application. Xcode is used to sign the xcarchive and generate the IPA(App which can be installed on devices).

> **Note:**
>
> To get the Oracle Mobile Authenticator (OMA) customizing app, contact Oracle support to get access to an unsigned mobile archive.

Before you Begin

The following resources are required to customize OMA:

- MAC with Xcode installed

Customize iOS OMA app

The iOS OMA app can be customized in 3 steps.

1. Extract/unpack xcarchive

2. Customize xcarchive

3. Sign xcarchive from Xcode using the certificates generated by your company. This action is required to install the app on the devices, to setup the notification services and to upload the app to the app store.

Extract/Unpack xcarchive

OracleAuthenticator.xcarchive is provided to the user.

1. Right click on *OracleAuthenticator.xcarchive* and navigate to `open package contents` --> `Products` --> `Applications` -->`Authenticator`
2. Right click on `Authenticator`, and select `open package contents`.

Customize xcarchive

1. Splash Screen

   This screen is visible for small amount of time while launching the app.

   - There are 8 images starting with *LaunchImage...* . Create customized images using the image sizes provided. Sizes are mentioned as a part of the image name.

2. App Icon

   This is the image shown in the Home screen or Spring board.

   - There are 12 images starting with *AppIcon...* Create customized images using the image sizes provided. Sizes are mentioned as a part of the image name.
   - Replace the image with the same name.

3. OMA logo

   These images are shown as the default icons of the account.

   - There are two images with different foreground colors with transparent background.

     Image 1. oma_blue.png with size 96*96 pixels.

     Image 2. oma_white.png with size 205*270 pixels.

   - Replace the customized files with the same name and png format.

     . Note that changing the Image size aspect ratio may at times cause the user interface to look distorted. Verify that the image size/aspect ratio looks as required, before it is finalized.

4. Other company icons.

   Each account has an image associated with it. The account image can be selected among the available images while adding the account during the manual entry of shared secret or, the existing account images can be changed in the edit screen.

- others.png with 140*140 pixels.

5. App display name.

   This is the name of the App which is shown in the home screen.

   - Open Info.plist file
   - Edit the value of the field `Bundle display name`. The default value is *Authenticator*.

6. EULA, Private Policy and Help.

   - Inside en.lproj there are files for each file(eula.txt, help.html and privacy.html). These files can be edited. The *eula.txt* is not localized.

   > **Note:**
   >
   > Files have to be updated in all the Localized languages. Localized language folder will have ".lproj" as extension.

7. 3rd party company list with images.

   These are the 3rd party companies which could be used to configure the account through OMA. This page can be viewed while creating a new account through a manual entry of Key or Editing the account information. The company name and image would be shown in the same order.

   - Open *CompanyList.json* file, The company name and image is shown in the same order. Any Item can be deleted. Available image name can be updated in *IconName*.

8. If EULA needs to be displayed on first launch.

   - Open *CustomizableFlags.plist*, change the value of `shouldEULAShownOnFirstLaunch` to `YES` or `NO`. Default is `YES`.

9. App version can be changed from (Bundle versions string, short) in info.plist.

10. Application name is shown inside the OMA in various places. Only the application name can be customized and localization is not supported. The default application name is `Oracle Mobile Authenticator`. This can be customized by changing the field `CompanyName` in info.plist file.

Sign xcarchive

- Get the *App Bundle Identifier* of the certificate which is generated in apple.developer.com. Certificates related to this bundle Identifier is used to sign the application. For an example the Bundle Identifier looks like com.acme.authenticator.
  - The customer generates the certificates which is used to sign the application.
  - They use their apple developer's account to generate the certificates.
  - The certificates have a unique string which is used to distinguish between the apps.
  - The unique string is called as Bundle Identifier that is used here.

  For more information , See https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingCertificates/MaintainingCertificates.html
- Open *info.plist* file, Update the field `Bundle identifier` with the *App Bundle Identifier*.

Save all changes, double click on the xcarchive folder. This will launch Xcode.

To sign the app, Follow the instruction given in https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html.

To sign and export iPA from Xcarchive, follow the steps given in https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/TestingYouriOSApp/TestingYouriOSApp.html

7.3 Customizing Oracle Mobile Authenticator on Android

The Oracle Mobile Authenticator is shipped to customers as an Android application package (.apk). The `apktool` is a tool that allows you to decompile an Android application, modify it and then rebuild it with the modifications. See the following sections for information on using the `apktool`.

- Using apktool to Customize Oracle Mobile Authenticator
- Customizing Options for Oracle Mobile Authenticator Android app

7.3.1 Using apktool to Customize Oracle Mobile Authenticator

The `apktool` installation and usage guide can be accessed from the apktool project home at https://ibotpeaches.github.io/Apktool/. The following sample command is used to decompile an Android app package.

```
apktool d OracleMobileAuthenticator-android-release_branch_21_03_2017-170403.181503.apk
```

This next sample command is used to recompile the updated contents of Android app package. It will create a signed version of the customized app.

```
apktool b OracleMobileAuthenticator-android-release_branch_21_03_2017-170403.181503
```

7.3.2 Customizing Options for Oracle Mobile Authenticator Android app

The following sections document the customizing options for the Oracle Mobile Authenticator Android app.

- Changing Application Icons
- Modifying the Application Name and Text
- Editing 3rd party company list with images
- Modifying EULA to be shown on first launch
- Modifying the Version and Code Number
- Modifying the Package Name
- Signing the Application
- Customizing Copyright Details
- Customizing Notifications and Enrollment Types

7.3.2.1 Changing Application Icons

For better UX control and multiple screen support, Android provides separate folders to better organize drawables for each screen type. (As an example the drawable-hdpi is for high pixel density devices.) Android application icons are located in the `res/` folder.

Based on the requirement the OMA application icons can also be updated in the corresponding drawable folder. In order to customize the application icons, replace the old icons with the new icons **without changing the icon name**.

The icon must be replaced in all the following folders:

- res\drawable-mdpi-v4
- res\drawable-hdpi-v4
- res\drawable-xhdpi-v4
- res\drawable-xxhdpi-v4
- res\drawable-xxxhdpi-v4

The following icons can be replaced with your customized icons with same size and name as that of original icons:

1. Launcher Icon: res\mipmap-mdpi-v4\ic_launcher.png
2. OMA logo There are two Images with different foreground colours with transparent background.

   Image 1. res\drawable-mdpi-v4\oma_logo.png

   Image 2. res\drawable-mdpi-v4\mfa_icon.png
3. Other company icons

   Other company icon is a default company icon shown for 3rd party companies. It can be changed in companylist.json

   res\drawable-mdpi-v4\others.png

7.3.2.2 Modifying the Application Name and Text

The name `Authenticator` can be customized by modifying the existing value of the string `app_name` in the `/res/values/strings.xml` file. Find the default value in the file as:

```
<string name="app_name">Authenticator</string>
```

Change this value to the preferred name and save; for example, *Acme Authenticator*. No special characters can be used.

```
<string name="app_name">Acme Authenticator</string>
```

The End-user License Agreement, Privacy and Help text can also be customized. To change the text, replace the original version of the file(s) with the new file(s) in the directory structure as specified below. Do not change the file name.

- End-user License Agreement: /res/raw/eula.txt
- Privacy: /res/raw/privacy.html
- Help: /res/raw/help.html

  Help is localized. To customize help, the help file needs to be changed in the localization folders.

> **Note:**
>
> When app name and strings are changed, Do make sure to change the corresponding translated strings. For example, res\values-ar, res\raw-ar

7.3.2.3 Editing 3rd party company list with images

Open res\raw\companylist.json file. The company name and image would be shown in the same order. Any item can be deleted. Available image name can be updated in "IconName".

7.3.2.4 Modifying EULA to be shown on first launch

Open `res\raw\prop.txt`. Change the value of `showEula` to `yes` . The values are treated as case-insensitive. Default value is `yes`.

7.3.2.5 Modifying the Version and Code Number

Modify the version and code number of the application by changing details in the `apktool.yml` located in the directory where the .apk file content has been de-compiled. (See Using apktool to Customize Oracle Mobile Authenticator) The apktool.yml file can be viewed and modified in any text editor. The versionCode and versionName parameters are located under the versionInfo property as illustrated in Example 7-1. In this example, the version name has been changed to test.xx.x.x from the default value 11.1.2.3.0.

Example 7-1 Changing the Android Version and Code Number

```
versionInfo:versionCode: '3'versionName: 'test.xx.x.x'
```

7.3.2.6 Modifying the Package Name

To modify the package name

1. Open AndroidManifest.xml present in the decompiled apk folder in Notepad++. "Find All" *<package name>* and replace with the new *package name* and set Directory as the location of the decompiled apk folder and filter as "*.*".

   Example: Find what: *oracle.idm.mobile.authenticator* Replace what : *example.idm.mobile.authenticator*

2. Similarly, Find All and Replace *package name* appended with '*L*' and '.' replaced with '/'

   Example : Find what :*Loracle/idm/mobile/authenticator* Replace what :*Lexample/idm/mobile/authenticator*

3. Go to smali folder in the decompiled apk folder and rename the folder names according to new package name.

4. Build the folder using apk tool. For example:

   ```
   apktool b OracleMobileAuthenticator-androidrelease_branch_21_03_2017-170403.181503
   ```

7.3.2.7 Signing the Application

Android requires that all apps be digitally signed before they can be installed. Android uses the certificate to identify the author of the app. The certificate does not need to be signed by a certificate authority so Android apps often use self-signed certificates. Additional details on this Android requirement and its process, including the procedure you can use to sign your apps,

are described at http://developer.android.com/tools/publishing/app-signing.html#signing-manually

7.3.2.8 Customizing Copyright Details

Follow the steps mentioned below to customize copyright details:

1. Decompile the `OMA.apk`.
2. Navigate to `OracleMobileAuthenticator/res/values/strings.xml`.
3. Update the copyright text written between the `<string>` tag as needed:

```
<string name="copyright">Copyright © 2013-2022, Oracle and/or its
affiliates. All rights reserved.</string>
```

7.3.2.9 Customizing Notifications and Enrollment Types

You can now enable or disable Push and Pull notifications and manage enrollment types.

After decompiling the `OMA.apk` navigate to the file location `OracleMobileAuthenticator/src/main/res/raw/prop.txt` and update following properties based on the requirement.

Property	Description
`isEnablePushAndPullNotification`	You can enable or disable Push and Pull notifications by setting the value of `isEnablePushAndPullNotification` to true/false. Set the value to false to disable the Push/Pull notification. The default value is true.
`enrollmentType`	You can set the `enrollmentType` value to `qr_and_manual` to utilize both the QR code scanner and manual enrollment methods, or can skip the QR Scanner activity and add an account manually by setting the value to `manual`. The default value is `qr_and_manual`.
`hideFooter`	You can set the `hideFooter` value to true to prevent the footer from appearing at the bottom of the **Manual Add Account** screen. The default value is false, which displays the footer.

7.4 Customizing Oracle Mobile Authenticator on Windows

The Oracle Mobile Authenticator (OMA) is distributed as a ZIP package which can be used to customize the application.

> **Note:**
>
> To get the Oracle Mobile Authenticator (OMA) customizing app, contact Oracle support to get access to an unsigned mobile archive.

Before you Begin

The following resources are required to customize OMA:

- Windows 10 onwards
- VIsual Studio 2015 onwards

Customize Windows OMA app

The Windows OMA app can be customized in 3 steps.

1. Extract/unpack package

2. Customize package

3. Create/pack package

Extract/Unpack Package

Open Developer Command Prompt for VS2015 and run the following commands:

1. **MakeAppx** unbundle /o /p / *<full path of appxbundle* > /d *<full folder path where appxbundle will be extracted* >

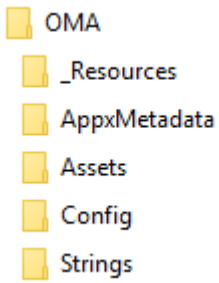
   For example, MakeAppx unbundle /o /p F:\OMA\OMA.10_1.9.2.0_x86_x64_arm_Release_UWP10.appxbundle /d F:\OMA\unbundle

2. **MakeAppx** unpack /o /l /p *<full path of appx file*> /d *<full folder path where appx file will be extracted*>

   For example, MakeAppx unpack /o /l /p F:\OMA\unbundle\OMA.10_1.9.2.0_x64_Release_UWP10.appx /d F:\OMA\unpack

Customize Package

Once you extract the package under the OMA folder, the folder structure will look similar to the sample as shown in the below figure.



Note:

- Customize every package present in appxbundle (or appx files present in output location of MakeAppx unbundle command.). the steps given here help you customize one package.
- When you update an image file, make sure that the image file name and dimension remain the same.
- When you update the Help file or EULA file, make sure that the file name remains the same.
- When you update any string resource, make sure that the string key name remains the same and only the value is updated.

1. Splash Screen

   This screen is visible for a small amount of time while launching the app.

   - The screen image files start with *SplashScreen...* under OMA\Assets. Create customized images using the image sizes provided. Sizes are mentioned as a part of the image name.

     for example, SplashScreen.scale-200.png

2. App Icon

   This is the image shown in the Home screen or Spring board.

   - The following app icon images are under OMA\Assets. Create customized images using the image sizes provided. Sizes are mentioned as a part of the image name. Replace the image with the same name.
     - LockScreenLogo.scale-200.png
     - Square44x44Logo.scale-200.png
     - Square44x44Logo.targetsize-24_altform-unplated.png
     - Square71x71Logo.scale-200.png
     - Square150x150Logo.scale-200.png
     - Square310x310Logo.scale-200.png
     - StoreLogo.png
     - Wide310x150Logo.scale-200.png

3. OMA Logo

   These images are shown as the default icons of the account.

   - There are two images that you can update under OMA\Assets.

     Image 1. MFA_Icon.png.

     Image 2. OMA_Logo_70.png

   - Replace the customized files with the same name and png format.

     . Note that changing the Image size aspect ratio may at times cause the user interface to look distorted. Verify that the image size/aspect ratio looks as required, before it is finalized.

4. Other Company Icons.

   Each account has an image associated with it. The account image can be selected among the available images while adding the account during the manual entry of shared secret or, the existing account images can be changed in the edit screen.

   - Others.png

5. App Display Name.

   This is the name of the App which is shown in the home screen.

   - Open AppxManifest.xml under OMA folder and update following information.
     - Package → Identity → Name
     - Package → Identity → Publisher
     - Package → Properties → DisplayName
     - Package → Properties → PublisherDisplayName
     - Package → Applications → Application Id="App" → uap:visualElements → DispalyName
     - Package → Applications → Application Id="App" → uap:visualElements → Description
     - Open Config.json file under OMA\Config folder and update following string resource
       * appConfig.json → appName
       * appConfig.json → companyName

6. EULA.

   Update the following file under OMA\Assets.

   - eula.html

7. Private Policy.

   Open Config.json file under OMA\Config folder and update the following string resource.

   - appConfig.json → privacyPolicyUrl

8. Help.

   Update the following file under OMA\Strings\en.

   - help.html

9. 3rd Party Company List with Images.

   These are the 3rd party companies which could be used to configure the account through OMA. This page can be viewed while creating a new account through a manual entry of Key or Editing the account information.

   - Open *companylist.json* file under OMA\Assets folder and update as many entries as needed, The company name and image is shown in the same order as present in file. Any Item can be deleted. Available image name can be updated in *IconName*. The image mentioned in IconName key must be present in OMA\Assets folder.

10. If EULA needs to be displayed on first launch.

    - Open *Config.json* under OMA\Config folder and update following information with value as true or false. True is for show at first launch and false is for not showing.
      - appConfig.json → showEULA

11. App Version Number.

    Open AppxManifest.xml under OMA folder and update the following information.

    - Package → Identity → Version

Create/Pack Package

Open Developer Command Prompt for VS2015 and run the following commands:

> **Note:**
>
> In the following commands, appx and appxbundle file name and full path must be same.

1. **MakeAppx** pack /o /l /d <*full path of customized folder*> /p <*full path of customized appx file*>

   For example, **MakeAppx** pack /o /l /d F:\OMA\unpack /p F:\OMA\unbundle\OMA.10_1.9.2.0_x64_Release_UWP10.appx

   > **Note:**
   >
   > The step mentioned above helps you to customize one package. Use the step to customize every package and then run the following commands to generate the appxbundle and signing it.

2. **MakeAppx** bundle /o /d <*full path of customized unbundle folder*> /p <*full path of customized appxbundle file*>

   For example, **MakeAppx** bundle /o /d F:\OMA\unbundle /p F:\OMA\OMA.10_1.9.2.0_x86_x64_arm_Release_UWP10.appxbundle

3. **SignTool** sign /fd SHA256 /a /f <*full path of pfx certificate file*> /p <*password of pfx file*> <*full path of customized appxbundle file*>

   For example, **SignTool** sign /fd SHA256 /a /f F:\OMA\OMA.10_TemporaryKey.pfx F:\OMA\OMA.10_1.9.2.0_x86_x64_arm_Release_UWP10.appxbundle

> **Note:**
>
> /p <*password*> parameter in the above command is optional, if pfx file is not protected by password then ignore this parameter. **https://msdn.microsoft.com/en-us/library/windows/desktop/jj835832(v=vs.85).aspx** describes how to create a signed certificate used in **SignTool** command.

For more information about MakeAppx utility, See **https://msdn.microsoft.com/en-us/library/windows/desktop/hh446767(v=vs.85).aspx.**

Part III

Developing with Identity Federation

Oracle Identity Federation enables business partners to achieve a federated environment by providing the mechanism with which companies can form a federation and securely share services and data across their respective security domains.

This part discusses developing applications using the Oracle Access Management Identity Federation APIs.

It contains the following chapters:

8

Developing a Custom User Provisioning Plug-in

Oracle Access Management Identity Federation (Identity Federation) leverages the Access Manager plug-in framework to facilitate the provisioning of users. A standard user provisioning plug-in is provided or you can develop a custom plug-in.
This chapter provides the following sections:

- Introduction to User Provisioning Plug-ins
- Introduction to Plug-in Interfaces
- Sample Code: Custom User Provisioning Plug-in
- Developing a User Provisioning Plug-in

See Also,

Using the Default Identity Provisioning Plug-in in *Fusion Middleware Administering Oracle Access Management*.

8.1 Introduction to User Provisioning Plug-ins

When Identity Federation is acting in Service Provider (SP) mode, the user assertion is mapped to a local store to complete the federated single sign-on. However, in some cases when a Service Provider is performing user assertion, a user may not be found. The default user provisioning plug-in (`LDAPProvisioningPlugin`) will provision the user in the LDAP store configured as the Access Manager identity store.

All the information collected at runtime is passed to any user provisioning plug-in, standard or custom. The custom user provisioning plug-in must decide, based on this information, what user information it needs to retrieve and use. Additionally, each custom plug-in can include its own configuration designed to perform extra processing of the user to be provisioned.

When Identity Federation is acting in SP mode and fails to map assertion to a user, it will look for a configuration property to check if the missing user should be provisioned. If the user provisioning flag is set to true, Identity Federation will look up the plug-in name that needs invoking. The stand plug-in (`LDAPProvisioningPlugin`) is invoked by default if a custom plug-in is not being used. The `GenricPluginFactory` is used to locate the plug-in defined and executes the provisioning logic.

Identity Federation retrieves the property associated with the partner `nameideattrname` to populate the `nameid` value in the attribute list sent to the plug-in. If Identity Federation is configured to use the standard plug-in, the options for data store selection is as follows:

- If Identity Federation is using the partner specific data store (multi-store), then Identity Federation will pass the identify store name to the plug-in.
- If Identity Federation uses the default user identity store, the standard plug-in will use the User Provisioning APIs to provision user data in the data store.
- If no partner specific store is configured, the default identity store is used.

The User Provisioning API used to provision a user is the same regardless whether a default identity store or a partner specific store is used.

8.2 Introduction to Plug-in Interfaces

Reviewers: Do we want to provide a short summary of each OIF interface here? If so, please provide. See Introduction to Plug-in Interfaces in the OAM plug-in chapter that provides a similar discussion.

The main class a custom user provisioning plug-in extends is `OIFUserProvisioningPlugin`. The following interfaces are exposed to custom plug-ins:

- `oracle.security.fed.plugins.fed.provisioning.OIFUserProvisioningPlugin.java` (extends `oracle.security.am.plugin.AbstractAMPlugin`)
- `oracle.security.fed.plugins.fed.provisioning.UserContext.java`
- `oracle.security.fed.plugins.fed.provisioning.UserProvisioningException.java`
- `oracle.security.fed.plugins.fed.provisioning.UserProvisioningConstants.java`

For more information about these interfaces, see *Oracle Fusion Middleware User Provisioning Plug-in Java API Reference for Oracle Access Management Identity Federation*.

8.3 Sample Code: Custom User Provisioning Plug-in

The custom user provisioning plug-in jar file structure must conform to an Access Manager custom authentication plug-in structure. Namely, it requires the following files: *plugin*.class, *plugin*.xml, and MANIFEST.MF. For more information about this structure, see Sample Code: Custom Database User Authentication Plug-in .

This section provides the following user provisioning plug-in code samples:

- Sample UserProvisioning.java
- Sample UserPlugin.xml
- Sample MANIFEST.MF

Sample UserProvisioning.java

The following sample code explains user provisioning plug-in:

```
package oif.test;

import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.StringTokenizer;

import javax.naming.Context;
import javax.naming.NamingException;
import javax.naming.directory.Attribute;
import javax.naming.directory.Attributes;
import javax.naming.directory.BasicAttribute;
import javax.naming.directory.BasicAttributes;
import javax.naming.directory.DirContext;
import javax.naming.directory.InitialDirContext;

import oracle.security.am.plugin.ExecutionStatus;
import oracle.security.am.plugin.MonitoringData;
```

```
import oracle.security.am.plugin.PluginConfig;
import oracle.security.fed.plugins.fed.provisioning.OIFUserProvisioningPlugin;
import oracle.security.fed.plugins.fed.provisioning.UserContext;
import oracle.security.fed.plugins.fed.provisioning.UserProvisioningConstants;
import oracle.security.fed.plugins.fed.provisioning.UserProvisioningException;


/*
 * Sample OIF User provisioning plugin
 */

public class ProvisioningPlugin extends OIFUserProvisioningPlugin {


    private boolean monitoringStatus = false;
    private Map paramMap ;
private String userRecordAttrList = null;
private String useridAssertionAttr = null;


    /* (non-Javadoc)
     */
    @Override
    public ExecutionStatus process(UserContext context) throws UserProvisioningException
{
       /*
        * Execute method for plugin
        */
        boolean provisioningStatus = false;
        try{
        Map<String, Object> attrs = context.getAttributes();
        Map<String, Object> attrsMapping = context.getAttributesUsedInMapping();
if (useridAssertionAttr == null) {
System.out.println("User id attribute to create user is not found in the attributes
list");
return ExecutionStatus.ABORT;
}

String userid = null;
if (attrs.containsKey(useridAssertionAttr)) {
Object valueObj = attrs.get(useridAssertionAttr);
if (valueObj instanceof String)
userid = (String) valueObj;
else {
userid = (String)((Set) valueObj).iterator().next();
}
}


        DirContext ctx = getContext();

        // creating the user record
        Attributes record = new BasicAttributes();

        // Create the objectclass to add
        Attribute objClasses = new BasicAttribute("objectClass");
        objClasses.add("top");
        objClasses.add("person");
        String objectClass = "inetOrgPerson";
        objClasses.add(objectClass);
        objClasses.add("organizationalPerson");
        record.put(objClasses);
```

```
            String userIDAttr = "uid";

        // Set the attributes
        record.put(new BasicAttribute(userIDAttr, userid));
StringTokenizer st = new StringTokenizer(userRecordAttrList, ",");
while (st.hasMoreTokens()) {
String key = (String) st.nextToken();
       record.put(new BasicAttribute(key, attrs.get(key)));
}
            Set keys = attrsMapping.keySet();
Iterator itr = keys.iterator();
while (itr.hasNext()) {
String key = (String) itr.next();
if (!attrs.containsKey(key)) {
       record.put(new BasicAttribute(key, attrsMapping.get(key)));

}

    String ldapUserBaseDN = "dc=iplanet,dc=com";
        // Create the record
        ctx.createSubcontext("cn=" + userid + "," + ldapUserBaseDN, record);
        provisioningStatus = true;
        }
}
         catch(Exception e){
             /*
              * If exception abort the authentication.
              */
         e.printStackTrace();
             return ExecutionStatus.ABORT;
         }

        if( provisioningStatus){
            /*
             * Success
             */
            return ExecutionStatus.SUCCESS;
        }else{
            /*
             * Failure.
             */
            return ExecutionStatus.FAILURE;
        }
    }

    /* (non-Javadoc)
     * @see oracle.security.am.plugin.GenericPluginService#initialize(java.util.Map)
     */
    @Override
    public ExecutionStatus initialize(PluginConfig config) {
        //success for the execution status
userRecordAttrList =
(String)config.getParameter(UserProvisioningConstants.KEY_USER_RECORD_ATTRIBUTE_LIST);
useridAssertionAttr =
(String)config.getParameter(UserProvisioningConstants.KEY_USERID_ATTRIBUTE_NAME);

        return ExecutionStatus.SUCCESS;
    }

    /* (non-Javadoc)
     * @see oracle.security.am.plugin.GenericPluginService#getDescription()
```

```
 */
@Override
public String getDescription() {
    return "Ldap Provisioning Plugin";
}

/* (non-Javadoc)
 * @see oracle.security.am.plugin.GenericPluginService#getMonitoringData()
 */
@Override
public Map < String, MonitoringData > getMonitoringData() {
    // TODO Auto-generated method stub
    return null;
}

/* (non-Javadoc)
 * @see oracle.security.am.plugin.GenericPluginService#getMonitoringStatus()
 */
@Override
public boolean getMonitoringStatus() {
    return monitoringStatus;
}

/* (non-Javadoc)
 * @see oracle.security.am.plugin.GenericPluginService#getName()
 */
@Override
public String getPluginName() {
    return "LDAP_Provisioning_plugin";
}

/* (non-Javadoc)
 * @see oracle.security.am.plugin.GenericPluginService#getVersion()
 */
@Override
public int getRevision() {
    return 10;
}


/* (non-Javadoc)
 * @see oracle.security.am.plugin.GenericPluginService#setMonitoringStatus(boolean)
 */
@Override
public void setMonitoringStatus(boolean status) {
    monitoringStatus = status;
}

private DirContext getContext() {
try {
DirContext context = null;

  String ldapURL = "ldap://myldap.example.com:389";
    String ldapUserBaseDN = "dc=iplanet,dc=com";

Hashtable<String, String> env = new Hashtable <String, String> ();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, ldapURL);
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.REFERRAL, "follow");

String credential = "password";
```

```
    String secPrincipal = "cn=Directory Manager";
    env.put(Context.SECURITY_PRINCIPAL, secPrincipal);
    env.put(Context.SECURITY_CREDENTIALS, credential);

    context = new InitialDirContext (env);
    return context;
    } catch (NamingException ne) {
    throw new UserProvisioningException(ne);
    } catch (Throwable e) {
    throw new UserProvisioningException(e);
    }

    }
}
```

Sample UserPlugin.xml

The following sample code provides UserPlugin.xml

```
<Plugin type="User Provisioning">
<author>uid=User1</author>
<email>User1@example</email>
<creationDate>09:32:20,2012-06-15</creationDate>
<description>User provisioning</description>
<configuration>
 <AttributeValuePair>
  <Attribute type="string" length="100">KEY_USERID_ATTRIBUTE_NAME</Attribute>
  <mandatory>false</mandatory>
  <instanceOverride>false</instanceOverride>
  <globalUIOverride>true</globalUIOverride>
  <value>uid</value>
 </AttributeValuePair>
 <AttributeValuePair>
  <Attribute type="string" length="200">KEY_USER_RECORD_ATTRIBUTE_LIST</Attribute>
  <mandatory>true</mandatory>
  <instanceOverride>false</instanceOverride>
  <globalUIOverride>true</globalUIOverride>
  <value>mail,uid</value>
 </AttributeValuePair>
</configuration>
</Plugin>
```

Sample MANIFEST.MF

The following sample code provides MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: ProvisioningPlugin
Bundle-SymbolicName: ProvisioningPlugin
Bundle-Version: 10
Bundle-Activator: oif.test.ProvisioningPlugin
Import-Package:
org.osgi.framework;version="1.3.0",oracle.security.fed.plugins.fed.provisioning
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
```

8.4 Developing a User Provisioning Plug-in

This section provides steps to write a custom Identity Federation user provisioning plug-in. The following describes the actions a developer must take after the system architect identifies the

business requirements for the custom plug-in and considers the user provisioning flow when a user is not mapped to a local user store.

This section contains the following topics:

- Developing a Custom Plug-in: Process Overview
- Files Required for Compiling a Plug-in

8.4.1 Developing a Custom Plug-in: Process Overview

As Identity Federation leverages the Access Manager plug-in framework, the process is similar for both. For more information, see About Planning, the Authentication Model, and Plug-ins.

1. Extend `OIFUserProvisioningPlugin` class and implement the following methods. For more information, see About Writing a Custom Authentication Plug-in .
   - Implement `initialize` method
   - Implement `process` method
2. Develop plug-in code using appropriate Access Manager interfaces and packages. For more information, see:
   - Introduction to Plug-in Interfaces
   - Sample Code: Custom Database User Authentication Plug-in
3. Prepare metadata for the custom plug-in. For more information, see Sample Plug-in Configuration Metadata Requirements.
4. Prepare the plug-in jar file and manifest and deliver to your deployment team. For more information, see:
   - Sample Manifest File for the Plug-in
   - Understanding the Plug-in JAR File Structure
5. Proceed to Files Required for Compiling a Plug-in.

For information about deploying and managing custom authentication plug-ins, see Deploying and Managing Individual Plug-ins for Authentication.

8.4.2 Files Required for Compiling a Plug-in

The following jar files are needed for compiling the custom user provisioning plug-in:

- felix.jar
- oam-plugin.jar
- fed.jar

The file are located in *DOMAIN_HOME*/servers/*managed_instance_name*/tmp/*_WL_user*/oam_server_11.1.2.0.0/*RANDOM_STRING*/APP-INF/lib.

9

Developing a Message Processing Plug-in

You can develop a plug-in that allows Oracle Access Management Identity Federation to process SAML messages that contain custom elements and attributes often required by third party or custom SAML implementations. In effect, you will be extending the functionality of the `OIFMessageProcessingPlugin`.

This chapter contains the following sections.

9.1 Understanding Custom SAML Elements

Because SAML is an extensible protocol, custom elements and attributes can be inserted into SAML messages where needed. Third party or custom SAML implementations might require these particular custom elements or attributes to function. For example, an Identity Provider (IdP) might require a custom `<CompanyInfo>` element included in the SAML extensions portion of the message to provide the name of the company issuing the SAML request. The Oracle Access Management Identity Federation (Identity Federation) `OIFMessageProcessingPlugin` can be modified to process these custom elements.

> **Note:**
>
> Only one plug-in is allowed in your Oracle Access Management environment but you can use conditional logic in the plug-in to accomplish different things for different messages.

9.2 Extending the OIFMessageProcessingPlugin

Follow this procedure to extend the `OIFMessageProcessingPlugin` code.

1. Create a directory.

   In this tutorial, we will call it `plugindev`.

2. Create the following subdirectories.

   `plugindev/src/msgprocplugin`

3. Create `SampleMsgProcPlugin.java` using the content in SampleMsgProcPlugin.java.

   Oracle Access Management overrides the standard JDK XML classes with Oracle-specific ones so the DOM factory objects are retrieved directly from the System Class Loader using Class.forName.

   SampleMsgProcPlugin.java

```
package msgprocplugin;

import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import oracle.security.am.plugin.*;
import oracle.security.fed.plugins.fed.msgprocessing.*;
import org.w3c.dom.*;
import org.w3c.dom.ls.*;
import org.xml.sax.*;
import static java.lang.System.err;

public class SampleMsgProcPlugin extends OIFMessageProcessingPlugin {

        private boolean monitoringStatus;

        public ExecutionStatus process(MessageContext messageCtx) throws
MessageProcessingException {
                try {
                        String msg = "";
                        msg += "************************************\n";
                        msg += "* SAMPLE MESSAGE PROCESSING PLUGIN *\n";
                        msg += "************************************\n";
                        msg += "Partner Name: " + messageCtx.getPartnerName() + "\n";
                        msg += "Message Type: " + messageCtx.getMessageType() + "\n";
                        msg += "Message Version: " + messageCtx.getMessageVersion()
+ "\n";
                        msg += "User DN: " + messageCtx.getUserDN() + "\n";
                        msg += "User ID: " + messageCtx.getUserID() + "\n";
                        msg += "User ID Store: " + messageCtx.getUserIDStore() +
"\n";

                        // Determine if this message meets our criteria for
modification
                        boolean matches =
                                "LoopbackIDP".equals("" +
messageCtx.getPartnerName()) &&
                                "SSO_AUTHN_REQUEST_OUTGOING".equals("" +
messageCtx.getMessageType()) &&
                                "SAML2.0".equals("" +
messageCtx.getMessageVersion());

                        if (!matches)
                                msg += "@@@@@@ CRITERIA NOT MET - SKIPPING THIS
MESSAGE @@@@@@\n";
                        else {
                                 // your business logic here
                        }

                        msg += "=================ENDS=================\n";
                        err.println(msg);
                        return ExecutionStatus.SUCCESS;
                } catch (Exception e) {
                        e.printStackTrace();
                        throw handle(e);
                }
        }

        @Override
        public String getDescription(){
                return "Sample Message Processing Plugin";
        }
```

```
        @Override
        public Map<String, MonitoringData> getMonitoringData(){
                return null;
        }

        @Override
        public boolean getMonitoringStatus(){
                return monitoringStatus;
        }

        @Override
        public String getPluginName(){
                return "SampleMsgProcPlugin";
        }

        @Override
        public int getRevision() {
                return 123;
        }

        @Override
        public void setMonitoringStatus(boolean status){
                this.monitoringStatus = status;
        }
}
```

4. Place `SampleMsgProcPlugin.java` in the `plugindev/src/msgprocplugin` directory.
5. Create the `SampleMsgProcPlugin.xml` plug-in manifest using the content in SampleMsgProcPlugin.java.

   In this step, you can also define configuration settings for the plug-in that can then be modified using the Oracle Access Management Console.

   SampleMsgProcPlugin.xml

```
<?xml version="1.0"?>
<Plugin type="Message Processing">
  <author>John Doe</author>
  <email>donotreply@example.com</email>
  <creationDate>2015-04-16 12:53:37</creationDate>
  <description>Sample Message Processing Plugin</description>
  <configuration>
  </configuration>
</Plugin>
```

6. Put `SampleMsgProcPlugin.xml` in the `plugindev/` directory
7. Create the MANIFEST.MF file using the content in MANIFEST.MF.

   This represents the OSGi bundle metadata. It lists the Java packages required by the plug-in.

> **Note:**
>
> Note that Import-Package is all on one-line.

MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: SampleMsgProcPlugin
Bundle-SymbolicName: SampleMsgProcPlugin
Bundle-Version: 1
Bundle-Activator: oracle.ateam.msgprocplugin.SampleMsgProcPlugin
Import-Package:
javax.xml.parsers,oracle.security.am.plugin,oracle.security.fed.plugins.fed.msgproces
sing,org.osgi.framework;version="1.3.0",org.w3c.dom,org.w3c.dom.ls,org.xml.sax
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
```

8. Put `MANIFEST.MF` in the `plugindev/` directory
9. Create the `compile.sh` shell script using the content in compile.sh.

   This shell script compiles the plug-in. Another option would be to use ANT or Maven. The path DOMAIN_HOME and the SERVER_NAME will need to be changed for your environment. Also note that JARS= is all on one line.

> **Note:**
>
> Note that JARS= is all on one-line.

compile.sh

```
#!/bin/bash
DOMAIN_HOME=/idmtop/config/domains/IAMAccessDomain
SERVER_NAME=wls_oam1

JARS="$(find $DOMAIN_HOME/servers/$SERVER_NAME/tmp/_WL_user/oam_server_11.1.2.0.0/ -
name fed.jar -o -name oam-plugin.jar -o -name felix.jar | tr '\n' ':' | sed -e
's/:$//')"
SRCS="$(find src -name '*.java')"
rm -rf build
mkdir build
javac -d build -classpath $JARS $SRCS
cp SampleMsgProcPlugin.xml build
mkdir build/META-INF
cp MANIFEST.MF build/META-INF
cd build
jar cvmf META-INF/MANIFEST.MF ../SampleMsgProcPlugin.jar *
```

10. Put `compile.sh` in the `plugindev/` directory
11. Run compile.sh to create SampleMsgProcPlugin.jar.

9.3 Deploying the Message Processing Plug-in

Use this procedure to import and activate the `SampleMsgProcPlugin.jar`.

1. Login to Oracle Access Management Console as administrator.
2. Click Application Security at the top of the Console.
3. Click Authentication Plug-ins under the Plug-ins section.

   The Authentication Plug-ins screen is used to configure all Oracle Access Management plug-ins.
4. Click Import Plug-in.

An Import Plug-in screen is displayed.

5. Click Browse and search for the `SampleMsgProcPlugin.jar` that was built in "Extending the OIFMessageProcessingPlugin".
6. Click Import to upload the JAR.
7. Refresh the table and search for the plug-in you just imported.
8. Click Distribute Selected.
9. Click the Refresh icon to confirm that the status has changed to Distributed.
10. Click Activate Selected.
11. Click the Refresh icon to confirm that the status has changed to Activated.

The plug-in has now been installed and activated.

9.4 Enabling the Message Processing Plug-in

Use this procedure to tell Identity Federation that the `SampleMsgProcPlugin.jar` is ready for use.

1. Open the `$DOMAIN_HOME/config/fmwconfig/oam-config.xml` file in a text editor.
2. Find the Setting with name `messageprocessingeplugin` tag defined under the Setting with name `fedserverconfig` tag.
3. Change the value of `messageprocessingeplugin` to the name of the plug-in.
4. Find the Setting with name `messageprocessingenabled` tag defined under the Setting with name `fedserverconfig` tag.
5. Change the value of `messageprocessingenabled` from false to true.
6. Find the Setting with name `Version` (near the top of the file) and increment the version number.

   This should be done every time the `oam-config.xml` file is modified.
7. Save the file.

   When the version number in the `oam-config.ref` file in the same directory has increased to the new version number, the modifications have been loaded.

10

Using the REST API for Identity Federation

The Identity Federation Wiring REST API is designed to support establishment and management of federation agreements. It facilitates SAML metadata exchange between the Identity Provider partner and a Service Provider partner and enables or disables federation SSO between those two partners. This chapter describes the Oracle Access Management Identity Federation API.

Notes About Using cURL

This chapter uses cURL to demonstrate the REST calls that the identity federation client sends to the identity federation server. cURL is free software that you can download from the cURL website at `http://curl.haxx.se/`

Using cURL to send REST calls to the server can help you better understand how the client interacts with the server. It can also be a helpful troubleshooting tool. Consider the following when using this chapter.

- cURL commands that contain single quotes (`'`) will fail on Windows. When possible, use double quotes (`"`) in place of single quotes.
- If a command requires both single quotes and double quotes, escape the double quotes with a backslash (for example: `\"`) and replace the single quotes with double quotes.

> **Note:**
>
> In this guide, line breaks in cURL commands and server responses are for display purposes only.

Available Java API References

In addition to this *Oracle Fusion Middleware Developer's Guide for Oracle Access Management*, the *Oracle Fusion Middleware Java API Reference for Oracle Access Management OAuth Services* is available.

This section provides the following topics:

10.1 Resource URLs

Resource URLs are structured to include the Access Manager product version, the component exposed by the REST service, and the resources being invoked. The basic structure of a resource URL is as follows:

`http(s)://host:port/oam/services/rest/path`

where:

- **host** is the host where the OAM Server is running.
- **port** is the HTTP or HTTPS port.
- **path** is the relative path that identifies a particular resource. *path* is constructed as /*version*/*component*/*service*/ where:
  - *version* - is the Access Manager product version, such as 11.1.2.0.0
  - *component* - is the component exposed by the RESTful service, such as ssa or fed
  - *service* - is the root resource for that given API, such as hostidentifier

  An example of a *path* value is: `/oam/services/rest/11.1.2.0.0/fed/admin/sso/hostidentifier/host_identifier_name`.

The Access Manager identity federation REST Web Application Description Language (WADL) file lists the supported identity federation resources and methods. The Identity Federation REST WADL document is available at `http://HOST:PORT/oam/services/rest/11.1.2.0.0/fed/admin/application.wadl`.

10.2 URL Resources and Supported HTTP Methods

Access Manager identity federation are mapped to URL resources. Each resource is referenced by a global identifier (URI).

Access to URL resources is based on user role. The RESTful service expects user credentials to be present in the Authentication header of the HTTP request in BASIC mode. If the authenticated user has the policy administration role, the requested policy administration action is performed.

10.3 Resources Summary

Table 5-5 provides detail about each policy resource, the supported HTTP methods, and the results of each action.

Table 10-1 ***Access Manager Identity Federation Resources Summary***

Resource	Method	Description
`/oam/services/rest/11.1.2.0.0/fed/admin/sso`	POST	Enable Federation SSO service on the server and configure the logout done URL.
	PUT	Same as POST operation.
	GET	Get the enable status of the Federation SSO service and the logout done URL
`/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners`	POST	A service provider (SP) or identity provider (IdP) partner resource is created. The request is performed on the resource that is the parent of the
`/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/idp`	GET	A list of IdP partners is retrieved by this method.

Table 10-1 (Cont.) *Access Manager Identity Federation Resources Summary*

Resource	Method	Description
/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp	GET	A list of SP partners is retrieved by this method. The resource that represents the Resource Types object is returned. This representation contains the matching Resource Type resource attributes and their values.
/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/idp/partnerName	POST	A specific IdP partner resource is created by this method, where partnerName is the name of the partner to be created.
	PUT	Same as POST operation.
	GET	A specified IdP partner resource is retrieved by this method, where partnerName is the name of the IDP partner requested.
	DELETE	A specified IdP partner resource is deleted by this method.
/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/partnerName	POST	A specific SP partner resource is created by this method, where partnerName is the name of the partner to be created.
	PUT	Same as POST operation.
	GET	A specified SP partner resource is retrieved by this method, where partnerName is the name of the SP partner requested.
	DELETE	A specified SP partner resource is deleted by this method. The SP partner resource matching the ID or NAME query parameters is deleted
/oam/services/rest/11.1.2.0.0/fed/admin/orchestrator	POST	A client uses this service to connect two federation servers to remote REST services by this method. In this case both the federation servers are OAM installation
/oam/services/rest/11.1.2.0.0/fed/admin/testsp	POST	A test SP resource is enabled or disabled by this method.
	PUT	Same as POST operation.
	GET	A test SP resource is retrieved by this method.
/oam/services/rest/11.1.2.0.0/fed/admin/ssoservice	POST	A local server for local authentication or federation SSO is created using this method. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.
/oam/services/rest/11.1.2.0.0/fed/admin/trustedsppartners	POST	A specific SP partner resource is created using this method. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.
/oam/services/rest/11.1.2.0.0/fed/admin/trustedidppartners	POST	A specific IdP partner resource is created by this method. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.

Table 10-1 (Cont.) ***Access Manager Identity Federation Resources Summary***

Resource	Method	Description
`/oam/services/rest/11.1.2.0.0/fed/admin/orchestratorservice`	POST	A client uses this service to connect two federation servers to remote REST services. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.
`/oam/services/rest/11.1.2.0.0/fed/admin/testspservice`	POST	A test SP resource is enabled or disabled by this method. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.
`oam/services/rest/11.1.2.0.0/ssa/policyadmin/resource`	POST	A Resource object is created by this method. The request is performed on the resource that is a parent of the object. A Resource object matching the request is created in the corresponding Application Domain.
`/fedrest/configuresso`	POST	Re-directs the respective fedrest url to `/oam/services/rest/11.1.2.0.0/fed/admin/ssoservice`. This is used to create a local server for local authentication or federation SSO. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services.The input data type is FORM POST data.
`/fedrest/createsp`	POST	Re-directs the respective fedrest url to `/oam/services/rest/11.1.2.0.0/fed/admin/trustedsppartners`. This is used to create a specific SP partner resource. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.
`/fedrest/createidp`	POST	Re-directs the respective fedrest url to `/oam/services/rest/11.1.2.0.0/fed/admin/trustedidppartners`. This is used to create a specific IdP partner resource. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.
`/fedrest/orchestrator`	POST	Re-directs the respective fedrest url to `/oam/services/rest/11.1.2.0.0/fed/admin/orchestratorservice`. This service is used by a client to connect two federation servers to remote REST services. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.

10.4 cURL Command Examples for Identity Federation

The following examples are provided as reference.

- **Configuring SSO Service using POST cURL Command**
- **Retrieving SSO Service using GET cURL Command**

10.4.1 Configuring SSO Service using POST cURL Command

The REST endpoint */oam/services/rest/11.1.2.0.0/fed/admin/sso* request is used to configure the SSO service when the customer is the identity provider using the POST method. This API is used for wiring with Fusion Applications and it configures the FAAuthScheme.

For Fusion Applications, IdP is configured at global level to:

- Enable SAML 2.0 only.
- Enable SSO POST, SSO Artifact, SLO Redirect profiles only.
- NameID
  - Email Address with mail as the attribute of the user.
  - Unspecified with uid as the attribute of the user (default).
- One set of keys/certificates for SAML operations

OAM/Fed will be able to have specific SP Partner configuration:

- SSO binding to be used.
- NameID format and value to be used.
- NameID format and value to be used.
- Extra attributes to be sent.
  - NameID value sent as an attribute: SP Partner will indicate the SAML Attribute name, and whether to send user's ID or email address.
  - Static attribute value used by the SP during Assertion mapping operations: SP Partner will indicate the SAML Attribute name and its value.

When IdP needs to authenticate the user, it will redirect the user to an URL protected by WebGate OAM with the FAAuthScheme:

- If OAM is configured for local authentication, FAAuthScheme will instruct OAM to display a login page for the user to enter its credentials.

- If OAM is configured for Federation SSO, FAAuthScheme will instruct OAM to start a Federation SSO flow by redirecting the user to SaaS OIF/SP.
- If OAM is configured to let the user decide how to authenticate, FAAuthScheme will instruct OAM to display a chooser page and to then perform a local authentication or Federation SSO operation, depending on the user's choice.

The following is a sample file for this cURL command.

Where

- ssoFederation
  - is the setting in FAAuthScheme that enables federated SSO for the protected resource.
- ssoChooser
  - is the setting that enables the login page to show both federated SSO link and local login with username and password.
- oamLogoutDoneURL
  - is the URL to redirect after user has been logged out through single logout.

```
curl -X
POST -H "Content-Type: application/json" -d '{"ssoFederation": "true",
"ssoChooser": "false", "oamLogoutDoneURL": "http://test.com/customLogout"}'
http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/sso --user
USER:PASSWORD
```

Sample result:

```
{
"status":"1",
"statusMessage":""}
```

10.4.2 Retrieving SSO Service using GET cURL Command

The REST endpoint */oam/services/rest/11.1.2.0.0/fed/admin/sso* request is used to retrieve the SSO service information when the customer is the identity provider using the GET method.

The following is a sample file for this cURL command.

```
curl -u USER:PASSWORD --request GET
'http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/sso'
```

Sample result:

```
{
"ssoFederation":"true",
"ssoChooser":"false",
"oamLogoutDoneURL":"http://test.com/customLogout"
}
```

10.4.3 Configuring SSO Service using PUT cURL Command

The REST endpoint */oam/services/rest/11.1.2.0.0/fed/admin/sso* request is used to configure the SSO service when the customer is the identity provider using the PUT method.

The following is a sample file for this cURL command.

```
curl -X PUT -H "Content-Type: application/json" -d '{"ssoFederation": "false",
"ssoChooser": "false", "oamLogoutDoneURL": ""}'
```

```
http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/sso --user
USER:PASSWORD
```

Sample result:

```
{
"status":"1",
"statusMessage":""
}
```

10.4.4 Creating an SP Partner cURL Command

The REST endpoint */oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/ sp/partnerName* request creates a Trusted Partners Service. The service */trustedpartners/sp/acmeSP* is used, containing the name of the SP Partner.

The following is a sample file for this cURL command.

Where

- metadataB64
  - is the hexadecimal string corresponding to base 64 encoding of the peer partner's metadata XML. When using curl, you will have to escape the + signs in the base 64 encoded metadata string.
- ssoProfile
  - the SAML 2.0 SSO profile to use (artifact or httppost)
- nameIDFormat
  - the NameID format used during Federation SSO. Possible values are emailaddress or unspecified. If emailaddress, then the NameID value of an Assertion created by the IdP will contain the user's email address; if unspecified, then the NameID value of an Assertion created by the IdP will contain the user's ID.

```
curl -X POST
-H "Content-Type: application/json" -d
'{ "metadataB64": "...", "partnerType": "sp", "partnerName": "acmeSP",
"nameIDFormat": "unspecified", "ssoProfile": "httppost" }'
http://SERVER:PORT/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/a
cmeSP --user USER:PASSWORD
```

Sample result:

```
{
"status":"1",
"statusMessage":""
}
```

10.4.5 Listing all SP Partners cURL Command

The REST endpoint */oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp* request retrieves a list of Trusted Partners Services.

The following is a sample file for this cURL command.

```
curl -u USER:PASSWORD --request GET
'http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp'
```

Sample result:

```
{
"partnerInfoList":
[
{
"metadataB64":"...",
"partnerName":"acmeSP",
"nameIDFormat":"unspecified",
"ssoProfile":"httppost",
"providerID":"http://acme:7499/fed/sp",
"assertionConsumerURL":"http://acme:7777/fed/sp/sso",
"logoutRequestURL":"http://acme:7777/fed/idp/samlv20",
"logoutResponseURL":"http://acme:7777/fed/idp/samlv20",
"adminManualCreation":"false",
"displaySigningCertDN":"CN=acme OIF Signing Certificate",
"displaySigningCertIssuerDN":"CN=OIFCert",
"displaySigningCertStart":"2014-10-07T06:32:16-07:00",
"displaySigningCertExpiration":"2024-10-11T06:32:17-07:00",
"displayEncryptionCertDN":" CN=acme OIF Enc Certificate",
"displayEncryptionCertIssuerDN":"CN=OIFCert",
"displayEncryptionCertStart":"2014-10-07T06:32:16-07:00",
"displayEncryptionCertExpiration":"2024-10-11T06:32:17-07:00"
},
{
"metadataB64":"...",
"partnerName":"ciscoSP",
"nameIDFormat":"emailaddress",
"ssoProfile":"httppost",
"providerID":"http://cisco:7499/fed/sp",
"assertionConsumerURL":"http://cisco:7777/fed/sp/sso",
"logoutRequestURL":"http://cisco:7777/fed/idp/samlv20",
"logoutResponseURL":"http://cisco:7777/fed/idp/samlv20",
"lastNameAttrName":"lastname",
"firstNameAttrName":"firstname",
"userNameAttrName":"username",
"emailAttrName":"email"
"adminManualCreation":"false",
"displaySigningCertDN":"CN=cisco OIF Signing Certificate",
"displaySigningCertIssuerDN":"CN=OIFCert",
"displaySigningCertStart":"2014-10-07T06:32:16-07:00",
"displaySigningCertExpiration":"2024-10-11T06:32:17-07:00",
"displayEncryptionCertDN":" CN=cisco OIF Enc Certificate",
"displayEncryptionCertIssuerDN":"CN=OIFCert",
"displayEncryptionCertStart":"2014-10-07T06:32:16-07:00",
"displayEncryptionCertExpiration":"2024-10-11T06:32:17-07:00"
}
]
}
```


10.4.6 Retrieving SP Partner Data cURL Command

The REST endpoint */oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/partnerName* request retrieves information for a specific Trusted Partners Service.

The following is a sample file for this cURL command.

```
curl -u USER:PASSWORD --request GET
'http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/acmeSP'
```

Sample result:


```
{
"metadataB64":"...",
```

```
"partnerName":"acmeSP",
"nameIDFormat":"unspecified",
"ssoProfile":"httppost",
"providerID":"http://acme:7499/fed/sp",
"assertionConsumerURL":"http://acme:7777/fed/sp/sso",
"logoutRequestURL":"http://acme:7777/fed/idp/samlv20",
"logoutResponseURL":"http://acme:7777/fed/idp/samlv20",
"adminManualCreation":"false",
"displaySigningCertDN":"CN=acme OIF Signing Certificate",
"displaySigningCertIssuerDN":"CN=OIFCert",
"displaySigningCertStart":"2014-10-07T06:32:16-07:00",
"displaySigningCertExpiration":"2024-10-11T06:32:17-07:00",
"displayEncryptionCertDN":" CN=acme OIF Enc Certificate",
"displayEncryptionCertIssuerDN":"CN=OIFCert",
"displayEncryptionCertStart":"2014-10-07T06:32:16-07:00",
"displayEncryptionCertExpiration":"2024-10-11T06:32:17-07:00"
}
```

10.4.7 Updating SP Partner Details cURL Command

The REST endpoint */oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/partnerName* request is used to modify information for a specific Trusted Partners Service.

The following is a sample file for this cURL command.

```
curl -X PUT
-H "Content-Type: application/json" -d
'{ "metadataB64": "..." }'
http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/acm
eSP --user USER:PASSWORD
```

Sample result:

```
{
"status":"1",
"statusMessage":""
}
```

10.4.8 Deleting SP Partner Details cURL Command

The REST endpoint */oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/partnerName* request is used to delete information for a specific Trusted Partners Service.

The following is a sample file for this cURL command.

```
curl -u
USER:PASSWORD --request DELETE
'http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/acmeSP'
```

Sample result:

```
{
"status":"1",
"statusMessage":""
}
```

10.4.9 Enabling Test SP using POST cURL Command

The REST endpoint*/oam/services/rest/11.1.2.0.0/fed/admin/testsp* service request is used to enable a test SP using the POST method.

The following is a sample file for this cURL command.

```
curl -X POST
-H "Content-Type: application/json" -d '{"enabled": "true"}'
http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/testsp
--user USER:PASSWORD
```

Sample result:

```
{
"status":"1",
"statusMessage":""
}
```

10.4.10 Retrieving Test SP Enablement using GET cURL Command

The REST endpoint */oam/services/rest/11.1.2.0.0/fed/admin/testsp* service request is used to retrieve test SP enablement details using the GET method.

The following is a sample file for this cURL command.

```
curl -u USER:PASSWORD --request GET
'http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/testsp'
```

Sample result:

```
{
"enabled":"true"
}
```

10.4.11 Disabling Test SP using PUT cURL Command

The REST endpoint*/oam/services/rest/11.1.2.0.0/fed/admin/testsp* service request is used to disable a test SP using the PUT method.

The following is a sample file for this cURL command.

```
curl -X PUT
-H "Content-Type: application/json" -d '{"enabled": "false"}'
http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/testsp
--user USER:PASSWORD
```

Sample result:

```
{
"status":"1",
"statusMessage":""
}
```

10.4.12 Configuring SSO Service using POST cURL Command using /fedrest/configuresso

The */fedrest/configuresso* request redirects the request url to the actual required url */oam/services/rest/11.1.2.0.0/fed/admin/ssoservice*, which is used to configure the SSO service when the customer is the identity provider using the POST method.

The following is a sample file for this cURL command.

```
curl -v -i -u USER:PASSWORD
-X POST -d @ssoConfigureData.in http://SERVER:PORT/fedrest/configuresso
```

File ssoConfigureData.in:

```
spTenantName= &idpProviderID=
&preverify=false
&ssoFederation=true
&ssoChooser=true
&oamadminuser=USER
&oamadminpassword=PASSWORD
&oamadminhost=SERVER
&oamadminport=PORT

curl -u USER:PASSWORD --data "spTenantName=""&idpProviderID=""
&preverify="false"&ssoFederation="true"&ssoChooser="true"
&oamadminuser="USER"&oamadminpassword="PASSWORD"
&oamadminhost="SERVER"
&oamadminport="PORT"
&oamLogoutDoneURL="""" --request
POST 'http://SERVER:PORT/oam/services/rest/11.1.2.0.0/fed/admin/ssoservice';
-X POST -d @ssoConfigureData.in http://SERVER:PORT/fedrest/configuresso
```

10.4.13 Creating an SP Partner cURL Command using /fedrest/createsp

The */fedrest/createsp* request redirects the request url to the actual required url*/oam/services/rest/11.1.2.0.0/fed/admin/trustedsppartners*, which creates a Trusted Partners Service.

The following is a sample file for this cURL command.

Where

- ssoProfile
  - the SAML 2.0 SSO profile to use (artifact or httppost).
- nameIDFormat
  - the NameID format used during Federation SSO. Possible values are emailaddress or unspecified. If emailaddress, then the NameID value of an Assertion created by the IdP will contain the user's email address; if unspecified, then the NameID value of an Assertion created by the IdP will contain the user's ID.

```
curl -v -i -u USER:PASSWORD
-X POST -d @spCurlData.in http://HOST:PORT/fedrest/createsp
```

File spCurlData.in:

```
idpTenantName=&idpTenantURL=
&spPartnerName=spPartner-sample
&spProviderID=&metadata=
&metadataURL=&assertionConsumerURL=&logoutRequestURL=
&logoutResponseURL=&signingCert=&encryptionCert=
&nameIDFormat=unspecified&ssoProfile=artifact&generateNewKeys=
&validityNewKeys=&preverify=false&lastNameAttrName=&firstNameAttrName=
&userNameAttrName=&emailAttrName=&staticAttrName=&staticAttrValue=&customAttrs=

curl -v -i -u USER:PASSWORD
-X POST -d @spCurlData.in
https://SERVER:PORT/oam/services/rest/11.1.2.0.0/fed/admin/trustedsppartners
```

File spCurlData.in:

```
idpTenantName=
&idpTenantURL=&spPartnerName=spPartner-sample
&spProviderID=&metadata=&metadataURL=
&assertionConsumerURL=&logoutRequestURL=
&logoutResponseURL=&signingCert=&encryptionCert=
&nameIDFormat=unspecified&ssoProfile=artifact&generateNewKeys=
&validityNewKeys=&preverify=false&lastNameAttrName=&firstNameAttrName=
&userNameAttrName=&emailAttrName=&staticAttrName=&staticAttrValue=&customAttrs=...
```

10.4.14 Creating an IdP Partner cURL Command using /fedrest/createidp

The */fedrest/createidp* request redirects the request url to the actual required url */oam/services/rest/11.1.2.0.0/fed/admin/trustedidppartners*, which creates a Trusted IdP Partner Service.

The following are sample files for this cURL command.

Where

- ssoProfile
  - the SAML 2.0 SSO profile to use (artifact or httppost).
- nameIDFormat
  - the NameID format used during Federation SSO. Possible values are emailaddress or unspecified. If emailaddress, then the NameID value of an Assertion created by the IdP will contain the user's email address; if unspecified, then the NameID value of an Assertion created by the IdP will contain the user's ID.

```
curl -v -i -u USER:PASSWORD
-X POST -d @idpCurlData.in http://SERVER:PORT/fedrest/createidp
```

File idpCurlData.in:

```
spTenantName=&spTenantURL=
&idpPartnerName=idpPartner-sample
&idpProviderID=&metadata=
&metadataURL=&ssoURL=&ssoSOAPURL=
&logoutRequestURL=&logoutResponseURL=&signingCert=
&encryptionCert=&succinctID=&nameIDFormat=emailaddress
&attributeLDAP=&attributeSAML=&ssoProfile=artifact
&faWelcomePage=&tenantKeyName=&tenantKeyValue=&generateNewKeys=
&validityNewKeys=&preverify=false
```

```
curl -v -i -u USER:PASSWORD
-X POST -d @idpCurlData.in
https://SERVER:PORT/oam/services/rest/11.1.2.0.0/fed/admin/trustedidppartners
```

File idpCurlData.in:

```
spTenantName=&spTenantURL=
&idpPartnerName=idpPartner-sample&idpProviderID=
&metadata=&metadataURL=&ssoURL=&ssoSOAPURL=
&logoutRequestURL=&logoutResponseURL=&signingCert=
&encryptionCert=&succinctID=&nameIDFormat=emailaddress
&attributeLDAP=&attributeSAML=&ssoProfile=artifact
&faWelcomePage=&tenantKeyName=&tenantKeyValue=
&generateNewKeys=&validityNewKeys=&preverify=false
```

10.4.15 Connecting Federation Servers to remote REST services using /fedrest/orchestrator

The */fedrest/orchestrator* request redirects the request url to the actual required url*/oam/services/rest/11.1.2.0.0/fed/admin/orchestrator*, which connect two federation servers to remote REST services.

The following are sample files for this cURL command.

Where

- ssoProfile
  - the SAML 2.0 SSO profile to use (artifact or httppost).
- nameIDFormat
  - the NameID format used during Federation SSO. Possible values are emailaddress or unspecified. If emailaddress, then the NameID value of an Assertion created by the IdP will contain the user's email address; if unspecified, then the NameID value of an Assertion created by the IdP will contain the user's ID.

```
curl -v -i -u USER:PASSWORD
-X POST -d @orch.in http://SERVER:PORT/fedrest/orchestrator
```

File orch.in:

```
command=setupSPAndIdPTrust
&spresturl=https://SERVER:PORT/fedrest/createidp
&spadminuser=USER&spadminpassword=PASSWORD
&spmetadataurl=&idpPartnerName=sample-idp&sptype=oif
&idpresturl=http://SERVER:PORT/fedrest/createsp
&idpadminuser=USER&idpadminpassword=PASSWORD
&idpmetadataurl=&spPartnerName=sample-sp
&idptype=oif&nameIDFormat=emailaddress&ssoProfile=httppost
```

> **Note:**
>
> idpmetadataurl and spmetadataurl should be url encoded.

```
curl -v -i -u USER:PASSWORD
-X POST -d @orch.in
https://SERVER:PORT/oam/services/rest/11.1.2.0.0/fed/admin/orchestratorservice
```

File orch.in:

```
command=setupSPAndIdPTrust
&spresturl=http://SERVER:PORT/oam/services/rest/11.1.2.0.0/fed/admin/trustedidppartners
&spadminuser=USER
&spadminpassword=PASSWORD
&spmetadataurl=
&idpPartnerName=sample-idp
&sptype=oif
&idpresturl=http://SERVER:PORT/oam/services/rest/11.1.2.0.0/fed/admin/trustedsp
```

11

Implementing Custom Authentication Actions

Custom authentication actions enable site-specific operations to be executed during a Federation single sign-on flow with Oracle Access Management Identity Federation acting as an Identity Provider. These actions can be used to authenticate the user or check the validity of the user's session if the user is already authenticated.
The following sections explain how to implement custom authentication actions.

11.1 Understanding Custom Authentication Actions

The Oracle Access Management Identity Federation server (Identity Federation) implements custom actions using the pre- and post-processing action plug-ins. The pre- and post-processing plug-ins are implemented as JSP or JavaEE servlets which are invoked during a Federation single sign-on (SSO) flow either before or after invoking Oracle Access Manager. The following sections explain how the actions work and how they interact with Identity Federation.

11.1.1 Using Pre and Post Processing Custom Authentication Actions

Identity Federation acting as an Identity Provider (IdP) always invokes Oracle Access Manager during a Federation SSO operation. This is done either to identify the user or to check the user's session to see if the user is already authenticated. If Identity Federation determines the user must be identified, the user is forwarded to Oracle Access Manager, specifying the root context and the relative path of the OAM endpoint. At this point, Oracle Access Manager will:

- Perform an authentication operation if necessary.
- Check the validity of the user's session
- (Optionally) perform an Authorization verification to ensure that the user can perform a Federation SSO operation with the SP Partner.

If these operations are successful, Oracle Access Manager forwards the user back to Identity Federation with the authentication information (a user identifier and the time at which the identity was established). Identity Federation analyzes the information and creates or updates the user session. Custom actions can be used during this interaction to:

- Manipulate the data exchanged between Identity Federation and the Oracle Access Manager Authentication Engine. For example, you can construct an email address from a user name: `johndoe` becomes `johndoe@mycompany.com`.
- Perform additional steps during authentication. For example, you can contact an external data source or system to obtain more information about the user.

11.1.2 Setting Up a Custom Authentication Action Plug-in

The following overview illustrates how to set up a custom action plug-in.

1. Implement one or more custom action plug-ins as explained.
   - Implement a pre-processing action plug-in to be performed before invoking Oracle Access Manager. (See "Using Pre-Processing Custom Actions.")
   - Implement a post-processing plug-in for any actions or changes to be performed after authentication. This would occur when the user is returned from Oracle Access Manager to Identity Federation. (See "Using Post-Processing Custom Actions.")
2. Deploy the plug-in(s) to the WebLogic Managed Server on which Oracle Access Manager is running.
3. Configure Identity Federation based on the plug-in task you are implementing.
   - Configure Identity Federation to invoke the plug-in (rather than Oracle Access Manager) if the plug-in is to perform pre-processing tasks.
   - Configure Identity Federation to have Oracle Access Manager invoke the plug-in (rather than redirecting to Identity Federation) if the plug-in is to perform post-processing tasks.

11.1.3 Understanding the Custom Action Flow

When Identity Federation needs to authenticate a user, the flow is as follows.

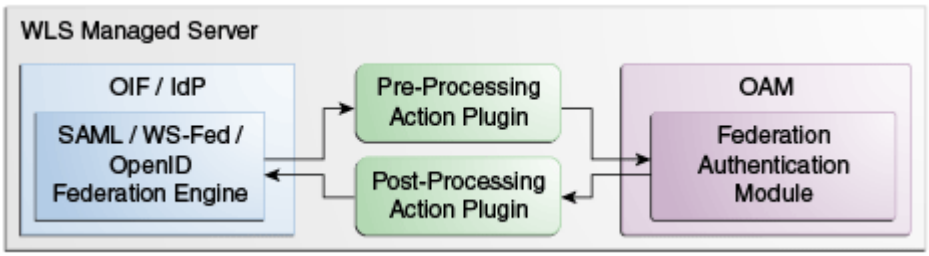
1. Identity Federation, as part of a runtime IdP SSO flow, determines whether the:
   - User needs to be locally authenticated by Oracle Access Manager.
   - User has an existing session and needs to be forwarded to Oracle Access Manager to check the validity of the session.
2. Identity Federation (acting as the IdP) invokes the pre-processing plug-in to perform the applicable custom tasks.
3. The pre-processing plug-in invokes Oracle Access Manager.
4. Oracle Access Manager challenges and authenticates the user, or checks the validity of the user's session.
5. Oracle Access Manager bundles the authentication data and invokes the post-processing plug-in to perform applicable custom tasks.
6. The post-processing plug-in invokes Identity Federation, providing the authentication data.
7. Identity Federation (acting as an IdP) resumes operations.

Figure 11-1 illustrates this flow when Identity Federation is customized and configured to invoke plug-ins:

- Before Identity Federation invokes Oracle Access Manager for authentication or session validation. (See "Using Pre-Processing Custom Actions.")

- Before Oracle Access Manager invokes Identity Federation after authentication or session validation. (See "Using Post-Processing Custom Actions.")

Figure 11-1 Custom Actions Plug-in Flow



11.2 Using Pre-Processing Custom Actions

The pre-processing plug-in is a module to which the user is directed, as part of an authentication operation, before invoking Oracle Access Manager. The plug-in enables custom actions to be taken before authentication. When the plug-in is in use, Identity Federation does not redirect the user to the Authentication Engine; rather, it forwards the user internally to the plug-in, passing to it certain data for use during authentication. After performing its custom actions, the plug-in forwards the user to Oracle Access Manager, along with the data originally provided by Identity Federation, to resume the authentication flow. The following sections contain details.

- Passing Data to the Pre-Processing Plug-in
- Configuring Identity Federation for the Pre-Processing Action

11.2.1 Passing Data to the Pre-Processing Plug-in

The pre-processing custom action interacts with Identity Federation. When Identity Federation redirects a user to Oracle Access Manager, it passes certain data as attributes in the `HttpServletRequest` object. The same data must be made available to pre-processing plug-ins. The data includes:

- The default scheme identifier to be used to challenge the user. This String is identified by `oracle.security.fed.authn.defaultschemeid`.
- A list of scheme identifiers requested by the service provider (SP) to be used to challenge the user. This String list is identified by `oracle.security.fed.authn.schemeidlevels`.
- The comparison rule requested by the SP to determine the scheme with which to challenge the user when a list of scheme identifiers is provided. This is a String identified by `oracle.security.fed.authn.schemeidcomp`.
- The Force Authentication flag indicating whether Oracle Access Manager should challenge the user - even if the user is already authenticated. This is a Boolean identified by `oracle.security.fed.authn.forceauthn`. If missing, false is assumed.
- The Is Passive flag indicating whether Oracle Access Manager is allowed to visually interact with the user. This is a Boolean identified by `oracle.security.fed.authn.passive`. If missing, false is assumed.
- An identifier referencing the action being performed. This String is identified by `oracle.security.fed.authn.refid`.

- The identifier (userID) of the user if set. This String is identified by `oracle.security.fed.authn.userid`.
- The canonical user identifier of the user (userID + Identity Store Name + LDAP DN) if set. This String is identified by `oracle.security.fed.authn.canonicaluserid`.
- The Identity Federation SessionID if set. This String is identified by `oracle.security.fed.sessionid`.
- The identifier referencing the Oracle Access Manager server used to authenticate the user. This String is identified by `oracle.security.fed.authn.engineid`.
- The partner name and the description of the remote SP for which this local authentication is requested, if a federated SSO operation is performed. This String is identified by `oracle.security.fed.authn.providerid` and `oracle.security.fed.authn.providerdescription` respectively.
- The web context where the user should be redirected after authentication by Oracle Access Manager. This String is identified by `oracle.security.fed.return.webcontext`. The root web context is /oam.
- The web relative path where the user should be redirected after authentication by Oracle Access Manager. This String is identified by `oracle.security.fed.return.webpath`. The relative path is /server/fed/authn.

> **Note:**
>
> The pre-processing plug-in can modify all attributes that were set in the `HttpServletRequest` object except the following:
>
> - oracle.security.fed.authn.defaultschemeid
> - oracle.security.fed.authn.schemeidlevels
> - oracle.security.fed.authn.schemeidcomp
> - oracle.security.fed.authn.refid
> - oracle.security.fed.authn.engineid
> - oracle.security.fed.return.webcontext
> - oracle.security.fed.return.webpath

11.2.2 Configuring Identity Federation for the Pre-Processing Action

Configure Identity Federation to forward the user to a pre-processing plug-in by performing these tasks.

1. Enter the WLST environment.

   ```
   $IAM_HOME/common/bin/wlst.sh
   ```

2. Connect to the WLS Admin Server.

   ```
   connect()
   ```

3. Navigate to the Domain Runtime folder.

   ```
   domainRuntime()
   ```

4. Execute the `putStringProperty()` WLST command to set the following properties:

- The preauthnenginewebcontext property references the web context where the custom JSP Page or servlet of the pre-processing plug-in resides. Replace CUSTOM_WEB_CONTEXT with the value specific to your plug-in.

  putStringProperty("/authnengines/preauthnenginewebcontext", "CUSTOM_WEB_CONTEXT ")

- The preauthnenginewebpath property references the path in the web context where the pre-processing plug-in resides. Replace CUSTOM_WEB_PATH with the value specific to your plug-in.

  putStringProperty("/authnengines/preauthnenginewebpath", "CUSTOM_WEB_PATH")

5. Execute the `putBooleanProperty()` WLST command to enable or disable the pre-processing custom plug-in.
   - putBooleanProperty("/authnengines/preauthnengineenabled", "true") to configure Identity Federation to invoke the pre-processing plug-in.
   - putBooleanProperty("/authnengines/preauthnengineenabled", "false") to configure Identity Federation not to invoke the pre-processing plug-in.

11.3 Example: Custom Action Pre-processing

This section illustrates a simple pre-processing plug-in that is invoked by Identity Federation before the user is redirected to Oracle Access Manager. This pre-processing plug-in retrieves the name of the SP partner with which the Federation SSO operation is performed and will save it in a custom cookie that can be used by custom pages; for example, a custom error page. In this example:

- Identity Federation acts as an IdP
- A custom pre-authentication plug-in is used in this example to set a cookie containing the SP partner name. The cookie is called fed-sppartner-cookie.

The pre-processing plug-in consists of a Web application with a root context set to /plugin. It contains one JSP page (named `cookiepartnerset.jsp`) which sets the SP partner name in a cookie. cookiepartnerset.jsp is an example implementation of `cookiepartnerset.jsp`.

cookiepartnerset.jsp

```
<%@page buffer="5" autoFlush="true" session="false"%>
<%@page language="java" import="java.util.*, javax.naming.*,
  javax.naming.directory.*, java.net.*"%>
<%
response.setHeader("Cache-Control", "no-cache");
response.setHeader("Pragma", "no-cache");
response.setHeader("Expires", "Thu, 29 Oct 1969 17:04:19 GMT");

String partnerName = (String)request.getAttribute
  ("oracle.security.fed.authn.providerid");

Cookie cookie = new Cookie("fed-sppartner-cookie", partnerName);
response.addCookie(cookie);

// forward to the OAM server to resume the flow
request.getSession().getServletContext().getContext("/oam").getRequestDispatcher
 ("/server/fed/authn").forward(request, response);
%>
```

> **Note:**
>
> The WAR file of this Web application will need to be deployed to the WLS Managed Server on which Oracle Access Manager is running.

To resume the flow, the plug-in redirects the user to Oracle Access Manager by means of an internal forward. Take the following steps to configure Identity Federation to invoke the pre-processing plug-in.

1. Enter the WLST environment:

   `$IAM_HOME/common/bin/wlst.sh`

2. Connect to the WLS Admin Server:

   `connect()`

3. Navigate to the Domain Runtime folder:

   `domainRuntime()`

4. Execute the `putStringProperty()` WLST command to set the preauthnenginewebcontext property.

   putStringProperty("/authnengines/preauthnenginewebcontext", "/plugin")

5. Execute the `putStringProperty()` WLST command to set the preauthnenginewebpath property.

   putStringProperty("/authnengines/preauthnenginewebpath", "/cookiepartnerset.jsp")

6. Execute the `putBooleanProperty()` WLST command to enable the pre-processing plug-in to be invoked by Identity Federation.

   putBooleanProperty("/authnengines/preauthnengineenabled", "true")

11.4 Using Post-Processing Custom Actions

The user is directed to the post-processing plug-in module, as part of an authentication operation, after the Oracle Access Manager Authentication Engine has completed processing and before the user is directed to Identity Federation. The plug-in enables custom actions to be taken after authentication.

When the post-processing plug-in is in use, Oracle Access Manager forwards the user and authentication data internally to it. After performing its custom actions, the plug-in returns the user to Identity Federation, supplying the authentication data. The plug-in must provide Identity Federation with the data that was passed to it as part of the authentication flow; this consists of attributes that were set in the `HttpServletRequest` object. The following sections have details.

- Passing Data to the Post-Processing Plug-in
- Configuring Identity Federation for the Post-Processing Action

11.4.1 Passing Data to the Post-Processing Plug-in

The post-processing plug-in interacts with Identity Federation. When Oracle Access Manager redirects a user to Identity Federation, it passes certain data to the plug-in as attributes in the `HttpServletRequest` object. The data includes:

- The identifier referencing the requested action that was performed. The String is identified by `oracle.security.fed.authn.refid`.
- The schemeID and level of the authentication performed by Oracle Access Manager. The String is identified by `oracle.security.fed.authn.result.schemeidlevel`.
- The result of the authentication operation. The String is identified by `oracle.security.fed.authn.result.statuscode` as SUCCESS, if the operation was successful.
- The partner name of the remote SP for which this local authentication is requested if a federated SSO operation is performed. The String is identified by `oracle.security.fed.authn.providerid`.
- The Oracle Access Manager module identifier used to authenticate the user. The String is identified by `oracle.security.fed.authn.engineid`.
- The canonical identifier of the user (userID + Identity Store Name + LDAP DN). The String is identified by `oracle.security.fed.authn.userid`.
- The authentication time. The Date is identified by `oracle.security.fed.authn.authntime`.
- The expiration time of the authenticated session. The Date is identified by `oracle.security.fed.authn.expirationtime`.
- The user's Oracle Access Manager SessionID. The String is identified by `oracle.security.fed.authn.oamsessionid`.
- The user's Oracle Access Manager Session type. The String is identified by `oracle.security.fed.authn.oamsessiontype`.
- The Identity Federation SessionID if set. The String is identified by `oracle.security.fed.sessionid`.

> **Note:**
>
> The plug-in can modify all attributes that were set on the `HttpServletRequest` object except:
>
> - oracle.security.fed.authn.result.schemeidlevel
> - oracle.security.fed.authn.engineid
> - oracle.security.fed.authn.oamsessionid
> - oracle.security.fed.authn.oamsessiontype
> - oracle.security.fed.sessionid

The custom post-processing plug-in can add these optional elements.

- A map of attributes to be included in the outgoing SAML Assertion as SAML Attributes. This map has String objects as keys and a Collection (Set/List) of String objects or a String object as values (identified by `oracle.security.fed.authn.fedattributes`).

  The attributes will be included in the outgoing SSO response as is, and will only be sent for this current Federation SSO operation. They will be discarded afterwards.
- A string to be used as the SAML NameID value instead of the configured NameID expression in the SP Partner entry. For SAML 2.0 SP Partners, this string will only be used

if the NameID format for the SP Partner is not Persistent or Transient. The String is identified by `oracle.security.fed.authn.feduserid`.

This string will only be used as the NameID for this current Federation SSO operation and will be discarded afterwards.

After processing, the post-processing plug-in must forward the user to Identity Federation. (Oracle Access Manager would invoke in the absence of the plug-in).

- The root web context is /oamfed
- The relative path is /user/loginsso

11.4.2 Configuring Identity Federation for the Post-Processing Action

Configure Identity Federation to forward the user to a post-processing plug-in by performing the following tasks.

1. Enter the WLST environment:

   `$IAM_HOME/common/bin/wlst.sh`

2. Connect to the WLS Admin Server:

   `connect()`

3. Navigate to the Domain Runtime folder:

   `domainRuntime()`

4. Execute the `putStringProperty()` WLST command to set the two following properties:

   - The `postauthnenginewebcontext` property references the web context where the custom JSP Page or servlet of the post-processing plug-in resides. Replace CUSTOM_WEB_CONTEXT by the value specific to your plug-in.

     putStringProperty("/authnengines/postauthnenginewebcontext", "CUSTOM_WEB_CONTEXT")

   - The `postauthnenginewebpath` property references the path in the web context where the post-processing plug-in resides. Replace CUSTOM_WEB_PATH by the value specific to your plug-in.

     putStringProperty("/authnengines/postauthnenginewebpath", "CUSTOM_WEB_PATH")

5. Execute the `putBooleanProperty()` WLST command to enable or disable the post-processing plug-in.

   - putBooleanProperty("/authnengines/postauthnengineenabled", "true") to configure Identity Federation to invoke the post-processing plug-in.
   - putBooleanProperty("/authnengines/postauthnengineenabled", "false") to configure Identity Federation not to invoke the post-processing plug-in.

11.5 Example: Custom Action Post-Processing

This section illustrates a simple post-processing plug-in that is invoked by Oracle Access Manager before the user is redirected to Identity Federation at the end of a local authentication operation. This plug-in accesses a custom cookie presented by the browser, extracts data from it, and sets the data as attributes. Identity Federation will then include it in the outgoing SAML assertion. In this example:

- Identity Federation acts as an IdP.

- The custom post authentication plug-in sets some attributes as session attributes called attr1 and attr2.
- Identity Federation (as the IdP) will send the attr1 and attr2 attributes to the SP Partner with which the Federation SSO operation is being performed when creating an assertion.
- A custom component sets the cookie used in this example.

In this sample, the plug-in adds the following attributes, extracted from a custom cookie that is previously set by another component, after a successful authentication:

- cookie-language contains the preferred language of the user.
- cookie-homepage contains the preferred home page of the user.

The post-processing plug-in consists of a Web application with a root context set to /plugin. It contains one JSP page (named cookieextract.jsp) which extracts the data from the custom cookie and sets it as session attributes. cookieextract.jsp is an example implementation of cookieextract.jsp.

cookieextract.jsp

```
<%@page buffer="5" autoFlush="true" session="false"%>
<%@page language="java" import="java.util.*, javax.naming.*,
  javax.naming.directory.*, java.net.*"%>
<%
response.setHeader("Cache-Control", "no-cache");
response.setHeader("Pragma", "no-cache");
response.setHeader("Expires", "Thu, 29 Oct 1969 17:04:19 GMT");

// check if authentication was successful
if
("SUCCESS".equals(request.getAttribute("oracle.security.fed.authn.result.statuscode")))
{
        // authentication was successful. Attributes will be added
        Map attributes = new HashMap();

        // get the cookie
        Cookie[] cookies = request.getCookies();
        String cookieValue = null;
        for(int i = 0; i < cookies.length; i++)
        {
                Cookie cookie = cookies[i];
                if (cookie.getName().equals("customcookie"))
                        cookieValue = cookie.getValue();
        }
        if (cookieValue != null && cookieValue.length() > 0)
        {
                StringTokenizer st = new StringTokenizer(cookieValue, "+");
                String language = st.nextToken();
                String homepage = st.nextToken();

                attributes.put("cookie-language", language);
                attributes.put("cookie-homepage", homepage);

                request.setAttribute( "oracle.security.fed.authn.fedattributes",
attributes);
        }
}

// forward to the OIF server to resume the flow
request.getSession().getServletContext().getContext("/oamfed").getRequestDispatcher("/
user/loginsso").forward(request, response);
%>
```

> **Note:**
>
> The WAR file of this Web application will need to be deployed on the WLS Managed Server where OAM is running

The plug-in redirects the user to the Identity Federation server by means of an internal forward to resume the flow. Take the following steps to configure Identity Federation to invoke the post-processing plug-in at the end of local authentication flow.

1. Enter the WLST environment:

   ```
   $IAM_HOME/common/bin/wlst.sh
   ```

2. Connect to the WLS Admin Server:

   ```
   connect()
   ```

3. Navigate to the Domain Runtime folder:

   ```
   domainRuntime()
   ```

4. Execute the `putStringProperty()` WLST command to set the postauthnenginewebcontext property.

   putStringProperty("/authnengines/postauthnenginewebcontext", "/plugin")

5. Execute the `putStringProperty()` WLST command to set the postauthnenginewebpath property.

   putStringProperty("/authnengines/postauthnenginewebpath", "/cookieextract.jsp")

6. Execute the `putBooleanProperty()` WLST command to enable the post-processing plug-in.

   putBooleanProperty("/authnengines/postauthnengineenabled", "true")

Part IV
Developing with Federation Protocol OAuth

This part discusses steps to develop a custom response plugin using Federation Protocol.

It contains the following chapters:

- Developing a Custom OAuth Plug-in

12

Developing a Custom OAuth Plug-in

With OAM 14c, OAuth tokens can have claims defined using policy responses. As a result, claims can be defined in an OAuth client or resource server definition using the user session (`$session`) and LDAP entry (`$user`) namespaces. For example, you can specify `$session.authn_level` or `$user.attr.givenname` in the OAuth client configuration. There can be use cases where an OAuth token must have claims retrieved from an external source. For example, the token must have claims provided by a REST endpoint.

The requirement here is to retrieve claims and inject the three claims listed here into Issued OAuth (JWT) token. To achieve this we must use a custom responses plugin.

Claim name	Claim value
my_plugin_claim1	myvalue1
my_plugin_claim2	myvalue2
my_plugin_claim3	myvalue3

The process of developing a custom response plugin involves following steps:

1. Develop a custom response plugin
2. Package and deploy the custom code as an OAM plugin
3. Register a plugin for use in an identity domain
4. Specify the custom claims in a client/resource server definition

Develop a Custom Response Plugin

The process of compiling a custom responses plugin is similar to a customer federation user provisioning plugin. For more information, see Developing a User Provisioning Plug-in.

Code snippets of `OAuthCustomClaimsPlugin` is as follows:

```
public class OAuthCustomClaimsPlugin extends ResponsesProviderPlugin{
  public ExecutionStatus initialize(PluginConfig config) {
        //Use initialization parameters for the plugin.
                super.initialize(config);

                //parameter REST_URL is not used. This is only for example.
                Object tmp = config.getParameter("REST_URL");
                LOGGER.finer("OAuthCustomClaimsPlugin initialize REST_URL
from configuration "+tmp);

                //If cacheResponsesInSession is false, responses will not be
cached in user session.
                cacheResponsesInSession = (String)
config.getParameter("cacheResponsesInSession");
                LOGGER.finer("OAuthCustomClaimsPlugin initialize
cacheResponsesInSession from configuration "+tmp);

                return ExecutionStatus.SUCCESS;
```

```
    }
@Override
        public ExecutionStatus execute(RequestContext requestContext,
Responses responses) {
                Object[] params = {requestContext.getUserId(),

requestContext.getOAuthClientID(),requestContext.getOAuthClientName(),

requestContext.getOAuthIdentityDomain(),getAsString(requestContext.getScopes()
)};
                LOGGER.logp(Level.FINER, "OAuthCustomClaimsPlugin","execute",
                                "userID: {0} "
                                + "client id: {1} client name:{2} identity
domain: {3} scopes:{4}", params);

                //Retrieve the custom claims
                //This is an example, in real world problem customer would
write custom code to retrieve
                // the claims for a user from the REST_URL.
                responses.setResponse("myclaim1", "myvalue1");
                responses.setResponse("myclaim2", "myvalue2");
                responses.setResponse("myclaim3", "myvalue3");

                //Cache claims in session
                //By default, claims are not cached in the user session.
                //After user obtains a token from an authz code, user session
can contain the claims returned by the plugin.
                //From an authz policy, this can be retrieved
using $session.attr.<idDomain>.<pluginName>.<claimnamefromplugin>
                if(cacheResponsesInSession != null &&
cacheResponsesInSession.equalsIgnoreCase("true")){
                        LOGGER.finer("enabling cache  ");
                        responses.setCacheInUserSession(true);
                }

                return ExecutionStatus.SUCCESS;
        }

}
```

Claims Cached in the User Session

When enabled, claims retrieved by the plugin can be cached in the User's session as part of the Authorization Code flow in the following format:

```
$session.attr.<idDomain>.<pluginName>.<claimnamefromplugin>
```

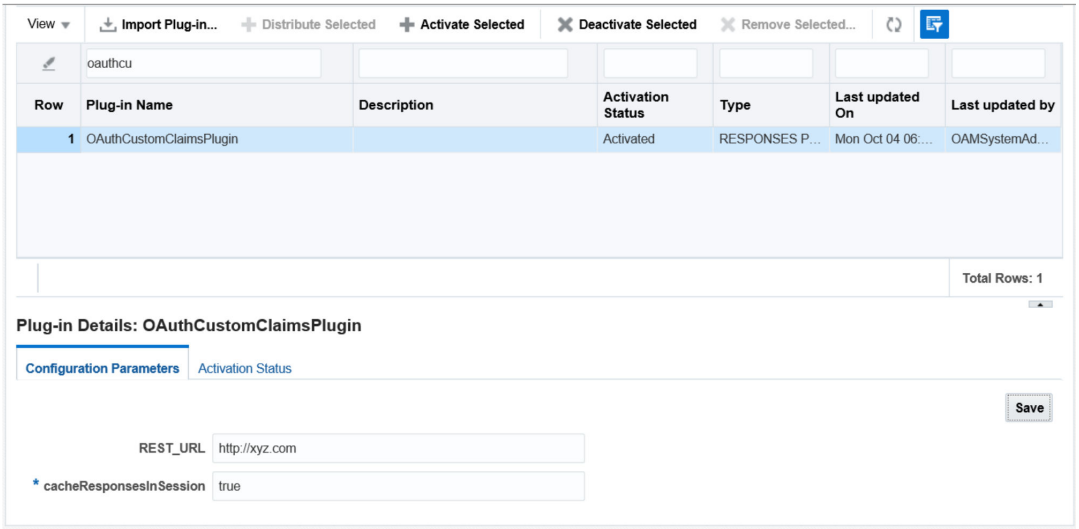
For more information you can refer to the claims in the session namespace (`$session`) in a Policy Response, Client or Resource Server definition.

> **Note:**
>
> Caching of claims in the session that is, as session attributes are disabled by default. Session attributes will be populated with claims from the plugin only if plugin sets `responses.setCacheInUserSession(true)`.

Package and Deploy the Custom Code as an OAM Plugin

- The process of deploying a custom responses plugin is similar to a custom federation user provisioning plugin. For more information, see Developing a User Provisioning Plug-in.
- Import, distribute, and activate the plugin in OAM console.

Figure 12-1 Deploy Plugin



> **Note:**
>
> Plugins are executed in a sandbox (oSGi) that ensures isolation from the OAM Server at runtime.

Register a Plugin for use in an Identity Domain

The list of allowed plugins are defined at an identity domain level using `customAttrs` key-value pair as JASON string.

The following example shows payload of an identity domain update request:

```
{
    "name": "IDDomainName",
    "identityProvider": "oidoci",
    "description": "Updated Domain",
    "tokenSettings": [
        {
            "tokenType": "ACCESS_TOKEN",
            "tokenExpiry": 3600,
```

```
            "lifeCycleEnabled": false,
            "refreshTokenEnabled": true,
            "refreshTokenExpiry": 86400,
            "refreshTokenLifeCycleEnabled": false
        },
        {
            "tokenType": "AUTHZ_CODE",
            "tokenExpiry": 3600,
            "lifeCycleEnabled": false,
            "refreshTokenEnabled": true,
            "refreshTokenExpiry": 86400,
            "refreshTokenLifeCycleEnabled": false
        },
        {
            "tokenType": "SSO_LINK_TOKEN",
            "tokenExpiry": 3600,
            "lifeCycleEnabled": false,
            "refreshTokenEnabled": true,
            "refreshTokenExpiry": 86400,
            "refreshTokenLifeCycleEnabled": false
        }
    ],
    "errorPageURL": "/oam/pages/servererror.jsp",
    "consentPageURL": "/oam/pages/consent.jsp",
    "customAttrs": "{\"allowedCustomPlugins\":\"OAuthCustomClaimsPlugin\"}"
}
```

Specify Custom Claims in a Client or Resource Server Definition

Create OAuth client to use the above plugin. The payload of OAuth client creation contains reference to claim name, plugin name, and plugin attribute.

```
{
    "attributes": [
       {
        "attrName": "my_plugin_claim1",
        "attrValue": "$plugin.OAuthCustomClaimsPlugin.myclaim1",
        "attrType": "DYNAMIC"
        },{
        "attrName": "my_plugin_claim3",
        "attrValue": "$plugin.OAuthCustomClaimsPlugin.myclaim3",
        "attrType": "DYNAMIC"
        }
    ],
    "secret": "mysecret",
    "id": "OAuthClientID",
    "scopes": [

"ResourceServer1.scope3","ResourceServer1.scope2","ResourceServer1.scope1"
    ],
    "clientType": "CONFIDENTIAL_CLIENT",
    "idDomain": "IDDomain",
    "description": "Client Description",
    "name": "OAuthClientName",
    "grantTypes": [
```

```
"CLIENT_CREDENTIALS","PASSWORD","JWT_BEARER","REFRESH_TOKEN","AUTHORIZATION_CO
DE"
    ],
    "defaultScope": "ResourceServer1.scope1",
    "redirectURIs": [
        {
            "url": "http://localhost:8080/Sample.jsp",
            "isHttps": true
        }
    ],


........
........

}
```

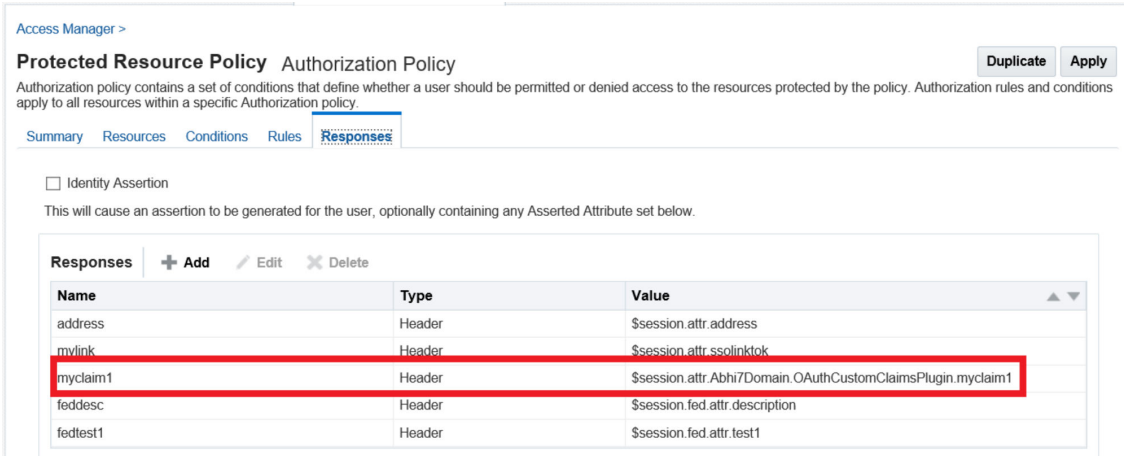
Test 3-legged flow. After getting the user token from and authorization code, following access token is received:

```
{
    "iss": "http://oamserverhost:<OAM_WLS_MANAGEDSERVER_PORT>/oauth2",
    "aud": [
        "ResourceServer1",
        "http://oamserverhost:<OAM_WLS_MANAGEDSERVER_PORT>/oauth2"
    ],
    "exp": 1633332319,
    "jti": "mTswLkuKKMknpVcd_p9oEg",
    "iat": 1633328719,
    "sub": "user1",
    "client": "OAuthClientID",
    "scope": [
        "ResourceServer1.scope1"
    ],
    "domain": "IDDomain",
    "grant": "AUTHORIZATION_CODE",
     .......
    "my_plugin_claim3": "myvalue3",
    "my_plugin_claim1": "myvalue1",
     ......
}
```

Sample Screenshots

The following screen shows the use of cached custom claim in an authorization policy response.

Figure 12-2 Authorization Policy Response Screen



The following screen shows CGI scripts printing the header information.

Figure 12-3 CGi Script Screen

Part V
Appendices

Information that is outside the scope of day-to-day developer tasks with Oracle Access Management is discussed here.

This part contains the following appendices:

- Creating Deployment-Specific Pages

A

Creating Deployment-Specific Pages

Oracle Single Sign-On provides a framework for integrating deployment-specific login, change password, and single sign-off pages with the single sign-on server. This means that you can tailor these pages to your UI look and feel and globalization requirements.

Oracle recommends that you use JavaServer (JSP) pages. Other Web technologies may provide inconsistent results. PLSQL pages are not supported. Sample pages are provided with the product. The Oracle Single Sign-On product ships with sample pages that are designed for testing with the Oracle Application Server.

This chapter contains the following topics:

A.1 How the Single Sign-On Server Uses Deployment-Specific Pages

The process that enables single sign-on pages can be summarized as follows:

1. The user requests a application and is redirected to the single sign-on server.
2. If the user is not authenticated, the single sign-on server redirects the user to the sample login page or to a deployment-specific page. As part of the redirection, the server passes to the page the parameters contained in Table A-2.
3. The user submits the login page, passing the parameters contained in Table A-3 to the authentication URL:

   `http://sso_host:sso_port/oam/server/auth_cred_submit`

   or

   `https://sso_host:sso_ssl_port/oam/server/auth_cred_submit`

   At least two of these parameters, `ssousername` and `password`, appear on the page as modifiable fields.
4. If authentication fails, the server redirects the user back to the login page and displays an error message.
5. To finish the single sign-on session, the user clicks **Logout** in the application he or she is working in. This act calls application logout URLs in parallel, logging the user out from all accessed applications and ending the single sign-on session.

6. The user is redirected to the single sign-on server, which presents the single sign-off page.

A.1.1 Change Password Page Behavior

Users who try to log in when their passwords have expired or are about to expire experience the following server behavior.

Table A-1 Change Password Page messages

Message	Description
Password Has Expired	Users are shown the password expiry page. User must enter the old and the new password. The new password must conform to the Access Manager password policy rules.
Password is About to Expire	A warning page is displayed where the user can either change their password, or continue without changing before continuing.
Grace Login Is in Force	Same behavior as when password is about to expire.
Force Change Password	This feature prompts users to change their password after it has been reset by an administrator. The reset is required after the attribute `obpasswordchangeflag` is set to 1. Once the attribute is set, the user is required to change the password at next login.

A.2 How to Write Deployment-Specific Pages

The URLs for login, change password, and single sign-off pages must accept the parameters described in the tables that follow if these pages are to function properly.

This section contains the following topics:

- Login Page Parameters
- Change Password Page Parameters

A.2.1 Login Page Parameters

The URL for the login page must accept the parameters listed in Table A-2.

Table A-2 Login Page Parameters Submitted to the Page by the Single Sign-On Server

Parameter	Description
`p_error_code`	Contains the error code in the form of a string. Passed when an error occurs during authentication.
`request_id`	Unique identifier that is used to track requests routed back and forth between client and server.
`OAM_REQ`	User login request context tracked at client until authentication process is completed.

The login page must pass the parameters listed in Table A-3 to the authentication URL:

`http://`*`sso_host`*`:`*`sso_port`*`/sso/auth`

Table A-3 Login Page Parameters Submitted by the Page to the Single Sign-On Server

Parameter	Description
`ssousername`	Contains the username. Must be UTF-8 encoded.
`password`	Contains the password entered by the user. Must be UTF-8 encoded.
`OAM_REQ`, if present in request	User login request context tracked at client until authentication process is completed.
`request_id`, if present in request	Unique identifier that is used to track requests routed back and forth between client and server.

The login page must have at least two fields: a text field with the parameter name `ssousername` and a password field with the parameter name `password`. The values are submitted to the authentication URL.

In addition to submitting these parameters, the login page is responsible for displaying appropriate error messages, as specified by `p_error_code`, redirecting to `p_cancel_url` if the user clicks **Cancel**.

A.2.2 Change Password Page Parameters

27 June: Lakshmi advises: "Is this not same as custom UI pages.. The intention seems to be the same to me." 22Jun2012: Dev advises that this behavior has changed in R2. Awaiting details from dev to update section.

The URL for the change password page must accept the parameters listed in Table A-4.

> **Note:**
>
> In a GIT deployment, when a partner logout flow requires query parameters in the `p_done_url`, the parameters must be URL encoded such that the Access Manager logout servlet does not interpret them as being Access Manager parameters but elements of the single `p_done_url`.

Table A-4 Change Password Parameters Submitted to the Page

Parameter	Description
`p_username`	Contains the user name to be displayed somewhere on the page.
`p_subscribername`	The subscriber nickname when hosting is enabled. Note: This field is required on the login page.
`p_error_code`	Contains the error code, in the form of a string, if an error occurred in the prior attempt to change the password.
`p_done_url`	Contains the URL of the appropriate page to return to after the password is saved.

Table A-4 (Cont.) Change Password Parameters Submitted to the Page

Parameter	Description
site2pstoretoken	Contains the site2pstoretoken that is required by the /sso/auth login URL if the password has expired or is about to expire.
p_pwd_is_exp	Contains the flag value indicating whether the password has expired or is about to expire. The value can be either WARN or FORCE. See Table A-6 for the associated error codes.
locale	User's language preference (optional). Must be in ISO format. For example, French is fr-fr. For more about this parameter, see "Adding Globalization Support".

The change password page must pass the parameters listed in Table A-5 to the change password URL:

http://*sso_host*:*sso_port*/sso/ChangePwdServlet

Table A-5 Change Password Page Parameters Submitted by the Page

Parameter	Description
p_username	Contains the user name to be displayed somewhere on the page. Should be posted as a hidden field by the change password page. Must be UTF-8 encoded.
p_old_password	Contains the user's old password. Must be UTF-8 encoded.
p_new_password	Contains the user's new password. Must be UTF-8 encoded.
p_new_password_confirm	Contains the confirmation of the user's new password. Must be UTF-8 encoded.
p_done_url	Contains the URL of the appropriate page to return to after the password is saved.
p_pwd_is_exp	Contains the flag value indicating whether the password has expired or is about to expire. The value can be either WARN or FORCE. See Table A-6 for the associated error codes.
site2pstoretoken	Contains the redirect URL information for login processing.
p_action	Commits changes. The values must be either OK (commit) or CANCEL (ignore).
p_subscribername	Contains the user name to be displayed somewhere on the page.
p_request	Protected URL requested by the user.
locale	User's language preference (optional). Must be in ISO format. Example: French is fr-fr. See "Adding Globalization Support".

The change password page must have at least three password fields: p_old_password, p_new_password, and p_new_password_confirm. The page should submit these fields to the change password URL.

The page should also submit `p_done_url` as a hidden parameter to the change password URL. In addition, it should display error messages according to the value of `p_error_code`.

A.3 Page Error Codes

URLs for login and change password pages must accept the process errors described in the tables that follow if these pages are to function properly.

When OAM Server is set up, the login page must process the error codes listed in Table A-6.

Table A-6 Login Page Error Codes

Value of p_error_code	Corresponding message and description
`acct_lock_err`	Description: The user has committed too many login failures. Message: "Your account is locked. Please notify the system administrator."
`pwd_exp_err`	Description: The user's password has already expired. Message: "Your password has expired. Please contact the administrator to reset it."
`null_uname_pwd_err`	Description: The user left the user name field blank. Message: "You must enter a valid user name."
`auth_fail_exception`	Description: Authentication has failed. Message: "Authentication failed. Please try again."
`null_password_err`	Description: The user left the password field blank. Message: "You must enter your logon password."
`sso_forced_auth`	Description: The application requires authentication. Message: "The application you are trying to access requires you to sign in again even if you have signed in previously."
`unexpected_exception`	Description: An unexpected error occurred during authentication. Message: "An unexpected error occurred. Please try again."
`unexp_err`	Description: Unexpected error. "Unexpected Error. Please contact Administrator."
`internal_server_err`	Description: Internal server error report. Message: "Internal Server Error. Please contact Administrator."
`internal_server_try_again_err`	Description: Internal server error report with "try again" prompt. Message: "Internal Server Error. Please retry the operation."
internal_server_try_later_err	Description: Internal server error report with "try later" prompt. Message: "Internal Server Error. Please try the operation later."

Table A-6 (Cont.) Login Page Error Codes

Value of p_error_code	Corresponding message and description
gito_err	Description: Inactivity timeout. User must log in again. Message: "Your Single Sign_on session has expired. For your security, your session expires after some duration of inactivity. Please sign in again."
cert_auth_err	Description: Certificate sign-on has failed. User should check that the certificate is valid or should contact the administrator. Message: "Certificate-based sign in failed. Please ensure that you have a valid certificate or contact the administrator."
session_exp_error	Description: Single sign-on session time limit reached. Message: "Your Single Sign-On session has expired. For your security, your session expires after the specified amount of time. Please sign in again."
userid_mismatch	Description: The user ID presented during a forced authentication does not match the user ID in the current single sign-on session. Message: "The user name submitted for authentication does not match the user name present in the existing Single Sign-On session."

A.4 Adding Globalization Support

The OracleAS Single Sign-On framework enables you to globalize deployment-specific pages to fit the needs of your deployment. When deciding what language to display the page in, you can adopt different strategies. Two strategies are presented in the following sections.

- Deciding What Language to Display the Page In
- Rendering the Page

A.4.1 Deciding What Language to Display the Page In

This section explains how to use either the HTTP Accept-Language header or deployment page logic to choose a language to display. The two strategies are explained in the following sections.

- Use the Accept-Language Header to Determine the Page
- Use Page Logic to Determine the Language

A.4.1.1 Use the Accept-Language Header to Determine the Page

Browsers enable end users to decide the language (locale) they would like to view their Web content in. The browser sends the language that the user chooses to the server in the form of the HTTP Accept-Language header. The logic of the deployment-specific page must examine this header and render the page accordingly. When it receives this page, the single sign-on server takes note of the header value for Accept-Language and sends it to applications when it propagates the user's identity. Note that, although many applications enable users to override this header, the single sign-off page appears in the language established at sign-on. The net effect is a consistent session language for all applications.

The Accept-Language header is the preferred mechanism for determining the language preference. A major benefit of this approach is that end users have typically already set their language preference while browsing other Web sites. The result is browsing consistency between these pages and single sign-on pages.

A.4.1.2 Use Page Logic to Determine the Language

Although Oracle recommends the approach described in the preceding section, you may choose to implement globalization based on mechanisms that extend or override the language preference set in the browser. You may, for instance, do one of the following:

- Display a list of languages on the login page and allow the user to select from this list. As a convenience to the user, you can make this selection persistent by setting a persistent cookie.
- Render the page in one, fixed language. This method is appropriate when you know that the user population is monolingual.
- Obtain language preferences from a centralized application repository or a directory. A centralized store for user and system preferences and configuration data is ideal for storing language preferences.

If you use page logic to set language preferences, the page must propagate this information to the single sign-on server. The server must propagate this information to applications. The net result is a consistent globalization experience for the user. Your page must pass the language in ISO-639 format, using the `locale` parameter (Table A-3) in the login form. A number of sites contain a full list of ISO-639 two-letter language codes. Here is a site that contains a full list of ISO-3166 two-letter country codes:

```
http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html
```

A.4.2 Rendering the Page

Once it determines the end-user's locale, the deployment-specific page must use the corresponding translation strings to render the page. To learn how to store and retrieve these strings, see the chapter about locale awareness in *Oracle Application Server Globalization Guide*. You may also want to consult standard documents about Java development. Here are two links:

- Java Internationalization Guide:

  http://java.sun.com/j2se/1.4.2/docs/guide/intl/index.html
- General link for Java documentation:

  http://java.sun.com/j2se/1.4.2/docs

A.5 Guidelines for Deployment-Specific Pages

When implementing deployment-specific pages, observe the following guidelines:

- Oracle recommends that login and change password pages be protected by SSL.
- The login and change password pages must code against cross-site scripting attacks.
- The login and change password pages must have auto-fill and caching set to `off`. This prevents user credentials from being saved or cached in the browser. Here is an example of the `AutoComplete` tag:

  ```
  <FORM NAME="foo" AutoComplete="off" METHOD="POST" ACTION="bar">
  ```

- Oracle recommends that you configure your login page to display a banner that warns against unauthorized access. You may, for example, want to use the following text or a variant thereof:

  ```
  Unauthorized use of this site is prohibited and may subject you to civil and
  criminal prosecution.
  ```

- Deploy the login and change password pages on the computer that hosts the single sign-on server. This makes it easier to detect false versions of these pages.

A.6 Customized Deployment-Specific Pages

The following sections describe customizing deployment-specific pages.

- Customizing Deployment-Specific Pages
- Using Custom Classes

A.6.1 Customizing Deployment-Specific Pages

The `ipassample.jar` file contains the files `login-ex.jsp`, `password-ex.jsp`, and `signoff-ex.jsp`. You may customize these to suit your deployment. If you want to use these files. Use this command to extract the file:

```
ORACLE_HOME/jdk/bin/jar -xvf ORACLE_HOME/sso/lib/ipassample.jar
```

A.6.2 Using Custom Classes

In general, customized deployment-specific pages must operate with the current versions of component classes in use by OC4J_SECURITY. If your custom application needs to use a different version of a given class, you must deploy that class in a separate OC4J instance and *not* in the OC4J_SECURITY instance.

For example, if your deployment requires the use of custom log4j classes that conflict with the versions in use by OC4J_SECURITY, start a separate OC4J_SECURITY instance that uses a local log4j `jar` file containing the custom classes.

> **WARNING:**
>
> Replacing the classes used by OC4J_SECURITY with custom versions may render Oracle Single Sign-On or other Oracle Application Server components unusable.

A.7 Add External Applications Page

From the Single Sign-On Server Administration page, clicking the Administer External Applications link, then clicking Add External Application link takes you to the Add External Applications page. The following sections describe the Add External Applications Page:

- Headings and Fields of Add External Applications Page
- Adding an External Application

A.7.1 Headings and Fields of Add External Applications Page

This page contains the following headings and fields:

Table A-7 External Application Login

Field	Description
Application Name	Enter a name that identifies the external application. This is the default name for the external application.
Login URL	Enter the URL to which the HTML login page for the external application is submitted for authentication. This, for example, is the login URL for Yahoo! Mail: `http://login.yahoo.com/config/login?6p4f5s403j3h0`
Username/ID Field Name	Enter the term that identifies the user name or user ID field of the HTML login form for the application. You find this term by viewing the HTML source of the form. (See the example after the steps immediately following). This field is not applicable if you are using basic authentication.
Password Field Name	Enter the term that identifies the password field of the HTML login form for the application. You find this term by viewing the HTML source of the form. (See the example after the steps immediately following). This field is not applicable if you are using basic authentication.

Table A-8 Authentication Method

Field	Description
Type of Authentication Use	Use the pull-down menu to select the form submission method for the application. This method specifies how message data is sent by the browser. You find this term by viewing the HTML source for the login form. Select one of the following three methods: POST: Posts data to the single sign-on server and submits login credentials within the body of the form. GET: Presents a page request to a server, submitting the login credentials as part of the login URL. Basic authentication: Submits the login credentials in the application URL, which is protected by HTTP basic authentication. Notes: • Basic authentication uses pop-up windows, which by default are blocked by Windows XP, service pack 2. If you use this service pack, make sure that you reconfigure browser settings to display the window for the single sign-on login page. Use the pop-up blocker item in the Tools menu of Internet Explorer. Other browsers and browser plug-ins are able to block pop-ups. Mozilla is one of these. Make sure that these do not block the single sign-on login page. • If you use Internet Explorer 5.0 or a later version, basic authentication may not work with external applications. This version of Internet Explorer includes Microsoft MS04-004 Cumulative Security Update (832894). See this link for a workaround: http://support.microsoft.com

Table A-9 Additional Fields

Field	Description
Field Name	Enter the name of any additional fields on the HTML login form that may require user input to log in. This field is not applicable if you are using basic authentication.

Table A-9 (Cont.) Additional Fields

Field	Description
Field Value	Enter a default value for a corresponding field name value, if applicable. This field is not applicable if you are using basic authentication.

A.7.2 Adding an External Application

Perform the following task to add an external application:

1. From the Administer External Applications page, select **Add External Application**.

   The Add External Applications page appears.

2. In the **External Application Login** field, enter the name of the external application and the URL to which the HTML login form is submitted. If you are using basic authentication, enter the protected URL.

3. If the application uses HTTP POST or HTTP GET authentication, in the **User Name/ID Field Name** field, enter the term that identifies the user name or user ID field of the HTML login form.

   You can find the name by viewing the HTML source of the login form.

   If the application uses the basic authentication method, the **User Name/ID Field Name** field should be empty.

4. If the application uses HTTP POST or HTTP GET authentication, in the **Password Field Name** field, enter the term that identifies the password field of the application.

   See the HTML source of the login form.

   If the application uses the basic authentication method, the **Password Field Name** field should be empty.

5. In the **Additional Fields** field, enter the name and default values for any additional fields on the HTML login form that may require user input.

   If the application uses the basic authentication method, these fields should be empty.

6. Select the **Display to User** check box to allow the default value of an additional field to be changed by the user on the HTML login form.

7. Click **OK**. The new external application appears under the **Edit/Delete External Application** heading on the Administer External Applications page, along with the other external applications.

8. Click the application link to test the login.

The following example shows the source of the values that are used for Yahoo! Mail.

```
<form method=post action="http://login.yahoo.com/config/login?6p4f5s403j3h0"
autocomplete=off name=a>
...
<td><input name=login size=20 maxlength=32></td>
....
<td><input name=passwd type=password size=20 maxlength=32></td>
...
<input type=checkbox name=".persistent" value="Y" >Remember my ID & password
...
</form>
```

The source provides values for the following:

- Login URL:

  `http://login.yahoo.com/config/login?6p4f5s403j3h0`

- Username/ID Field Name: `login`
- Password Field Name: `passwd`
- Type of Authentication Used: `POST`
- Field Name: `.persistent Y`
- Field Value: `[off]`

> **Note:**
>
> If you change the host name of the AS middle tier, you must manually update the Login URL field for external applications on this middle tier. You do this on the Edit External Applications page, described in the next section.