Oracle® Fusion Middleware

Developer's Guide for Oracle Access Management

11g Release 2 (11.1.2)

E27134-03

November 2012

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

ORACLE

Oracle Fusion Middleware Developer's Guide for Oracle Access Management, 11g Release 2 (11.1.2)

E27134-03

Primary Author: Trish Fuzesy

Contributing Author: Kevin Kessler, Michael Teger

Contributor: Toby Close, Vadim Lander, Peter Povinec, Umesh Waghode, Jeremy Banford, Sreehari Narasimhaiah

Contents

3 Developing Custom Authentication Plug-ins

4 Developing Custom Pages

8 Developing Mobile Services Applications with the Java Client SDK

11 Extending the Capabilities of the Mobile and Social Server

12 Sending Mobile and Social REST Calls With cURL

List of Examples

List of Tables

List of Figures

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 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 11g Release 2 (11.1.2) documentation set:

- *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*
- *Oracle Fusion Middleware Java API Reference for Oracle Access Management Security Token Service*
- *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 Java API Reference for Oracle Access Management Mobile and Social*
- *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*

Conventions

The following text conventions are used in this document:

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

What's New in Oracle Access Management?

This section describes new features available in Oracle Access Management 11*g*.

The following sections describe the new features and changes:

Guide Changes: 11*g* Release 2 (11.1.2) November 2012 Library Refresh

This guide (Part no. E27134-03) has been updated in several ways. Following are the sections that have been added or changed.

- Revised information about multi-step authentication plug-in development and added code examples. See Chapter 3.

New Features in 11*g* Release 2 (11.1.2)

Oracle Access Management 11g Release 2 (11.1.2) includes the following components. The new features discussed in this guide are described in the following sections.

Oracle Access Management Access Manager

This release adds the following functionality to the Access Manager Access software development kit (SDK):

- Support for 11*g* cookies

  Access Clients developed with the SDK can use the 11*g* agent profile, enabling the OAM Server to encrypt tokens using a secret key generated specifically for this Access Client. For more information, see Chapter 2.

- API based initialization

Access Clients developed with the SDK can initialize by providing boot strap configuration from its own configuration store or mechanism. For more information, see Chapter 2.

- Interfaces for developing Web SSO agents

  Provides simple interfaces to enable WebSSO agents to work with Access Manager. For more information, see Chapter 2.

The following Access Manager APIs have been added:

- Policy Administration API

  The Oracle Policy Administration API supports representational state transfer (REST) interfaces for administering OAM policy objects as RESTful resources. The Policy Administration API enables Create, Read, Update, and Delete (CRUD) operations on policy objects. For more information, see Chapter 5.

Oracle Access Management Mobile and Social

Mobile and Social is a new Oracle Access Management service that acts as an intermediary between a user seeking to access protected resources, and the backend Access Management and Identity Management services that protect the resources. Mobile and Social provides simplified client libraries that allow developers to quickly add feature-rich authentication, authorization, and identity capabilities to registered applications. On the backend, the Mobile and Social server's pluggable architecture lets system administrators add, modify, and remove Identity and Access Management services without having to update user installed software. Mobile and Social features individual SDKs for iOS devices and Java. If you are developing an application on a platform or device that cannot use the iOS or Java SDKs, you can write code to directly send Mobile and Social REST calls to the Mobile and Social server. For more information about Mobile and Social, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*. For information about developing applications using Mobile and Social SDKs, see Part III, "Developing with Mobile and Social".

Oracle Access Management Identity Federation

This release adds the User Provisioning API to Security Token Service. Use this API to develop a custom user provisioning plug-in. For more information, see Chapter 13.

Oracle Access Management Security Token Service

There are no changes to Security Token Service APIs in this release.

New Features in 11*g* Release 1 (11.1.1)

11*g* Release 1 provides a pure Java software developer kit (SDK) for the creation of custom Access Clients and extensions of authentication and authorization functionality. This release also provides compatibility with the Oracle Access Manager 10*g* JNI SDK, which can be migrated to use the Oracle Access Manager 11*g* release.

Product and Component Name Changes

Many Oracle Access Manager component names remain the same. However, there are several important changes. For more information, see "What's New in Oracle Access Management" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

Part I

Introduction

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

Part I contains the following chapters:

1

Developing with Oracle Access Management Components

This chapter introduces the Oracle Access Management components and developing with the software development kits (SDKs) and application programming interfaces (APIs). This chapter provides the following sections:

1.1 About Oracle Access Management

Oracle Access Management release 11.1.2 provides converged multi-services with several integrated components:

- Authentication and SSO, provided by Oracle Access Management Access Manager (Access Manager)
- Federated SSO, provided by Oracle Access Management Identity Federation (Identity Federation)
- Mobile security and social identity, provided by Oracle Access Management Mobile and Social (Mobile and Social)
- Security Token Service, provided by Oracle Access Management Security Token Service (Security Token Service)

You can developing applications to customize your environment or otherwise extend functionality using the Oracle Access Management component supplied SDKs and APIs, related Javadocs, and this guide.

1.2 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 more information about Access Manager, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

For information about developing applications using Access Manager SDKs and APIs, see Part II, "Developing with Access Manager".

1.3 About Mobile and Social

Mobile and Social acts as an intermediary between a user seeking to access protected resources, and the backend Access Management and Identity Management services that protect the resources. Mobile and Social provides simplified client libraries that allow developers to quickly add feature-rich authentication, authorization, and Identity capabilities to registered applications. On the backend, the Mobile and Social server's pluggable architecture lets system administrators add, modify, and remove Identity and Access Management services without having to update user installed software. Mobile and Social features individual SDKs for iOS devices and Java. If you are developing an application on a platform or device that cannot use the iOS or Java SDKs, you can write code to directly send Mobile and Social REST calls to the Mobile and Social server.

For more information about Mobile and Social in Oracle Access Management, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

For information about developing applications using Mobile and Social SDKs, see Part III, "Developing with Mobile and Social"

1.4 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 initial release, convergence is limited to Service Provider functionality. In this initial release any Identity Provider functionality still requires a Oracle Identity Federation 11*g*R1 installation. However, the linking of Oracle Access Management 11*g*R2 and Oracle Identity Federation 11*g*R1 is very simple and well integrated.

For more information about Identity Federation in Oracle Access Management, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

In this release, you can develop a custom user provisioning plug-in if the out-of-the-box solution does not meet your needs. For more information about developing applications with Identity Federation APIs, see Part IV, "Developing with Identity Federation".

1.5 About Security Token Service

Security Token Service is a standards-based security solution that issues, validates, or exchanges security tokens and acts as a trusted authority that an enterprise web services infrastructure may use to enforce appropriate security token policies across web services providers and consumers. It also provides a means for propagating identity and security information across infrastructure tiers.

For more information about Security Token Service in Oracle Access Management, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

In this release, when Security Token Service does not support the token that you want to validate and is not provided out-of-the-box, you can write your own validation and issuance module classes. For more information about developing tokens with Security Token Service, see Part V, "Developing with Security Token Service".

1.6 System Requirements and Certification

Refer to the system requirements and certification documentation for information about hardware and software requirements, platforms, databases, and other information. Both of these 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

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

Part II 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

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 11*g* 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 11*g* Access SDK and APIs are intended for Java application developers for the development of tightly coupled, performant integrations. In addition to this guide, for

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

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. It is provided on JDK versions that are supported across Oracle Fusion Middleware applications.

The `oracle.security.am.asdk` package provides the 11*g* Java APIs. The 11*g* version is very similar to the 10*g* JNI APIs, with enhancements for use with the 11*g* OAM Server. The 11*g* Access SDK provides backwards compatibility by supporting 10*g* based `com.oblix.access` interfaces. From a functional perspective, the 11*g* Access SDK maintains parity with the 10*g* (10.1.4.3) Access SDK to ensure that you can re-write existing custom code using the 11*g* API layer.

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

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

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. The following Access SDK packages are included:

- **oracle.security.am.asdk**: An authentication and authorization API that provides enhancements to take advantage of 11*g* OAM Server functionality. The 11*g* Access SDK API can be used with either Oracle Access Manager 10*g*R3 (10.1.4.3) or Oracle Access Manager 11*g*R1 (11.1.1.5+) version of the server.
- **com.oblix.access**: This is the 10*g* version of the authentication and authorization API with some enhancements for the 11*g* release. It is available for backward compatibility with programs written with the 10*g* JNI ASDK.

The 11*g* Access SDK includes authentication and authorization functionality. However, it does not include Administrative APIs (for instance, there is no 11*g* 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.

Table 2–1 describes the primary features of the 11*g* Access SDK.

Table 2–1 11g 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 11*g* 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. **Note:** Access Clients and plug-ins developed with Oracle Access Manager 10*g* (10.1.4.3) can be used with 11*g* release. Oracle Access Manager 10*g* (10.1.4.3) bundle patches are used to distribute Java SDK code enhancements for use with 11*g* environments.
Built In Versioning	Enables you to: ■ Determine the Access SDK version that is installed. ■ Validate compatible versions it can operate with (Oracle Access Manager 10*g* (10.1.4.3) and 11*g*). 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 Installing Access SDK

To install the Java Access SDK Client for Access Manager 11*g*, perform the following steps:

1. Download the `oam-java-asdk.jar` 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. Choose from the following depending upon your needs:
   - If you are using a non-JRF environment, add the following jar files to your CLASSPATH:

     ```
     identitystore.jar
     jps-api.jar
     jps-common.jar
     jps-internal.jar
     jps-unsupported-api.jar
     oraclepki.jar
     osdt_cert.jar
     osdt_core.jar
     osdt_xmlsec.jar
     ```

   - If you are configuring an Access Client in 10*g* mode, add the following jar files to your CLASSPATH:

     ```
     jps-api.jar
     ```

   - If you are configuring an Access Client in 11*g* mode, add the following jar files to your CLASSPATH:

     ```
     jps-api.jar
     jps-common.jar
     jps-internal.jar
     jps-unsupported-api.jar
     ```

```
    oraclepki.jar
    osdt_cert.jar
    osdt_core.jar
    osdt_xmlsec.jar
```

4. If you are using a JRF environment, add the follow to system-jazn-data.xml in order to run in `OAM_11G` mode:

```
<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>
```

2.1.3 About Custom Access Clients

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.

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

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

- Authenticate users by validating their credentials against 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 11*g*, 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 *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

2.1.3.1 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.

2.1.3.2 Access Client Architecture

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

1. Custom Access Client code.

   Built into a servlet or standalone application. For the 11*g* release, you write Access Client code using the Java language platform.

2. Configuration information.

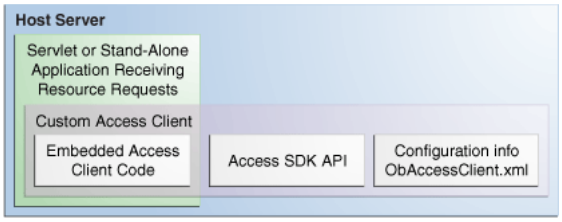
   - **OBAccessClient.xml file**: Primary configuration file, which contains configuration information that constitutes an Access Client profile.
   - **cwallet.sso** and **jps-config.xml** files: For an 11*g* agent only.
   - If the transportation security mode is Simple or Cert, then the following files 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 unprotect private key file.

3. 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



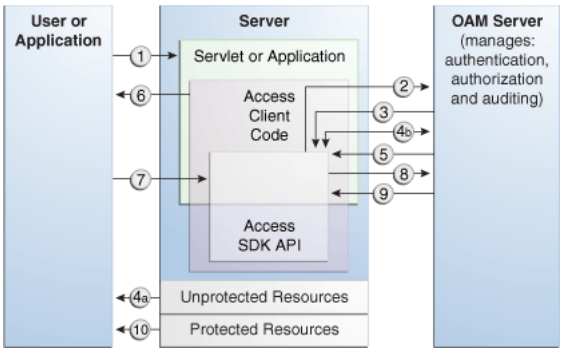
2.1.4 About Access Client Request Processing

Regardless of the variability introduced by the types of resources discussed in Section 2.1.3.2, "Access Client Architecture", most Access Clients follow the same basic steps to process user requests.

When a user or application submits a resource request to a servlet or application running on the server where the Access Client is installed, the Access Client code embedded in that servlet or application initiates the basic process shown in Figure 2–2.

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

Figure 2–2 Process Overview: Handling a Resource Request



Process Overview: Handling a Resource Request

1. The application or servlet containing the Access Client code receives a user request for a resource.
2. The Access Client constructs 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 11*g* 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 either in 10*g* or 11*g* 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 Developing Access Clients

The following topics are discussed in this section:

2.2.1 Structure of an Access Client

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

Access Client Application Structure Overview

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

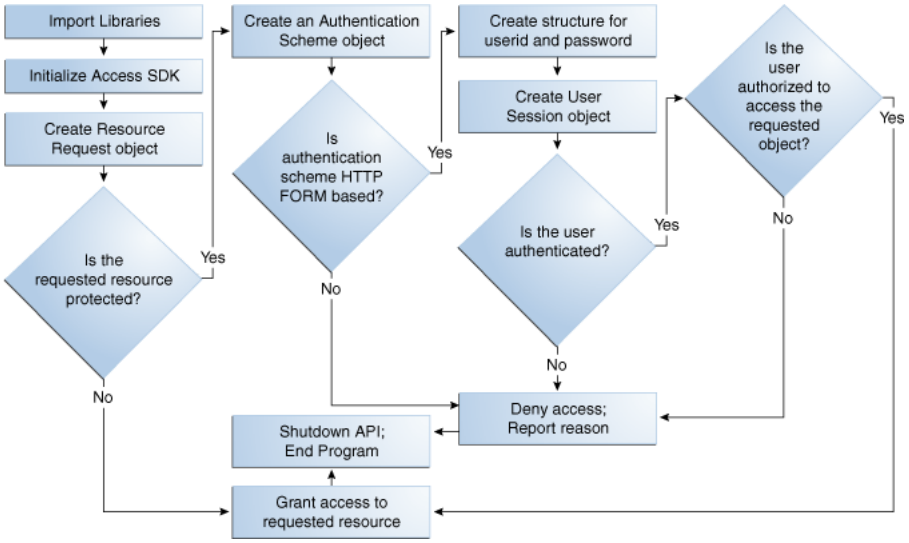
7. Clean up (Java uses automatic garbage collection).
8. Shut down.

2.2.2 Typical Access Client Execution Flow

All HTTP FORM-based Access Client applications and plug-ins follow the same basic pattern, as illustrated Figure 2–3.

Figure 2–3 shows a process flow for form-based applications:

Figure 2–3 Process Flow for Form-based Applications



Process Overview: Access Client Execution for Form-based Applications

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

   Resource Not Protected: 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 11*g* 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 *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

2.2.3 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 Section 2.2.4, "Annotated Sample Code: Simple Access Client" for an annotated version of this example to help you become familiar with 11*g* 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 ms_resource = "//Example.com:80/secrets/
         index.html";
   public static final String ms_protocol = "http";
   public static final String ms_method = "GET";
   public static final String ms_login = "jsmith";
   public static final String ms_passwd = "j5m1th";
   public static final String m_configLocation = "/myfolder";
   public static void main(String argv[]) {
  AccessClient ac = null;
      try {
      ac = AccessClient.createDefaultInstance(m_configLocation,
 AccessClient.CompatibilityMode.OAM_10G);

         ResourceRequest rrq = new ResourceRequest(ms_protocol, ms_resource,
               ms_method);
         if (rrq.isProtected()) {
```

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

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

Launch the main method on the Java interpreter. An array of strings named `argv` is passed to the main method. In this particular case, the user `jsmith`, whose password is `j5m1th`, has requested the HTTP resource `//Example.com:80/secrets/index.html`. GET is the specific HTTP operation that will be performed against the requested resource. For details about supported HTTP operations and protecting resources with application domains, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

```
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 configuration file ObAccessClient.xml. 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*.

You only need to create an instance of `AccessClient` and it initializes Access SDK API. `AccessClient.CompatibilityMode.OAM_10G` indicates that Access SDK will be initialized to work in an older 10*g* agent mode that is compatible with both the 10*g* and 11*g* servers. By default, if this compatibility mode is not provided, then default `OAM_11G` is used, and the agent will be operating in 11*g* agent mode and can only talk with 11*g* OAM Servers.

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

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

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

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

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

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

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

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

If the resource is protected, report that fact.

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

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

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

Determine if the authorization scheme is FORM-based.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
} 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.

2.2.5 Sample Code: Java Login Servlet

This example follows the basic pattern of API calls that define an Access Client, as described in Section 2.2.3, "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 the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

This example Java servlet does not provide SSO to resources protected by mod_osso, Access Manager Webgates, or OpenSSO Policy Agents.

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. Section 2.2.6, "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" ,
 AccessClient.CompatibilityMode.OAM_10G);
      } catch (AccessException ae) {
         ae.printStackTrace();
      }
   }

   public void service(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
      AuthenticationScheme authnScheme = null;
      UserSession user = null;
      ResourceRequest resource = null;
       response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HTML>");
      out.println("<HEAD><TITLE>LoginServlet: Error Page</TITLE></HEAD>");
      out.println("<BODY>");
      HttpSession session = request.getSession( false);
      String requestedPage = request.getParameter("request");
      String reqMethod = request.getMethod();
      Hashtable cred = new Hashtable();
      try {
         if (requestedPage == null || requestedPage.length()==0) {
            out.println("<p>REQUESTED PAGE NOT SPECIFIED\n");
            out.println("</BODY></HTML>");
            return;
         }
```

```
            resource = new ResourceRequest("http", requestedPage, "GET");
            if (resource.isProtected()) {
               authnScheme = new AuthenticationScheme(resource);
               if (authnScheme.isBasic()) {
                  if (session == null) {
                     String sUserName = request.getParameter("userid");
                     String sPassword = request.getParameter("password");
                     if (sUserName != null) {
                        cred.put("userid", sUserName);
                        cred.put("password", sPassword);
                        user = new UserSession(resource, cred);
                        if (user.getStatus() == UserSession.LOGGEDIN) {
                           if (user.isAuthorized(resource)) {
                              session = request.getSession( true);
                              session.putValue( "user", user);
                              response.sendRedirect( requestedPage );
                           } else {
                              out.println("<p>User " + sUserName + " not" +
                                   " authorized for " + requestedPage + "\n");
                           }
                        } else {
                           out.println("<p>User" + sUserName + "NOT LOGGED IN\n");
                        }
                     } else {
                        out.println("<p>USERNAME PARAM REQUIRED\n");
                     }
                  } else {
                     user = (UserSession)session.getValue("user");
                     if (user.getStatus() == UserSession.LOGGEDIN) {
                        out.println("<p>User " + user.getUserIdentity() + " already"+
                              "LOGGEDIN\n");
                     }
                  }
               } else {
                  out.println("<p>Resource Page" + requestedPage + " is not"+
                        " protected with BASIC\n");
               }
            } else {
               out.println("<p>Page " + requestedPage + " is not protected\n");
            }
         } catch (AccessException ex) {
         out.println(ex);
         }
      out.println("</BODY></HTML>");
      }
   }
```

2.2.6 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*. The `OAM_10G` compatibility flag initializes Access SDK in a mode such that it is compatible with both 10*g* and 11*g* servers The `OAM_10G` compatibility flag initializes Access SDK in an old 10*g* agent mode that is compatible with both 10*g* and 11*g* servers. By default, if this compatibility mode is not provided, then default `OAM_11G` flag is used and the agent will operate in 11*g* agent mode and can only talk with 11*g* OAM Server.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Additional Code for authnScheme.isForm

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

1. Process the original request and determine that form-based login is required.

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

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

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

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

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

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

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

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

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

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

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

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

Redirect the user's browser to the target page.

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

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

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

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

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

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

```
 } else {
```

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

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

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

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

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

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

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

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

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

If an error occurs, report the backtrace.

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

Complete the output stream to the user's browser.

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

2.2.7 Sample Code: Additional Methods

Building on the basic pattern established in the sample application `JAccessClient.java`, discussed in Section 2.2.3, "Sample Code: Simple Access Client", the following sample program invokes several additional OAM Server methods. For instance, it inspects the session object to determine which actions, also named responses, are currently configured in the policy rules associated with the current authentication scheme.

For this demonstration to take place, you must configure some actions through the OAM Server prior to running the application. For details about authentication action and configuring user authentication, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*. The complete listing for this sample application appears in Example 2–3. An annotated version of the code is provided in Section 2.2.8, "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 ,
 AccessClient.CompatibilityMode.OAM_10G);

    } catch (AccessException ae) {
      System.out.println("OAM Server SDK Initialization
 failed");
      ae.printStackTrace();
      return;
    }
    cred.put("userid", userid);
    cred.put("password", password);
    try {
      res = new ResourceRequest(type, url, method);
      if (res.isProtected()) {
        System.out.println("Resource " + type + ":" + url + "
```

```
protected");
     } else {
       System.out.println("Resource " + type + ":" + url + "
unprotected");
     }
   } catch (Throwable t) {
     t.printStackTrace();
     System.out.println("Failed to created new resource
request");
     return;
   }
   UserSession user = null;
   try {
     user = new UserSession(res, cred);
   } catch (Throwable t) {
     t.printStackTrace();
     System.out.println("Failed to create new user session");
     return;
   }
   try {
   if (user.getStatus() == UserSession.LOGGEDIN) {
     if (location != null) user.setLocation(location);
     System.out.println("user status is " + user.getStatus());

       if (parameters != null ? user.isAuthorized(res,
parameters) :
             user.isAuthorized(res)) {
         System.out.println("Permission GRANTED");
         System.out.println("User Session Token =" +
             user.getSessionToken());
         if (location != null) {
           System.out.println("Location = " +
user.getLocation());
         }
       } else {
         System.out.println("Permission DENIED");
         if (user.getError() == UserSession.ERR_NEED_MORE_DATA)
{
           int nParams =
res.getNumberOfAuthorizationParameters();
           System.out.print("Required Authorization Parameters
(" +
               nParams + ") :");
           Enumeration e =
res.getAuthorizationParameters().keys();
           while (e.hasMoreElements()) {
             String name = (String) e.nextElement();
             System.out.print(" " + name);
           }
           System.out.println();
         }
       }
  }
  else
  {
  System.out.println("user status is " + user.getStatus());
  }
   } catch (AccessException ae)
   {
   System.out.println("Failed to get user authorization");
```

```
    }
   String[] actionTypes = user.getActionTypes();
   for(int i =0; i < actionTypes.length; i++)
   {
   Hashtable actions = user.getActions(actionTypes[i]);
   Enumeration e = actions.keys();
   int item = 0;
   System.out.println("Printing Actions for type " +
 actionTypes[i]);
   while(e.hasMoreElements())
   {
   String name = (String)e.nextElement();
   System.out.println("Actions[" + item +"]: Name " + name + "
 value " +  actions.get(name));
   item++;
   }
}
   AuthenticationScheme auths;
   try
   {
   auths = new AuthenticationScheme(res);
       if (auths.isBasic())
       {
     System.out.println("Auth scheme is Basic");
       }
       else
       {
     System.out.println("Auth scheme is NOT Basic");
       }
   }
   catch (AccessException ase)
   {
   ase.printStackTrace();
   return;
   }
   try
   {
   ResourceRequest resNew = (ResourceRequest) res.clone();
   System.out.println("Clone resource Name: " +
 resNew.getResource());
   }
   catch (Exception e)
   {
   e.printStackTrace();
   }
   res = null;
   auths = null;
   ac.shutdown();
   }
  }
```

2.2.8 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_10G);
}
```

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

```
catch (AccessException ae) {
  System.out.println("
```

```
OAM Server SDK Initialize failed");
  ae.printStackTrace();
```

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

```
 return;
}
```

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

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

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

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

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

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

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

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

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

```
  return;
}
```

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

```
UserSession user = null;
```

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

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

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

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

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

```
 return;
}
```

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

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

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

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

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

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

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

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

Display also a serialized representation of the user session token.

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

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

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

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

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

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

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

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

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

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

```
while (e.hasMoreElements()) {
  String name = (String) e.nextElement();
```

```
  System.out.print(" " + name);
}
System.out.println();
}
```

Otherwise, simply proceed to the next statement.

```
    else
  }
}
```

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

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

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

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

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

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

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

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

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

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

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

```
  return;
}
```

Determine if the authorization scheme is basic.

```
try
{
```

```
if (auths.isBasic())
```

If it is, report the fact.

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

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();
  }
}
```

2.2.9 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[] encodecCert = cert.getEncoded();

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

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

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

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

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

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

The following import statements are associated with the snippet:

```
import java.security.cert.CertificateFactory;
     import java.security.cert.X509Certificate;
     import java.io.FileInputStream;
     import oracle.security.am.common.nap.util.Base64;
```

2.3 Messages, Exceptions, and Logging

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

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

The following topics are discussed in this section:

- Messages
- Exceptions
- Logging

2.3.1 Messages

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

2.3.2 Exceptions

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

- **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 11*g* 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.

2.3.3 Logging

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

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

For example:

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

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

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

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

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

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

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

# The following special tokens can be used in the pattern property
# which specifies the location and name of the log file.
#   / - standard path separator
#   %t - system temporary directory
```

```
#   %h - value of the user.home system property
#   %g - generation number for rotating logs
#   %u - unique number to avoid conflicts
# FileHandler writes to %h/demo0.log by default.
java.util.logging.FileHandler.pattern=%h/asdk%u.log


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

The following is a sample of the log output:

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

2.4 Building an Access Client Program

The following topics are discussed in this section:

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

2.4.1 Setting the Development Environment

The required environment is as follows:

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

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

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

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

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

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

  For a list of all jar files required in the CLASSPATH variable, see Section 2.1.2, "About Installing Access SDK".

2.4.2 Compiling a New Access Client Program

After the development environment is configured (see Section 2.4.1, "Setting the Development Environment"), you can compile your Access Client program using a command similar to the following:

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

Modify details such as CLASSPATH and Access Client program name as needed. For more information about the jar files to add to CLASSPATH, see Section 2.1.2, "About Installing Access SDK".

2.5 Configuring and Deploying Access Clients

After development, the Access Client must be deployed in a live Access Manager 11*g* environment before you can test and use it. 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 topics:

- Task Overview: Configuring and Deploying an Custom Access Client
- Configuration Requirements
- Generating the Required Configuration Files
- SSL Certificate and Key Files

2.5.1 Task Overview: Configuring and Deploying an Custom Access Client

The following overview outlines the tasks that must be performed by a user with Oracle Access Management administrator credentials. It is assumed that the Access Client program is already developed and compiled.

1. Retrieve the Access SDK jar file and copy this to the computer you will use to build the Access Client. For more information, see Section 2.1.2, "About 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 (standalone application), verify that the necessary JPS jar files are in the CLASSPATH. For more information, see Section 2.1.2, "About Installing Access SDK".
6. To deploy the Access Client, see "Registering Agents and Applications by Using the Console" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

2.5.2 Configuration Requirements

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 11.1.2 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 11*g* agent is registered with Access Manager. The cwallet.sso file contains the secret key that is used by 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.

- **jps-config.xml**

  This file is required by the libraries used to read the cwallet.sso file. It must reside in the Access SDK configuration directory. 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.

  A sample jps-config.xml file is included in the client install package zip file.

- **JKS Keystores for SSL**

  This file is required only if the transport security mode is Simple or Cert. Both the 10*g* OAM Server and 11*g* 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 11*g* agent has its own agent key, unlike the 10*g* agent that shares the same global key across all 10*g* agents. The 11*g* agent key is stored in cwallet.sso. This key is used to encrypt the 11*g* format 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. For more information, see *Oracle 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

  For more information, see Section 2.1.3.2, "Access Client Architecture" and Section 2.5.3, "Generating the Required Configuration Files".

- **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.

  For more information, see Section 2.5.3, "Generating the Required Configuration Files".

- **Log Configuration**

  Is required in order to generate a log file. For more information, see Section 2.3.3, "Logging".

2.5.3 Generating the Required Configuration Files

The ObAccessClient.xml configuration file can be obtained by registering an Access Client as either an 10*g* or 11*g* agent with the OAM 11*g* Server, using the administration console or a remote registration tool. When registering 11*g* agents the cwallet.sso file is

also created. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

The 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*.

2.5.4 SSL Certificate and Key Files

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

2.5.4.1 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/.

2.5.4.2 Cert Transport Security Mode

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

Follow these steps to prepare for Cert mode:

1. Import a CA certificate of the certifying authority using the certificate and key pair issued for Access Client and OAM Server. Follow the steps in Section 2.5.4.2.1, "Importing the CA Certificate". Instead of cacert.pem or cacert.der, substitute the CA certificate file of the issuing authority.
2. If 10*g* JNI ASDK install is available, it provides a way to generate certificate and key file for the Access Client. These certificates will be in PEM format.

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

   To import this certificate, key pair in the oamclient-keystore.jks in PEM format, follow instructions in Section 2.5.4.2.2, "Setting Up The Keystore".

2.5.4.2.1 Importing the CA Certificate

This step is not required when using the 11*g* Java Access SDK.

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

- **10g Access SDK**: The CA certificate (cacert.pem) is located in ASDK_INSTALL_DIR/oblix/tools/openssl/simpleCA.

- **OAM 11g 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.

2.5.4.2.2 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 in Simple mode so the Access Client can communicate with OAM Server.

- **10g Access SDK**: provides for generating a certificate and key file for the Access Client. These certificates are in PEM format.
- **11g 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.

2.6 Compatibility: 11*g* versus 10*g* Access SDK and APIs

The following topics are discussed in this section:

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

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

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

10*g* and 11*g* Access Manager API functionality has been organized into seven basic classes. Table 2–3 lists the corresponding class names for the Java language platform.

Table 2–3 Comparison: 11g versus 10g Access API Classes

Purpose of the Class	11g Java Class	10g Java Class
Creates and manipulates structures that handle user authentication	AuthenticationScheme class from oracle.security.am.asdk	ObAuthenticationScheme implements ObAuthenticationSchemeInterface
Creates and manipulates structures that handle user requests for resources	ResourceRequest class from oracle.security.am.asdk	ObResourceRequest implements ObResourceRequestInterface
Creates and manipulates structures that handle user sessions, which begin when the user authenticates and end when the user logs off or the session times out.	UserSession class from oracle.security.am.asdk	ObUserSession implements ObUserSessionInterface
Creates and manipulates structure that handles a unified session for the user, which begins when user is authenticated for the first time and ends when the user logs off or the session times out.	PseudoUserSession class from oracle.security.am.asdk	ObPseudoUserSession
Retrieves and modifies Access Client configuration information	AccessClient class from oracle.security.am.asdk	ObConfig
Handles errors thrown by the Access Manager API	AccessException, OperationNotPermittedException from oracle.security.am.asdk	ObAccessException
Notifies the change in configuration to the calling application.	ConfigUpdateCallback class from oracle.security.am.asdk	

2.6.1 Compatibility of the 11*g* Access SDK

The 11*g* Access SDK implements the same functionality that is supported by the 10*g* JNI ASDK. This functionality is implemented so that you can use it to develop custom Access Clients that work seamlessly with both the 10*g* and 11*g* OAM Server.

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

The new functionality in the 11*g* Access SDK (`oracle.security.am.asdk`) is as follows:

- Enumerating sessions for the given user
- Terminating the given session
- Setting attributes in the given user session
- Retrieving attributes set in the given session
- Validating user credentials without establishing a session
- Validating user credentials without establishing a session and performing authorization in the same request

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

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

2.6.2 Compatibility of 10*g* JNI ASDK and 11*g* Access SDK

There is a one-to-one mapping between the 10*g* JNI ASDK and the 11*g* Access SDK version of the `com.oblix.access` package.

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

The following classes have been added to the 11*g* Access SDK `com.oblix.access` package:

- **ObPseudoUserSession**: This class provides the following functionality that can be used only with 11*g* OAM Server:
  - Validating user credentials without establishing a session.
  - Validating user credentials without establishing a session and performing authorization in the same request.
- **ObAccessRuntimeException**: This class indicates a runtime error while performing operations that use `ObAuthenticationScheme` and `ObResourceRequest` classes.

2.6.3 Deprecated: 10*g* JNI ASDK

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

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

2.7 Migrating Earlier Applications or Converting Your Code

This section describes the migration processes to follow if you want to use the 11*g* Access SDK. Migrating to the Access SDK can be necessary for the following reasons:

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

This section contains the following topics:

2.7.1 Modifying Your Development and Runtime Environment

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

2.7.2 Migrating Your Application

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

Note: For information about the similarities and differences between the `com.oblix.access` APIs in 10*g* JNI ASDK and in 11*g* Access SDK, see Section 2.6.2, "Compatibility of 10g JNI ASDK and 11g Access SDK".

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

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

A new runtime exception, `ObAccessRuntimeException`, was introduced in the `com.oblix.access` package. This exception is thrown when performing operations of `AuthenticationScheme` and `ResourceRequest` classes.

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

2.7.2.1 Configuration Specific to Migration

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

- The configuration items listed in Section 2.5.2, "Configuration Requirements" are referenced from the 10*g* ASDK installation directory (ASDK_INSTALL_DIR).
- ObAccessClient.xml is read from ASDK_INSTALL_DIR/access/oblix/lib.

- password.xml is read from ASDK_INSTALL_DIR/access/oblix/config if the transport security mode is Simple or Cert.

Simple Mode

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

Perform the following steps:

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

   The aaa_cert.pem and aaa_key.pem files are located in ASDK_INSTALL_DIR/access/oblix/config/simple.

2. Located the self-signed CA certificate used for issuing Simple mode certificates in ASDK_INSTALL_DIR/access/oblix/tools/openssl/simpleCA.

3. Import the self-signed CA certificate into oamclient-truststore.jks.

4. Import the certificate and key files into the JKS store by following the steps in Section 2.5.4, "SSL Certificate and Key Files".

5. Copy the JKS stores to ASDK_INSTALL_DIR/access/oblix/config/simple.

Cert Mode

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

Perform the following steps:

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

   The aaa_cert.pem, aaa_key.pemand aa_chain.pem files are located in ASDK_INSTALL_DIR/access/oblix/config.

2. Import the certificate and key files into the JKS store by following the steps in Section 2.5.4, "SSL Certificate and Key Files".

3. Copy the JKS stores to ASDK_INSTALL_DIR/access/oblix/config/simple.

Configuration File Location

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

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

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

Environment

To set your environment, follow the instructions in Section 2.4.1, "Setting the Development Environment". The 10*g* JNI ASDK is named jobaccess.jar. If jobaccess.jar is in your CLASSPATH, it must be removed.

2.7.3 Converting Your Code

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

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

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

2.7.3.1 Understanding Differences Between 10*g* JNI ASDK and 11*g* Access SDK

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

Table 2–4 Package Differences: com.oblix.access and oracle.security.am.asdk

JNI ASDK com.oblix.access Package	Access SDK oracle.security.am.asdk Package
Interface Summary: ■ ObAuthenticationSchemeInterface ■ ObResourceRequestInterface ■ ObUserSessionInterface	**Interface Summary:** None
Class Summary: ■ ObAuthenticationScheme ■ ObConfig ■ ObDiagnostic ■ ObResourceRequest ■ ObUserSession	**Class Summary:** ■ AuthenticationScheme ■ AccessClient ■ Supported through AccessClient ■ ResourceRequest ■ UserSession ■ PseudoUserSession ■ BaseUserSession
Exception Summary: ObAccessException	**Exception Summary:** ■ AccessException ■ OperationNotPermittedException
Enumeration Summary: None	**Enumeration Summary:** AccessClient.CompatibilityMode.OAM_10G AccessClient.CompatibilityMode.OAM_11G

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

2.7.3.2 Converting Code

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

2.7.3.2.1 Initializing and Uninitializing Access SDK

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

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

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

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

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

  If no `AccessClient` instance is specified when instantiating these classes, then the default instance is used.

  You can pass either `AccessClient.CompatibilityMode.OAM_10G` or `AccessClient.CompatibilityMode.OAM_11G` when initializing `AccessClient` objects. If not specified, then default `OAM_11G` would be used, in which case make sure the 11*g* agent is registered and the necessary 11*g* agent configuration files are set up properly.

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

  You can pass either `AccessClient.CompatibilityMode.OAM_10G` or `AccessClient.CompatibilityMode.OAM_11G` when initializing `AccessClient` objects. If not specified, then default `OAM_11G` would be used, in which case make sure the 11*g* agent is registered and the necessary 11*g* agent configuration files are set up properly.

  If you use the `createInstance` method, you must use the `AccessClient` class instance obtained using this method when instantiating the `AuthenticationScheme`, `ResourceRequest`, or `UserSession` classes. Otherwise, if no `AccessClient` instance is specified when instantiating these classes, then the default instance is used.

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

- **For 10g JNI ASDK**

```
Public static void main (String args[]) {
  try {
```

```
   ObConfig.Initialize (); // Configuration is read from the location pointed
by OBACCESS_INSTALL_DIR
                                          // environment variable
```

OR

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

- **For 11g Access SDK**

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

OR

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

2.7.3.2.2 Performing Access Operations

As shown in Table 2–4, there is a one-to-one mapping between the classes that are used to perform access operations. The classes in `oracle.security.am.asdk` are `AuthenticationScheme`, `ResourceRequest`, and `UserSession`.

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

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

- **For 10g JNI ASDK**

```
Public static void main (String args[]) {
  try {
   ObConfig.Initialize (); // Configuration is read from the location pointed
by OBACCESS_INSTALL_DIR
                                          // environment variable
   ObResourceRequest rrq = new ObResourceRequest(ms_protocol, ms_resource,ms_
method);
   if (rrq.isProtected()) {
     System.out.println("Resource is protected.");
     ObAuthenticationScheme authnScheme = new ObAuthenticationScheme(rrq);
```

```
     if (authnScheme.isForm()) {
       System.out.println("Form Authentication Scheme.");
       Hashtable creds = new Hashtable();
       creds.put("userid", ms_login);
       creds.put("password", ms_passwd);
       ObUserSession session = new ObUserSession(rrq, creds);
       if (session.getStatus() == ObUserSession.LOGGEDIN) {
         if (session.isAuthorized(rrq)) {
           System.out.println("User is logged in and authorized for the
           request at level " + session.getLevel());
         } else {
           System.out.println("User is logged in but NOT authorized");
         }
      } else {
        System.out.println("User is NOT logged in");
      }
    } else {
      System.out.println("non-Form Authentication Scheme.");
  }
} else {
  System.out.println("Resource is NOT protected.");
}
}catch (ObAccessException oe) {
  System.out.println("Access Exception: " + oe.getMessage());
}
ObConfig.shutdown();
}//main ends here
```

- **For 11g Access SDK**

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

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

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

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

        UserSession session = new UserSession(rrq, creds);
        if (session.getStatus() == UserSession.LOGGEDIN) {
          if (session.isAuthorized(rrq)) {
            System.out.println("User is logged in " +
                "and authorized for the request " +"at level " +
```

```
session.getLevel());
          } else {
            System.out.println("User is logged in but NOT authorized");
          }
        } else {
          System.out.println("User is NOT logged in");
        }
      }
    }catch (AccessException oe) {
      System.out.println("Access Exception: " + oe.getMessage());
    }
    ac.shutdown();
} //main ends here
```

2.8 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 Access Clients
- Identifying and Resolving Access Client Problems
- Resolving Environment Problems
- Tuning for High Load Environment

2.8.1 Avoiding Problems with Access Clients

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

- Make sure that your Access Client attempts to connect to the correct OAM Server.
- Make sure the configuration information on your OAM Server matches the configuration information on your Access Client. You can check the Access Client configuration information on your OAM Server, using the administration console. For details, see "Registering Agents and Applications" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.
- To ensure clean connect and disconnect from the OAM Server, use the `initialize` and `shutdown` methods in the `AccessClient` class.
- The environment variable, OBACCESS_INSTALL_DIR, *must* be set on your Windows or UNIX-like host computer so that you can compile and link your Access Client. In general, you also want the variable to be set whenever your Access Client is running.
- Use the exception handling features (try, throw, and catch) of the language used to write your custom Access Client code to trap and report problems during development.

2.8.1.1 Thread Safe Code

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

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

- Use a thread safe function instead of its single thread counterpart. For instance, use localtime_r instead of localtime.
- Specify the appropriate build environment and compiler flags to support multithreading. For instance, use -D_REENTRANT. Also, use -mt for UNIX-like platforms and /MD for Windows platforms.
- Take care to use in thread-safe fashion shared local variables such as FILE pointers.

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 11*g* Access Tester enables you to check which policy applies to a particular resource. For details about using the Access Tester and protecting resources with application domains, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

2.8.3 Resolving Environment Problems

This section provides information about resolving environment conflicts that can develop when using the 11*g* Java Access SDK.

2.8.3.1 Java EE Containers

Use this procedure to resolve Java class version conflicts when a web application using the 11*g* 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 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-application.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 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 11*g* 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 that test the credentials provided by a user when he or she tries to access a resource. The plug-ins can be taken from a standard set provided with OAM Server installation, or the custom plug-ins created by your own Java developers. This chapter provides the following sections:

- Section 3.1, "Introduction to Authentication Plug-ins"
- Section 3.2, "Introduction to Multi-Step Authentication Framework"
- Section 3.3, "Introduction to Plug-in Interfaces"
- Section 3.4, "Sample Code: Custom Database User Authentication Plug-in"
- Section 3.5, "Developing an Authentication Plug-in"

> **See Also:** For information about deploying and managing authentication plug-ins using the Oracle Access Management administration console, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

3.1 Introduction to Authentication Plug-ins

The 11*g* release provides authentication modules for immediate use out-of-the-box, as well as the following:

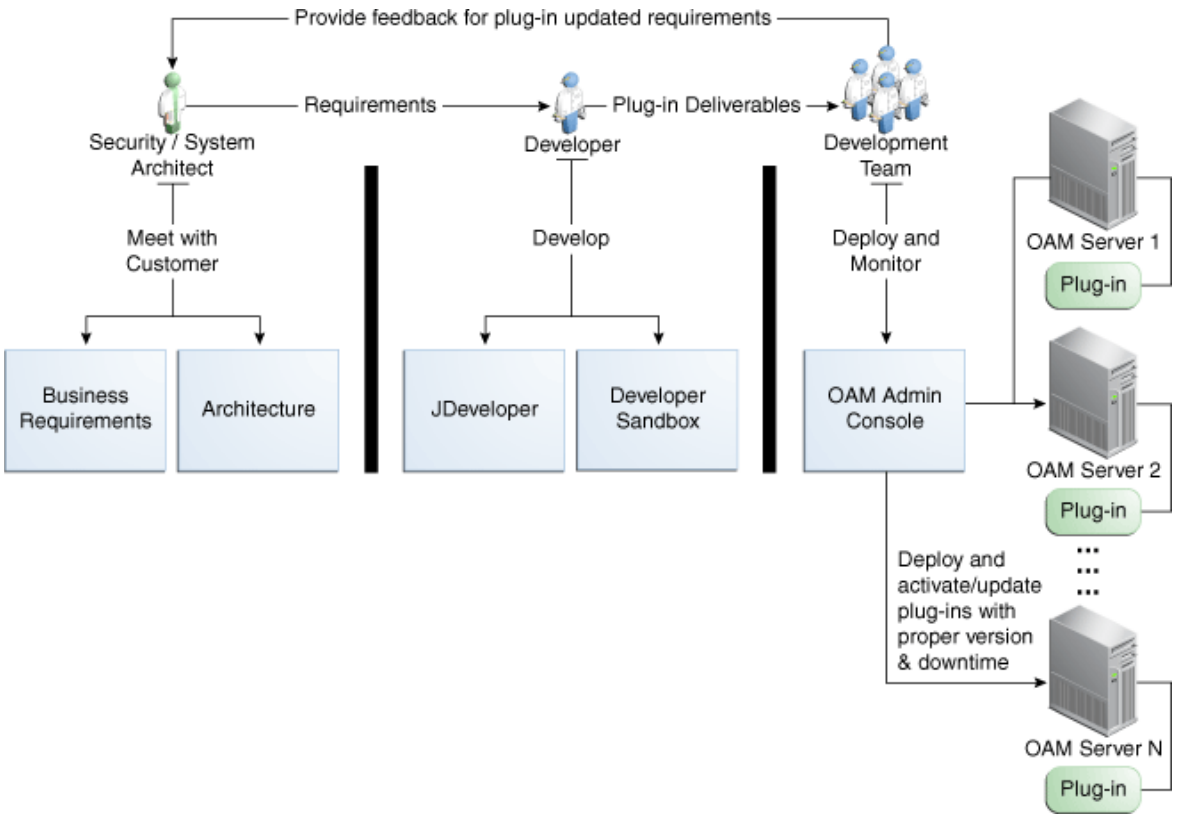
- Authentication plug-in interfaces and SDK tooling to build customized authentication modules (plug-ins) to bridge the out-of-the-box features with individual requirements. The new interfaces and SDK tooling:
  - Provide backward compatibility to support custom Oracle Access Manager 10*g* plug-ins.
  - 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:** Section 3.3.1, "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.

Task overview: Deploying a Custom Plug-in

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

   The security architect knows how Access Manager 11*g* 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 *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*

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 *Oracle Fusion Middleware Administrator's Guide for 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 *Oracle Fusion Middleware Administrator's Guide for 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 for:

- 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.

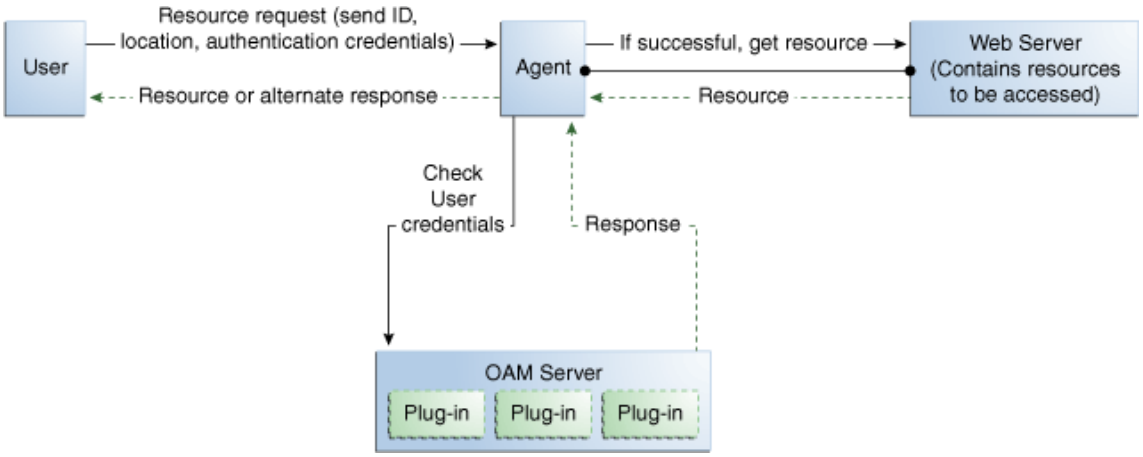
- Custom Responses

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

- Other types of plug-ins are also supported.

Figure 3–2 illustrates the authentication flow when a user requests a protected resource. Remember that authentication is a process and not a protocol. The green 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).

Example: Decision Engine versus Hard-Coded Authentication

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

The differences between the two approaches are simple but important.

Process overview: Decision Engine Approach

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.

Process overview: Hard-Coded Approach

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

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

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

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.

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
- About Information Collected

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, OAM10G, TAP, and NONE. The challenge mechanism controls the way in which the required credentials are collected. Currently, this is tied to the authentication scheme.

See Also: "Configuring 11*g* Webgate for Detached Credential Collection" in *Oracle Fusion Middleware Administrator's Guide for 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 About Information Collected

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.

The 11*g* release provides a generic plug-in interface and a more specific authentication interface as described in the following topics:

- GenericPluginService
- AuthnPluginService

3.3.1.1 GenericPluginService

oracle.security.am.plugin

The public interface, `oracle.security.am.plugin`, is a generic plug-in interface that provides methods to get plug-in name, plug-in implementation class name, plug-in version, plug-in execution status, plug-in monitoring data, plug-in configuration data, 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 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, "Plug-in Package Hierarchy"
- Figure 3–4, "Plug-in Class Hierarchy"
- Figure 3–5, "Plug-in Interface Hierarchy"
- Figure 3–6, "Plug-in Annotation Type Hierarchy"
- Figure 3–7, "Plug-in Enum Hierarchy"

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

Figure 3–3 Plug-in Package Hierarchy

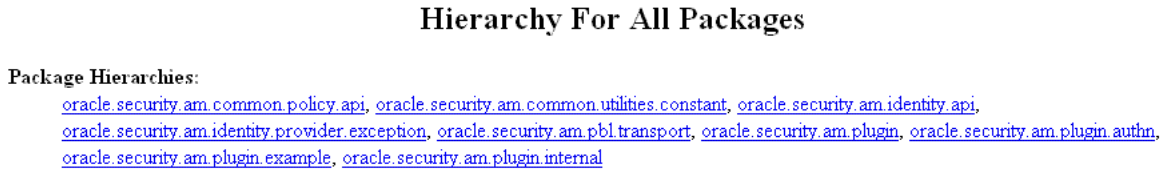


Figure 3–4 Plug-in Class Hierarchy

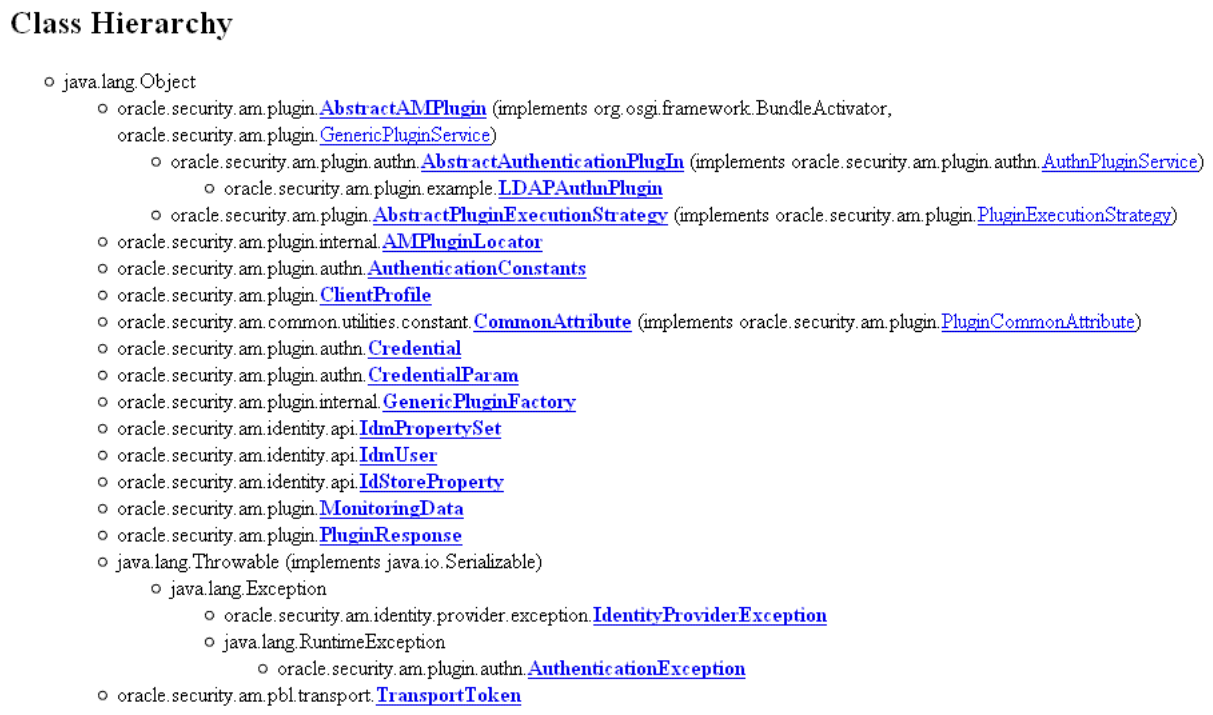


Figure 3–5 Plug-in Interface Hierarchy

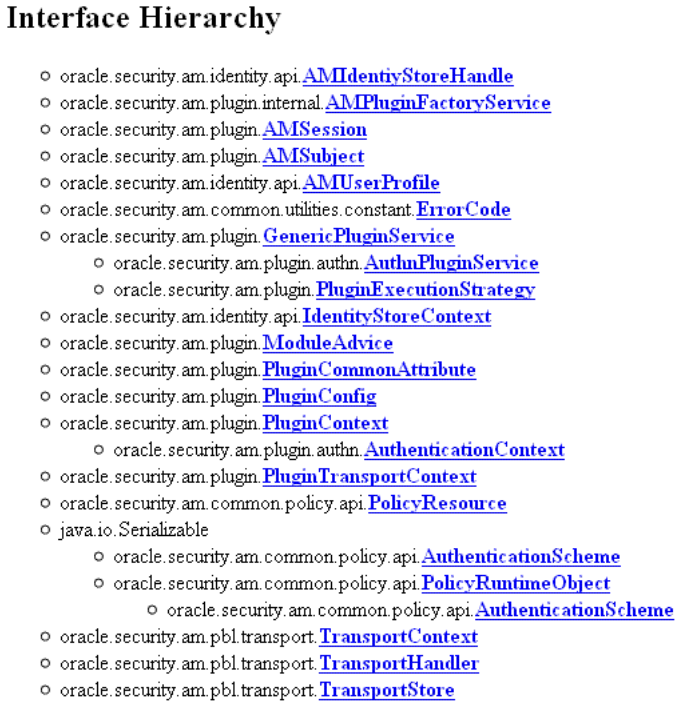
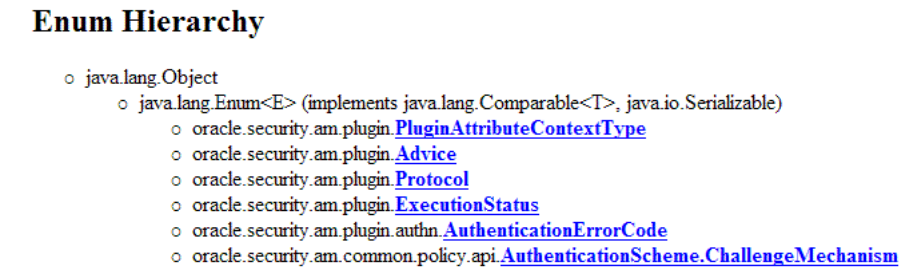


Figure 3–6 Plug-in Annotation Type Hierarchy

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

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

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

```
public class DBUserAuthentication extends AbstractAuthenticationPlugIn {

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

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

Map<String, Object> module = null;

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

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

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

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

     }
     return ExecutionStatus.SUCCESS;
  }

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

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

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

Continued ..

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

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

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

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

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

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

Continued...

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

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

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

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

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

3.4.2 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@mycompany.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@mycompany.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 Plug-in JAR File Structure

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

- <*plugin*>.xml
- <*plugin*>.class (per the package structure, as shown in Section 3.3, "Introduction to Plug-in Interfaces")
- META-INF (MANIFEST.MF)

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.

This section guides as you develop an authentication plug-in for use with Access Manager 11*g* authentication schemes. 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 Section 3.3.1, "About the Plug-in Interfaces")
- Implement `initialize` method
- Implement `process` method

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

Table 3–3 Required Plug-in Methods

Required Method	Description
initialize	Gives a handle to the `PluginConfig` object. The `PluginConfig` object can be exercised to get plug-in specific system configuration data that is entered when the plug-in is uploaded. This data is required for the plug-in's own functionality.

Table 3–3 (Cont.) Required Plug-in Methods

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

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

3.5.2 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 Section 3.1.2, "About Planning, the Authentication Model, and Plug-ins".

Prerequisites

Introduction to Authentication Plug-ins

Sample Code: Custom Database User Authentication Plug-in

Task overview: Developers write a custom authentication plug-in

1. Extend `AbstractAuthenticationPlugIn` class and implement the following methods (see also Section 3.5.1, "About Writing a Custom Authentication Plug-in"):
   - Implement `initialize` method
   - Implement `process` method
2. Develop plug-in code using appropriate Access Manager 11*g* interfaces and packages. See:
   - Section 3.1, "Introduction to Authentication Plug-ins"
   - Section 3.4, "Sample Code: Custom Database User Authentication Plug-in"
3. Prepare Metadata for the Custom Plug-in. See:

- Section 3.4.2, "Sample Plug-in Configuration Metadata Requirements"

4. Prepare the Plug-in Jar file and manifest and turn these over to your deployment team. See:
   - Section 3.4.3, "Sample Manifest File for the Plug-in"
   - Section 3.4.4, "Plug-in JAR File Structure"
5. Proceed to:
   - Section 3.5.4, "JAR Files Required for Compiling a Custom Authentication Plug-in"
   - For information about deploying and managing custom authentication plug-ins, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

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

The OAM Server provides default login, logout, and error pages. You can also create custom login and error pages tailored for your company look and feel to use with Access Manager. This chapter explains how to develop custom pages and how to deploy them in your environment. This chapter provides the following sections:

4.1 Introduction to Custom Pages

This section contains the following topics:

4.1.1 About Developing Custom Pages

Access Manager provides an extensible framework for creating customized user login experience. You can create custom interactive pages for authentication during login, logout, and error conditioning processing. In its simplest form, Access Manager provides a set of static HTML pages that are displayed to the user. However, the user interface is generally dynamic, which requires that it be implemented as a script or an application that can perform the required logic.

Access Manager provides a set of default dynamic pages for user interaction. These default pages may be customized for your company look and feel, or replaced entirely with custom pages. For example, you can design, implement, and deploy a custom dynamic page that displays a different version of the login form depending on whether the user is accessing via a mobile browser or a desktop browser.

A custom page can be developed to use in combination with existing Access Manager authentication modules or in combination with a custom authentication plug-in. This chapter provides information about developing custom pages for login, logout, and errors. For more information about developing a custom authentication plug-in, see Chapter 3, "Developing Custom Authentication Plug-ins".

The following two OAM Server credential collection components can be alternatively enabled to serve as the communication endpoint and facilitate interaction with the customized user interface:

- The Embedded Credential Collector (ECC) can be used with no additional installation and setup steps.
- The Detached Credential Collector (DCC) is recommended for greater scalability and security isolation in production deployments.

For more information about the OAM Server credential collectors, see "Configuring 11*g* Webgate for Dynamic Credential Collection" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

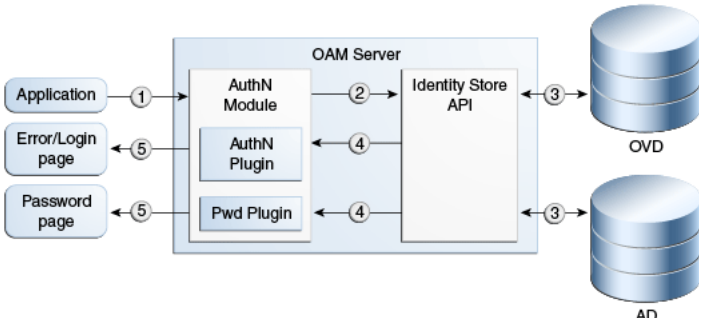
Regardless which credential collection component is enabled for communicating with users, the design and implementation of custom pages in your environment is almost identical. For information about development differences between the two credential collection options, see Section 4.4, "Developing Using the Detached Credential Collector".

4.1.2 About Authentication and 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 is based on the results of credential validation.

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

Figure 4–1 Authentication Request Flow



Process Overview: Authentication Request Flow

1. A user requests a protected resource. An authentication flow is triggered and the login page is displayed. The required credentials are submitted to the authentication engine.
2. The resource is protected by a specific authentication scheme, which uses the authentication (AuthN) plug-in. The AuthN plug-in uses the Identity Store APIs to authenticate the credentials. The AuthN plug-in can also call a third-party API (not shown).
3. The Identity Store API establishes a connection to the backend store to complete authentication. AD and OVD are shown as example identity stores.
4. The results, including any authentication error codes, are returned by the Identity Store layer and sent to the plug-in and the authentication engine layer. The authentication layer maps the error codes from the backend to the corresponding

OAM Server error codes. For more information about the standard error codes, see Table 4–2.

The results include native error codes acquired from the identity stores, and returning those error codes to the login page as unmapped secondary error codes (`p_sec_error_msg`). For more information about secondary error codes, see Section 4.3.6, "Secondary Error Message Propagation".

5. The 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`.
6. 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.

4.2 Developing Custom Login Pages

This section contains the following topics:

- Creating a Form-Based Login Page
- Page Redirection Process

4.2.1 Creating a Form-Based Login Page

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 using your preferred technology to process the redirect from the user and render the HTML. A custom login page can be written as a JSP page, or using ASP.net, Perl, PHP, and so on. One advantage is you can customize the look and feel of the page so it matches company standards. Another advantage is it enables pre-processing of the user's submission (POST) before their credentials are sent to the OAM Server, if this is desired.

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 login page or application you have written. For more information, see Section 4.2.2, "Page Redirection Process".

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 Section 4.2.1.1, "Returning OAM_REQ Token".
- The page must retrieve the end point. See Section 4.2.1.2, "Returning the End Point".

4.2.1.1 Returning OAM_REQ Token

OAM_REQ is a transient cookie that is set or cleared by the OAM Server if the authentication request context cookie is enabled. This cookie is protected with keys known to the OAM Server only. For more information, see "Introduction to SSO Cookies" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

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 to the credential collector page to collect credentials. The OAM Server also send 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%>">
<%
}
%>
```

4.2.1.2 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".
```

4.2.2 Page Redirection Process

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%2Fmycompany.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.

4.3 Developing Custom Error Pages

This section discusses the following topics:

- Process Overview: Creating a Custom Error Page
- Standard Error Codes
- Default Page Locations
- Security Level Configuration
- Password Policy Validation Error Codes
- Secondary Error Message Propagation
- Retrieving Error Codes
- Error Data Sources Summary

See Also: Section 3.5.3, "Error Codes in an Authentication Plug-In"

4.3.1 Process Overview: Creating a Custom Error Page

The custom error page is packaged as part of the custom login application. Under authentication policy, set the failure redirect URL to be the absolute URL to the page.

An out-of-the-box custom UI Web application archive (WAR) file is provided that can be used as a starting point to develop customized login and password pages. This WAR file is located in *MW_HOME*/OAM1/oam/server/tools/custompages/oamcustompages.war.

To create a custom error page, follow these steps:

1. Create the HTML page required to display the error message. For more information, see Section 4.2, "Developing Custom Login Pages".
2. Consider how error code query parameters `p_error_code` and `p_sec_error_msg` will map to the custom error codes in your environment. For more information, see Section 4.3.7, "Retrieving Error Codes".

4.3.2 Standard Error Codes

Access Manager provides standard error codes that indicate the reason for failure. Common reasons for failure include providing an invalid username and password combination, a user account is locked or disabled, or an internal processing error occurred. The reason for the authentication error is received from the backed identity store, which is then mapped to a specific error code maintained in the OAM Server. This error code is then propagated to the login or error page. For more information about the authorization request flow, see Section 4.1, "Introduction to Custom Pages".

Table 4–1 summarizes the standard error information that 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.

4.3.3 Default Page Locations

In the case where the Embedded Credential Collector (ECC) is used, the default pages are accessed as follows:

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

In the case where the Detached Credential Collector (DCC) is used, the default pages are accessed as follows:

- 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.

4.3.4 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.

4.3.5 Password Policy Validation Error Codes

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 4–4 lists the available error codes, message key, and the corresponding message that can be returned during password validation.

Table 4–4 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}

Table 4–4 (Cont.) Password Validation Error Codes

Message Key in URL	Message Key for Resource Bundle	Message Text
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
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

4.3.6 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 Section 4.3.4, "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:

Entity	Description
Error Code	OAM-1
Primary Message (retrieved from the code)	An incorrect Username or Password was specified
Secondary Error Code	LDAP:error code 49-Invalid Credentials

4.3.7 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, such as OAM-1, that is returned by the OAM Server. 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 be meaningful messages that are 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.

4.3.7.1 Code Samples

A local resource bundle file must be created with and 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>
```

4.3.7.2 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

4.3.7.3 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
- `PassswordRuleDescription.eachDesc.getDisplayValue()` returns fully translated message

4.3.8 Error Data Sources Summary

Table 4–5 summarizes the error data sources available to an authentication plug-in.

Table 4–5 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: `ruleDesc`
Plugin client responses	HTTP Request parameter: PLUGIN_CLIENT_RESPONSE
Plugin error responses	HTTP Request parameter: PLUGIN_ERROR_RESPONSE

4.4 Developing Using the Detached Credential Collector

This section contains the following topics:

4.4.1 Detached Credential Collector Considerations

The primary differences when developing a custom page that uses the Detached Credential Collector (DCC) are as follows:

- 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 login pages for supported authentication mechanisms, the default error page and default 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, for en-us the exact location is: *WebgateOH*/webgate/ohs/lang/en-us/WebGate.xml.

  The default logout page is located in *WebgateOH*/webgate/ohs/oamsso-bin/logout.pl.
- DCC by itself does not constrain the choice of the language to develop the custom pages in. The pages can be either deployed on the Oracle HTTP Server hosting the DCC, or in the case of JSP or Servlets, they need to be deployed on a web container that is fronted by it.
- HTML forms used to post collected data to the DCC can do it by using a configurable URL. The `action` challenge parameter in the authentication scheme specifies the URL where the credentials are expected.
- `requestid` query parameter handling is not required.

4.4.2 Process Overview: Creating a Form-Based Login Page Using DCC

1. Create an HTML form where the user's credentials, such as user name and password, can be submitted. For more information, see Section 4.2.1, "Creating a Form-Based Login Page".
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. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.
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.

4.5 Deploying the Custom Login Page

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`.

For more information about authentication schemes and managing them, see "About the Authentication Schemes Page" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

4.6 Programmatic Authentication

This section contains the following topics:

- Using mod_osso Agent
- Using Unsolicited Post

4.6.1 Using mod_osso Agent

Programmatic authentication using HTTP client APIs is supported for both OSSO 10*g* and 11*g* OAM Server.

4.6.1.1 OSSO 10*g*

An OSSO 10*g* programmatic client typically looks for URL redirects to identify the authentication flow. The default authentication schemes are configured to use embedded login pages. OAM Server will forward to the login page instead of using redirection. For OSSO 10*g* style programmatic clients to work, the credential collector must be configured in external mode.

In cases of upgrade from 10*g* release, configure the authentication scheme `SSOCoexistMigrateScheme` to use the new custom login page. In cases of new 11*g* release installation, edit the scheme used for authenticating a resource, namely `LDAPScheme`.

Set the challenge URL to point to a fully qualified custom URL. For example, `http://host:port/sample-web/login.jsp`. Also set the context type to `external`. For more information, see "Managing Authentication Schemes" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

4.6.1.2 11*g* OAM Server

In 11*g* 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>
```

4.6.1.3 Process Overview: Developing Programmatic Clients

Use the following process when developing both OSSO 10*g* and Access Manager 11*g* programmatic clients.

1. The application invokes a protected resource via HTTP channel using the client API.
2. The `mod_osso` partner protecting the accessed resource generates the `site2pstoretoken` and redirects to OAM Server for authentication.
3. 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.
4. Programmatic clients are expected to submit credentials to the programmatic endpoint.
5. Clients must submit the following information to `p_submit_url`: `ssousername`, `password`, `s2pstoretoken` (obtained in Step 3).
6. OAM Server validates credentials, and if valid, redirects the request to the initial protected application.
7. In case of credential validation error, `p_error_code` is returned.

4.6.2 Using Unsolicited Post

In 11*g* release, 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.mycompany.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.

4.7 Setting Custom OSSO Cookies After Authentication

For `mod_osso` agents, 11*g* release supports setting custom cookies. For 10*g* and 11*g* Webgates, this is achieved through an authentication and authorization response type cookie.

To configure OSSO custom cookies:

1. Login to Oracle Access Management administration console.
2. Navigate to the application domain for the OSSO agent.
3. Select the protected resource policy for authentication.
4. Click **Responses** tab.
5. Add the cookie responses. The cookies for your deployment should be created with a Name and Value that follow Access Manager 11*g* authentication response format. For example:
   - **Name**: ORASSO_AUTH_HINT, **Type**: Cookie, **Value**: v1.0~${session.expiration}
   - **Name**: ORASSO_UCM_COOKIE1, **Type**: Cookie, **Value**: v2.0~${user.attr.displayname}~${user.attr.given}
   - **Name**: ORASSO_UCM_COOKIE2, **Type**: Cookie, **Value**: v3.0~${user.attr.uid}~${user.attr.mail}
6. Save the changes.
7. Access the `mod_osso` protected application to verify that the cookies are being created after authentication.

The OSSO cookies are set by default on *.example.com* domain and the cookies are set to expire after one year. The settings can be changed using the WLST commands:

- updateOSSOResponseCookieConfig
- deleteOSSOResponseCookieConfig

For example, using updateOSSOResponseCookieConfig:

```
help('updateOSSOResponseCookieConfig')

Description:

Updates OSSO Proxy response cookie settings in system configuration.
Syntax:


 updateOSSOResponseCookieConfig(cookieName = "<cookieName>",cookieMaxAge =
"<cookie age in minutes>", isSecureCookie = "true | false",cookieDomain="<domain
of the cookie>", domainHome = "<wls_domain_home_path>")
 cookieName = Name of the cookie for which settings are updated. This is optional
parameter. If parameter is not specified global setting is updated.
 cookieMaxAge = Max age of cookie in minutes. A negative value will set a session
cookie.
 isSecureCookie = Boolean flag specifies if cookie should be secure which wouldbe
sent only in SSL channel.
 cookieDomain = The domain of the cookie.
 domainHome = location of domain home <mandatory for offline commands,not required
for online>

Example:


 updateOSSOResponseCookieConfig(cookieName = "ORASSO_AUTH_HINT",cookieMaxAge =
"525600", isSecureCookie = "false", cookieDomain=".example.com",domainHome =
"<wls_domain_home_path>")
```

For example, using deleteOSSOResponseCookieConfig:

```
help('deleteOSSOResponseCookieConfig')

Description:

Deletes OSSO Proxy response cookie settings in system configuration.
Syntax:


 deleteOSSOResponseCookieConfig(cookieName = "<cookieName>",domainHome = "<wls_
domain_home_path>")
 cookieName = Name of the cookie for which settings are updated. This is
mandatoryparameter. The global cookie setting cannot be deleted.
 domainHome = location of domain home <mandatory for offline commands,not required
for online>

Example:


 deleteOSSOResponseCookieConfig(cookieName = "ORASSO_AUTH_HINT",domainHome =
"<wls_domain_home_path>")
```

In the update command, if cookie name is not specified, global settings for all the cookies are updated. If cookie name is specified, the parameters are overridden for the specific cookie. For example:

```
updateOSSOResponseCookieConfig(cookieMaxAge = "525600", isSecureCookie = "false",
```

```
cookieDomain=".mycompany.com")

updateOSSOResponseCookieConfig(cookieName="ORASSO_AUTH_HINT", cookieMaxAge = "-1",
isSecureCookie = "false", cookieDomain=".mycompany.com")

updateOSSOResponseCookieConfig(cookieName="ORASSO_UCM_COOKIE2", isSecureCookie =
"true", cookieDomain=".us.mycompany.com")

deleteOSSOResponseCookieConfig(cookieName = "ORASSO_UCM_COOKIE2")
```

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. This chapter provides the following sections:

5.1 Introduction to Policy Administration API

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

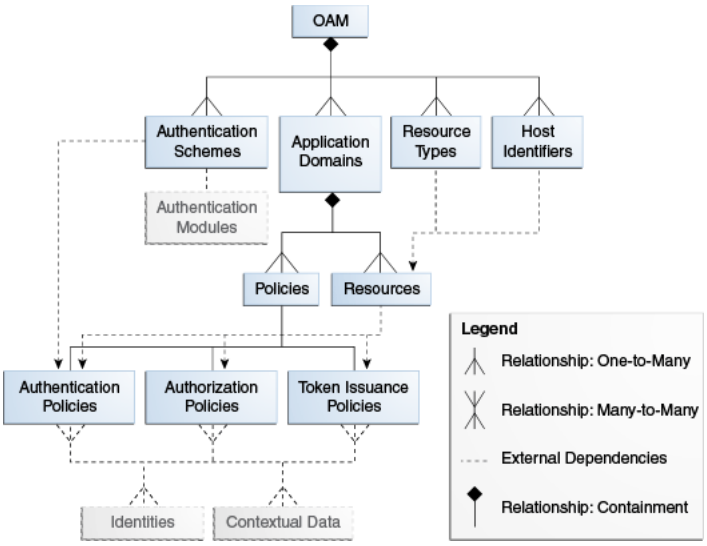


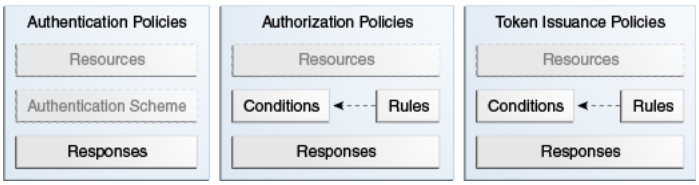
Table 5–1 provides details about the policy objects that can be managed using the RESTful interfaces. Each policy object is represented as an HTTP resource that is accessible through an HTTP uniform resource locator (URL).

Table 5–1 Policy Objects

Object Name	Description
Application Domain	The top-level construct of the 11*g* 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.
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

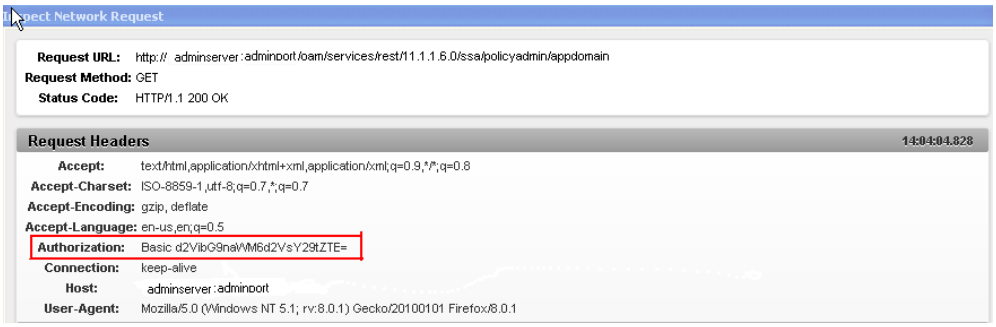


You can access the OAM Server RESTful interfaces through client applications such as:

- Web browsers
- cURL
- GNU Wget

5.1.2 Security Model

The Policy Administration REST API is protected by administrative roles. The REST services are protected by the container security that enforces the required 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
The host where the OAM Server is running

port
The HTTP or HTTPS port

path
The relative path that identifies a particular resource, and *path* is constructed as follows: */version/component/service/*

where:

- *version* - is the Access Manager product version, such as 11.1.2.0.0
- *component* - 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.mycompany.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 objects 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
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

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

Error Condition	HTTP Return Code	Content
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

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.
	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.
	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.
	DELETE	An Application Domain object is deleted by this method. The Application Domain matching NAME or ID query parameter is deleted.
`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.
	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.

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

Resource	Method	Description
	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.
	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.
	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.
	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.
	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.

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

Resource	Method	Description
`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.
	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.
	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.
	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.
	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.

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

Resource	Method	Description
	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.
	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.
	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.
	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.
	DELETE	A Resource Type object is deleted by this method. The Resource Type matching the NAME or ID query parameter is deleted.
`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.
	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.

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

Resource	Method	Description
	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.
	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 Examples

This section provides the following examples:

- Retrieve Application Domains
- Create a New Application Domain
- Get All Authentication Schemes
- Create a New Authentication Scheme
- Get a Particular Authentication Scheme
- Get All Resources in an Application Domain
- Create a Resource in an Application Domain
- Get All Policies in an Application Domain

5.4.1 Retrieve Application Domains

cURL Command

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

Sample Output

```
<?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>
```

5.4.2 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
```

Sample Input File

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

Sample Output

```
http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/ssa/policyadmin/appdomain?id=f
a60e312-fe65-4aa8-aace-1735a39c4058
```

5.4.3 Get All Authentication Schemes

cURL Command

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

Sample Output

```
<?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>
```

5.4.4 Create a New 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
```

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>
~
```

Sample Output

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

5.4.5 Get a Particular Authentication Scheme

cURL Command

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

Sample Output

```
<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>
```

5.4.6 Get 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?appdo
main="IAM Suite"
```

5.4.7 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?appdo
main="AppDomain1"
```

5.4.8 Get 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?ap
pdomain="IAM Suite"
```

5.5 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/.

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:

- Section 6.1, "About Impersonation"
- Section 6.2, "Configuring Impersonation Support"
- Section 6.3, "Testing SSO Login and Impersonation"

> **See Also:** "Integrating Oracle ADF Application with Oracle Access Manager 11*g* SSO" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

6.1 About Impersonation

Access Manager user impersonation feature enables one user to perform operations and access resources on the behalf of another. 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:

- Impersonation Concepts and Terminology
- Impersonation Grant Syntax
- Impersonation Trigger Invocation Using the SSO Service

6.1.1 Impersonation Concepts and 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.

Access Manager user impersonation support allows an end user (the impersonatee) to designate one or more other users (impersonators) to act on her 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

6.1.2 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 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=mycompany,dc=com"  "cn=jdoe" orclguid
```

For example, LDAP search for `orclguid`:

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

where:

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

For example, LDAP search for `orclImpersonationGrantee`:

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

where:

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

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

```
A14BEB42E822D605E040E50AB29327E7|20100324163000Z|2012
0524172000Z
```

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.

6.1.3 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 Example 6–1 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 SSO" in *Oracle Fusion Middleware Application Security Guide*.

Example 6–1 illustrates the methods required to use the SSO Service to abstract the specifics of the triggering mechanism (preferred).

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
```

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. Example 6–2 shows an abbreviated example.

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

    sso.beginImpersonation(req, res, m);


    [....]

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

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

Example 6–3 provides a snippet from jps-config.xml showing the configuration changes needed (`imp.begin.url` and `imp.end.url` properties):

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.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
1.xsd">
    <property value="off" name="oracle.security.jps.jaas.mode"/>
    <propertySets>
        <propertySet name="saml.trusted.issuers.1">
            <property value="www.oracle.com" name="name"/>
        </propertySet>

[...]

        <propertySet name="props.auth.uri.0">
            <property value="/oamsso/logout.html" name="logout.url"/>
            <property
value="https://login01.mycompany.com:7777/oam/server/impersonate/end
              name="imp.end.url"/>
            <property
value="https://login01.mycompany.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>

[...]
```

6.1.4 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 Example 6–4.

Example 6–4 Triggering Impersonation Without API Abstraction

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

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 Example 6–5. Use of `failure_url` is optional.

Example 6–5 Restore Original Impersonator's Session

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

6.1.5 Impersonator Identity Communication During Impersonation Sessions

Table 6–2 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

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.

This section contains the following topics:

- 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 as shown in Example 6–6.

This example 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>
```

6.2.2 Configuring Impersonation Using idmConfigTool

For more information about the `idmConfigTool` command, see "Using the idmConfigTool Command" in *Oracle Fusion Middleware Integration Guide for Oracle Identity Management Suite*.

To configure the impersonation feature using `idmConfigTool`:

1. Use `idmConfigTool` with the `-prepareStoreID` 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;90FE8C8083CEBC
   1FE040E50AEBB9176A|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

For impersonation support the authentication scheme for the protected application must be set to LDAPScheme. This must be done before initiating an impersonation session. For more information about the LDAPScheme, see "Managing Authentication

Schemes" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

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**.

6.3 Testing SSO Login and Impersonation

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

To test SSO login and impersonation

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 "About the Session Management Page" in *Oracle Fusion Middleware Administrator's Guide for 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 Section 6.1.5, "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.

Part III

Developing with Mobile and Social

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

Part III contains the following chapters:

7

Developing Applications Using the Mobile and Social Client SDKs

This chapter briefly introduces the Mobile and Social client SDKs. This chapter includes the following topics:

7.1 Before you Begin

Before you start work on an application that will use the Oracle Access Management Mobile and Social service, you should read the "Understanding Mobile and Social" chapter in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*. This Developer's Guide assumes that you understand Mobile and Social terminology and concepts.

In this guide the Mobile and Social *client program* (or *client app*) is the portion of code that you build into an application to utilize authentication, authorization, and user profile services on a remote server. Your application can be any application that uses HTTP. It does not have to be a mobile application.

7.2 Introduction to Developing Mobile Services Applications

Two Client SDKs—iOS and Java—are provided for building Identity security features into your applications and enabling you to use your existing Identity infrastructure for authentication, authorization, and directory-access services. The easiest way to get your app to interact with Mobile Services is to use one of the offered client SDKs.

Note: If you are developing an application on a platform or device that cannot use the iOS or Java SDKs, you can write code to directly send Mobile and Social REST calls to the Mobile and Social server.

See Chapter 12, "Sending Mobile and Social REST Calls With cURL", which documents the Mobile and Social REST API.

If you use a Mobile Services SDK, you do not need to know the REST call syntax that the Mobile Services client uses to communicate with the Mobile and Social server.

The following table lists the features that each Mobile Services Client SDK is capable of.

Table 7–1 Features and Capabilities of the Java and iOS Mobile Services Client SDKs

Feature	iOS	Java	Notes
Build a mobile application that can acquire Client Registration Handle, User, and Access Tokens through a Mobile and Social Server	✔		See Chapter 9, "Developing Mobile Services Applications with the iOS Client SDK."
Build a desktop application that can acquire Client, User, and Access Tokens through a Mobile and Social Server		✔	See Chapter 8, "Developing Mobile Services Applications with the Java Client SDK."
Interact with a Directory server and implement User Profile Services	✔	✔	See Chapter 9, "Developing Mobile Services Applications with the iOS Client SDK." See Chapter 8, "Developing Mobile Services Applications with the Java Client SDK."
Create a mobile single sign-on (SSO) app	✔		See Chapter 9, "Developing Mobile Services Applications with the iOS Client SDK."

7.2.1 Building Applications With User Profile Services

This section contains notes and information about building applications with User Profile Services. This information is not specific to any one SDK.

Case Sensitivity

In general, LDAP attribute names are not case sensitive. When communicating with the Oracle Identity Governance Framework (IGF) APIs, however, LDAP attribute names *are* case sensitive.

Special Characters

Special characters should be replaced with their hex value equivalents in the search filter.

Note: The WebLogic Server embedded LDAP server does not allow special characters to be included in the user name. User names are case sensitive and must be unique. Do not use commas, tabs, or any other characters in the following comma-separated list:

```
< >, #, |, &, ?, ( ), { }
```

7.3 Introduction to Developing Internet Identity Services Applications

Developers who maintain Java-compliant Web applications can add Internet Identity Services functionality to their Web offering using the Mobile and Social Internet Identity Services SDK. This SDK is available for Java-powered Web applications only.

For information about how to use the SDK to integrate Internet Identity Services with a Java-powered Web application, see Chapter 10, "Developing Applications Using the Internet Identity Services Client SDK."

This Developer's Guide also includes information about how to add additional OpenID and OAuth Service Providers by implementing a Java interface. For information, see Section 11.2, "Create a new Identity Service Provider for Internet Identity Services".

8

Developing Mobile Services Applications with the Java Client SDK

This chapter describes how to use the Java Client SDK to build desktop applications. The Java Client SDK does not provide support for building applications on mobile devices. This chapter includes the following topics:

8.1 Overview

The Mobile and Social Java Client SDK for Mobile Services is included in the Oracle Access Management distribution package and can also be downloaded from the Oracle Technical Network (OTN) website.

In addition to this *Developer's Guide*, API documentation generated by the Javadoc tool is available. Refer to the available API documentation for descriptions of API classes, interfaces, constructors, methods, and fields. This documentation is provided as HTML in the SDK, and can also be downloaded from the Oracle Access Management product library in PDF and HTML formats as the *Oracle Fusion Middleware Java API Reference for Mobile and Social*.

8.2 Invoking Authentication Services With the Java Client SDK

This section provides sample code that illustrates how to request a Client Token, a User Token, and an Access Token.

A token contains attributes related to the item, as well as encrypted information that establishes the authority, validity, or identity of the token bearer. A Client Token contains credential information, a User Token encapsulate the Client Token, and an Access Token contains the security information needed to access a protected resource.

The sample code in this section supports the "JWTAuthentication" (JSON Web Token Authentication) service type. Refer to "Configuring Mobile Services" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management* for information about configuring a service provider.

The code samples in this section are organized into the following categories:

- Create a Client Token
- Create a User Token
- Create an Access Token
- Validate a Client Token
- Validate a User Token
- Perform a User Lookup Using the User Token
- Delete the Client Token

8.2.1 Getting Started

Import the following Java client SDK classes from the `oic_clientsdk.jar` file:

```
import oracle.security.idaas.rest.jaxrs.client.sdk.ClientSDKConfig;
import oracle.security.idaas.rest.jaxrs.client.sdk.Headers;
import oracle.security.idaas.rest.jaxrs.client.sdk.HeadersDefaultImpl;
import oracle.security.idaas.rest.jaxrs.client.sdk.OICClientException;
import oracle.security.idaas.rest.jaxrs.client.sdk.authentication.AuthenticationClient;
import oracle.security.idaas.rest.jaxrs.client.sdk.authentication.AuthenticationResult;
import oracle.security.idaas.rest.jaxrs.client.sdk.authentication.TokenCreateRequest;
import oracle.security.idaas.rest.jaxrs.client.sdk.authentication.TokenCreateRequestImpl;
import oracle.security.idaas.rest.jaxrs.client.sdk.authentication.TokenDeleteRequest;
import oracle.security.idaas.rest.jaxrs.client.sdk.authentication.TokenDeleteRequestImpl;
import oracle.security.idaas.rest.jaxrs.client.sdk.authentication.TokenExchangeRequest;
import oracle.security.idaas.rest.jaxrs.client.sdk.authentication.TokenExchangeRequestImpl;
import oracle.security.idaas.rest.jaxrs.client.sdk.authentication.TokenReadRequest;
import oracle.security.idaas.rest.jaxrs.client.sdk.authentication.TokenReadRequestImpl;
```

Initialize the `ClientSDKConfig` object, then define the endpoints for various actions using the service provider `jwtauthentication`. Then initialize the `AuthenticationClient` object.

```
AuthenticationClientSDKConfig cc = new AuthenticationClientSDKConfig();
cc.setRegisterationServiceURI("http://hostcomputer.example.com:18001/
oic_rest/rest/jwtauthentication/register");

cc.setAuthenticationServiceURI("http://hostcomputer.example.com:18001/
oic_rest/rest/jwtauthentication/authenticate");

cc.setAccessTokenServiceURI("http://hostcomputer.example.com:18001/
oic_rest/rest/jwtauthentication/access");

cc.setTokenInfoServiceURI("http://hostcomputer.example.com:18001/
oic_rest/rest/jwtauthentication/tokens/info");

AuthenticationClient tc = new AuthenticationClient(cc);
```

8.2.2 Create a Client Token

Define the required parameters for the Client Token request and then request to create the token. Save the result of the token request in a variable named `savedClientToken`:

```
String subjectType = "USERCREDENTIAL";
String uname = "profileid1";
String password = "secret12";
String tokenTypeToCreate = "CLIENTTOKEN";
TokenCreateRequest tcrd = new TokenCreateRequestImpl(subjectType, uname, password,
tokenTypeToCreate);
```

```
Headers headers = new HeadersDefaultImpl();
AuthenticationResult savedClientToken = tc.createToken(tcrd, headers);
```

8.2.3 Create a User Token

Define the required parameters for the User Token request and request to create the token. Add the Client Token from the previous step to the REST authorization header and save the result of the User Token request in a variable named `savedUserToken`:

```
String subjectType = "USERCREDENTIAL";
String uname = "sean";
String password = "secret12";
String tokenTypeToCreate = "USERTOKEN";
TokenCreateRequest tcrd = new TokenCreateRequestImpl(subjectType, uname, password,
tokenTypeToCreate);
Headers headers = new HeadersDefaultImpl();

//Value expects certain format including type...
String tokenHeaderValue = "TOKEN" + " " + savedClientToken.getValue();
headers.setIdaasRestAuthZHeader(tokenHeaderValue);
AuthenticationResult savedUserToken = tc.createToken(tcrd, headers);
```

8.2.4 Create an Access Token

Define the required parameters for the Access Token request and request to create the token. Save the result of the token request in a variable named `savedAccessToken`.

```
String resource = "http:myserver.com:8080/index.html";
String context = "QaZdhh77randomstuff";
String tokenSubjectValue = savedClientToken.getValue();
String credentialSubjectType = "TOKEN";
String newTokenTypeToCreate = "ACCESSTOKEN";
TokenExchangeRequest tcberd = new TokenExchangeRequestImpl(credentialSubjectType,
tokenSubjectValue, resource, context, newTokenTypeToCreate);
AuthenticationResult savedAccessToken = tc.createToken(tcberd, headers);
```

8.2.5 Validate a Client Token

```
String tokenValueToVerify = savedClientToken.getValue();
String tokenSubjectTypeToVerify = "TOKEN";

headers = new HeadersDefaultImpl();
headers.setIdaasRestAuthZHeader("TOKEN " + tokenValueToVerify);

TokenReadRequest tokenToRead = new TokenReadRequestImpl();
tokenToRead.setSubjectValue(tokenValueToVerify);
tokenToRead.setSubjectType(tokenSubjectTypeToVerify);

AuthenticationResult retrievedToken = tc.readToken(tokenToRead, headers);
System.out.println("Token returned from readToken() =" + retrievedToken.getValue());
if (null != savedClientToken && null != retrievedToken) {
  System.out.println("Does value in savedClientToken == retrievedToken?" +
  savedClientToken.getValue().equals(retrievedToken.getValue()));
}
```

8.2.6 Validate a User Token

```
Headers headers = new HeadersDefaultImpl();
headers.setIdaasRestAuthZHeader("TOKEN " + savedClientToken.getValue());

TokenReadRequest tokenToRead = new TokenReadRequestImpl();
tokenToRead.setSubjectValue(savedUserToken.getValue());
tokenToRead.setSubjectType("TOKEN");
AuthenticationResult retrievedToken = tc.readToken(tokenToRead, headers);
System.out.println("Token returned from readToken() =" + retrievedToken.getValue());
if (null != savedUserToken && null != retrievedToken) {
  System.out.println("Does value in savedUserToken == retrievedToken?" +
  savedUserToken.getValue().equals(retrievedToken.getValue()));
}
```

8.2.7 Perform a User Lookup Using the User Token

In this step, User is a protected resource that is protected by the authentication provider.

```
UserProfileClientSDKConfig cc = new UserProfileClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);

final String SEARCH_PAGE_POSITION_QUERY_PARAM_NAME = "pagePos";
final String SEARCH_PAGE_SIZE_QUERY_PARAM_NAME = "pageSize";
String pageSizeValue = "1";  //Just get one user for this test.
String pageSizePosition = "0";

//Now do a search and fetch first page o results.
Map<String, String> queryParameters = new HashMap<String, String>();
queryParameters.put(SEARCH_PAGE_SIZE_QUERY_PARAM_NAME, pageSizeValue);
queryParameters.put(SEARCH_PAGE_POSITION_QUERY_PARAM_NAME, pageSizePosition);

// Set Header to include the User Token for authetication.
Headers headers = new HeadersDefaultImpl();
headers.setAuthZHeader(savedUserToken);

//Perform search operation.
JSONCollection searchResults = pc.searchUsers(queryParameters, headers);
```

8.2.8 Delete the Client Token

```
String deleteSubjectValue = savedClientToken.getValue(); //use first token value
String deleteTokenType = "TOKEN";
TokenDeleteRequest tokenToDelete = new TokenDeleteRequestImpl();
tokenToDelete.setSubjectValue(deleteSubjectValue);
tokenToDelete.setTokenType(deleteTokenType);
boolean result = false;
result = tc.deleteToken(tokenToDelete, headers);
```

8.3 Invoking User Profile Services with the Java Client SDK

Before working with the code samples in this section, see "Building Applications With User Profile Services" for notes and information that are not specific to this SDK.

The code samples in this section are organized into the following categories:

- Working with People

- Working With Groups
- Working With Organizations
- Searching With Paging Support

8.3.1 Working with People

The following code samples demonstrate how to interact with User records located in a Directory store that User Profile Services can access and update. This section covers the following basic scenarios:

- Getting set up
- Creating a User
- Reading a User
- Updating a User
- Deleting a User
- Searching for a User
- Retrieving User Attributes and Validating the Results

8.3.1.1 Getting set up

First import the following Java classes from the `oic_clientsdk.jar` file, then declare the "people" Service URI global variable.

```
import oracle.security.idaas.rest.jaxrs.client.sdk.ClientSDKConfig;
import oracle.security.idaas.rest.jaxrs.client.sdk.Headers;
import oracle.security.idaas.rest.jaxrs.client.sdk.HeadersDefaultImpl;
import oracle.security.idaas.rest.jaxrs.client.sdk.OICClientException;
import oracle.security.idaas.rest.jaxrs.client.sdk.userprofile.JSONCollection;
import oracle.security.idaas.rest.jaxrs.client.sdk.userprofile.PeopleClient;

private static String serviceURI = "http://hostcomputer.example.com:18001/oic_
rest/rest/userprofile/people";
```

8.3.1.2 Creating a User

The following sample creates a User record with uid `peopletestuser123`.

```
UserProfileClientSDKConfig cc = new UserProfileClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);

//Just generate some fake user info.
String uid = "peopletestuser123";
String userpassword = "secret123";
String sn = uid;
String cn = uid;
String mail = uid + "@example.com";

//Now put these values into the resourceAttrs map, and pass to helper.
Map<String, Object> resourceAttrs = new HashMap<String, Object>();
resourceAttrs.put("uid", uid);
resourceAttrs.put("password", userpassword);
resourceAttrs.put("lastname", sn);
resourceAttrs.put("commonname", cn);
resourceAttrs.put("mail", mail);
List<String> phoneNums = new ArrayList<String>();
phoneNums.add("408-123-5555");
```

```
phoneNums.add("408-123-9999");
resourceAttrs.put("telephone", phoneNums);
String personJson = pc.createUser(resourceAttrs, new HeadersDefaultImpl());
```

8.3.1.3 Reading a User

The following sample retrieves the User record with uid `peopletestuser123`.

```
UserProfileClientSDKConfig cc = new UserProfileClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);

String uidForExistingUser = "peopletestuser123";

//now GET that user just to check
Map<String, String> queryParameters = new HashMap<String, String>();//none yet
String existingUser = pc.readUser(uidForExistingUser, queryParameters, new HeadersDefaultImpl());
boolean found = false;
JSONObject jo = new JSONObject(existingUser);
String s = jo.getString("uid");
found = s.equalsIgnoreCase(uid);
```

8.3.1.4 Updating a User

The following sample updates the User record with uid `peopletestuser123`.

```
UserProfileClientSDKConfig cc = new UserProfileClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);

//Just generate some fake user info.
final String CN_VALUE = "UPDATED CN";

String uidForExistingUser = "peopletestuser123"; //From class-defined uid.

//now make some attributes with new values to update
Map<String, Object> attrsToUpdate = new HashMap<String, Object>();
attrsToUpdate.put("commonname", CN_VALUE);
String result = pc.updateUser(uidForExistingUser, attrsToUpdate,new HeadersDefaultImpl());
```

8.3.1.5 Deleting a User

The following sample deletes the User record with uid `peopletestuser123`.

```
UserProfileClientSDKConfig cc = new UserProfileClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);

boolean deleteResult = pc.deleteUser("peopletestuser123", new HeadersDefaultImpl());
```

8.3.1.6 Searching for a User

The following sample searches for the User record with uid `peopletestuser123`.

```
UserProfileClientSDKConfig cc = new UserProfileClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);

//now do a search on uid attribute
Map<String, String> queryParameters = new HashMap<String, String>();
String queryValue = "peopletestuser"+ "*";
queryParameters.put("searchparam.uid", queryValue);

//Set query parameters and empty headers.
JSONCollection searchResult = pc.searchUsers(queryParameters, new HeadersDefaultImpl());

//Get raw JSON array value in "elements" attribute.
```

```
String elementJSONString = searchResult.getJsonArrayElements();
JSONArray ja = null;
ja = new JSONArray(elementJSONString);

//Now try to match the result to the expected User with uid.
JSONObject elem = null;
boolean found = false;
for(int i=0; i<ja.length() && found==false; i++) {
  elem = ja.getJSONObject(i);  //Get item from array
  String u = elem.getString("uid");

  //Check if attr is present AND matches some value.
  if(u.equalsIgnoreCase("peopletestuser123")) {
    found = true;
  }
}
```

8.3.1.7 Retrieving User Attributes and Validating the Results

The following sample retrieves the user attribute `commonname` and checks that the attribute `description` is not present.

```
final String ATTRIBUTES_TO_FETCH_QUERY_PARAM_NAME = "attrsToFetch";
String attributeToFetchName = "commonname";  //fetch this attribute
String attributeShouldNotBePresent = "description";
ClientSDKConfig cc = new ClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);

//Now GET that User just to check.
Map<String, String> queryParameters = new HashMap<String, String>();
queryParameters.put(ATTRIBUTES_TO_FETCH_QUERY_PARAM_NAME, attributeToFetchName);
String existingUser = pc.readUser("peopletestuser123", queryParameters, new HeadersDefaultImpl());
boolean found = false;
try {
  JSONObject jo = new JSONObject(existingUser);
  //Throws exception if attribute not present
  String s = jo.getString(attributeToFetchName);
  found = true;
} catch (JSONException je) {
  found = false;
}

//Now verify that a certain attribute is NOT present.
found = false;
try {
  JSONObject jo = new JSONObject(existingUser);

  //throws exception if attribute not present
  for(Iterator it = jo.keys(); it.hasNext() && found==false; ) {
    String key = (String) it.next();
      if(key.equalsIgnoreCase(attributeShouldNotBePresent)) {
        found = true; //Bad if present because it should not be.
      }
  }
} catch (JSONException je) {}
```

8.3.2 Working With Groups

A *group* is a set of Users.

This section presents code samples that cover the following basic scenarios:

- Getting set up
- Creating a Group
- Reading a Group
- Updating a Group
- Deleting a Group
- Searching a Group
- Searching Groups With Paging Support
- Adding a User to a Group
- Getting Group Membership Info
- Searching for a Member Within a Group
- Removing a Member From a Group
- Assigning Group Ownership
- Getting Group Ownership Info
- Searching for the Owner of a Group
- Removing a Group Owner
- Adding a Group (or a User) to a Group Using addMemberOf
- Getting the Membership of a Group Using getMemberOf
- Searching a Group Using searchMemberOf
- Removing a Group (or a User) from a Group Using deleteMemberOf
- Assigning Group Ownership Using addOwnerOf
- Getting Group Ownership Info Using getOwnerOf
- Searching for the Owner of a Group Using searchOwnerOf

8.3.2.1 Getting set up

First import the following Java classes, then declare the "groups" Service URI global variable.

```
import oracle.security.idaas.rest.jaxrs.client.sdk.ClientSDKConfig;
import oracle.security.idaas.rest.jaxrs.client.sdk.HeadersDefaultImpl;
import oracle.security.idaas.rest.jaxrs.client.sdk.OICClientException;
import oracle.security.idaas.rest.jaxrs.client.sdk.userprofile.GroupsClient;
import oracle.security.idaas.rest.jaxrs.client.sdk.userprofile.JSONCollection;

private static GroupsClient gc = null;
private static PeopleClient pc = null;

private static String roleServiceURI = 'http://hostcomputer.example.com:18001/oic_rest/
rest/userprofile/groups";

private static String peopleServiceURI = "http://hostcomputer.example.com:18001/oic_rest/
rest/userprofile/people";
```

```
Map<String, String> accessURIMap = Util.createAccessURIMap("manager", "reports", "memberOf",
"members", "groupMemberOf", "groupMembers", "ownerOf", "personOwner", "groupOwner",
"groupOwnerOf");

Map<String, String> entityURIMap = Util.createEntityURIMap("report-uri", "manager-uri",
"person-uri", "group-uri", "member-uri", "group-uri", "owner-uri", "group-uri", "group-uri",
"owner-uri");

UserProfileClientSDKConfig cc = new UserProfileClientSDKConfig(roleServiceURI);
cc.setAccessURIMap(accessURIMap);
cc.setEntityURIMap(entityURIMap);
gc = new GroupsClient(cc);

UserProfileClientSDKConfig cc2 = new UserProfileClientSDKConfig(peopleServiceURI);
cc2.setAccessURIMap(accessURIMap);
cc2.setEntityURIMap(entityURIMap);
pc = new PeopleClient(cc2);
```

8.3.2.2 Creating a Group

```
Map<String, Object> resourceAttrs = new HashMap<String, Object>();
resourceAttrs.put("commonname", "testGroup");
resourceAttrs.put("description", "testGroupDescription");
String creategroup = gc.createGroup(resourceAttrs, new HeadersDefaultImpl());
```

8.3.2.3 Reading a Group

```
String readgroup = gc.readGroup("testGroup", new HashMap<String, String>(), new
HeadersDefaultImpl());
```

8.3.2.4 Updating a Group

```
Map<String, Object> resourceAttrs2 = new HashMap<String, Object>();
resourceAttrs2.put("description", "new description");
String udpatedgroup = gc.updateGroup("testGroup", resourceAttrs2, new HeadersDefaultImpl());
```

8.3.2.5 Deleting a Group

```
boolean deletedgroup = gc.deleteGroup("testGroup", new HeadersDefaultImpl());
```

8.3.2.6 Searching a Group

```
//search with searchOperator = OR, commonname and description
Map<String, String> queryParams = new HashMap<String,String>();
String commonname = "testGroup" + 1;
String description = "testGroup" + "Description";
queryParams.put("searchparam.commonname", commonname);
queryParams.put("searchparam.description", description);
queryParams.put("searchFilter","SimpleOR");

JSONCollection searchResult = gc.searchGroups(queryParams, new HeadersDefaultImpl());

//get raw JSON array value in "elements" attribute
```

```
String elementJSONString = searchResult.getJsonArrayElements();
JSONArray ja = new JSONArray(elementJSONString);
```

8.3.2.7 Searching Groups With Paging Support

The following sample searches for a group and returns the results one page at a time.

```
final String SEARCH_PAGE_POSITION_QUERY_PARAM_NAME = "pagePos";
final String SEARCH_PAGE_SIZE_QUERY_PARAM_NAME = "pageSize";
String pageSizeValue = "1"; //just get one group for this test
String pageSizePosition = "0";

//now do a search and fetch first page o results
Map<String, String> queryParams = new HashMap<String, String>();
queryParams.put(SEARCH_PAGE_SIZE_QUERY_PARAM_NAME, pageSizeValue);
queryParams.put(SEARCH_PAGE_POSITION_QUERY_PARAM_NAME, pageSizePosition);
JSONCollection searchResults = gc.searchGroups(queryParams, new HeadersDefaultImpl());

//get raw JSON array value in "elements" attribute
String elementJSONString = searchResults.getJsonArrayElements();
JSONArray ja = null;
ja = new JSONArray(elementJSONString);
boolean justOneFound = false;


//the search returns a set with just one user
if (ja.length() == Integer.parseInt(pageSizeValue)) {
  justOneFound = true;
}
```

8.3.2.8 Adding a User to a Group

The following sample uses the `addPersonMember` method. Also see Adding a Group (or a User) to a Group Using addMemberOf.

```
String resultRoleMembership = gc.addPersonMember("testGroup", "testuser123", new
HeadersDefaultImpl());
```

8.3.2.9 Getting Group Membership Info

The following sample uses the `getPersonMember` method. Also see Getting the Membership of a Group Using getMemberOf.

```
Map<String, String> queryParameters = new HashMap<String, String>(); //none yet
String membershipId ="testuser123";
String result = gc.getPersonMember("testGroup",membershipId,queryParameters, new
HeadersDefaultImpl());
```

8.3.2.10 Searching for a Member Within a Group

The following sample uses the `searchGroupMembers` method. Also see Searching a Group Using searchMemberOf.

```
String queryFilter = "(uid=" +"*)";
Map<String, String> queryParams = new HashMap<String, String>();
queryParams.put("nativequery", queryFilter);

//need to use membership uri such as ...doctors/members
JSONCollection searchResults = gc.searchPersonMembers("testGroup", queryParams, new
HeadersDefaultImpl());
```

```
//get raw JSON array value in "elements" attribute
String elementJSONString = searchResults.getJsonArrayElements();
JSONArray ja = null;
ja = new JSONArray(elementJSONString);

//Sample of how to get the members' URIs. A client could call GET on each of these
// persons' URIs using the person client API to get details about each member.
Set<String> userUriSet = new HashSet<String>();
final String PERSON_URI_FIELD_NAME = "person-uri";
for (int i=0; i<ja.length(); i++) {
   JSONObject jo = ja.getJSONObject(i);

   //Get the URI field of this user.
   String uri = jo.getString(PERSON_URI_FIELD_NAME);
   if (uri != null && !uri.isEmpty()) {
      userUriSet.add(uri);
   }
}

// Get Group members in the group.
searchResults = gc.searchGroupMembers("testGroup", queryParams, new HeadersDefaultImpl());
```

8.3.2.11 Removing a Member From a Group

The following sample uses the `deletePersonMember` method. Also see Removing a Group (or a User) from a Group Using deleteMemberOf.

```
boolean result = gc.deletePersonMember("testGroup", "testuser123", new HeadersDefaultImpl());
```

8.3.2.12 Assigning Group Ownership

The following sample demonstrates how to assign ownership of a group to a user or a group.

```
// Add user testuser123 to group testGroup as group owner.
String resultRoleOwnership = gc.addPersonOwner("testGroup", "testuser123",
new HeadersDefaultImpl());

// Add group testSubGroup to group testGroup as group owner.
String resultRoleOwnership2 = gc.addGroupOwner("testGroup", "testSubGroup",
new HeadersDefaultImpl());
```

8.3.2.13 Getting Group Ownership Info

```
Map<String, String> queryParameters = new HashMap<String, String>();//none yet
String ownershipId="testuser123";
String result = gc.getPersonOwner("testGroup", ownershipId, queryParameters,
new HeadersDefaultImpl());
ownershipId ="testSubGroup";
result = gc.getGroupOwner("testGroup", ownershipId, queryParameters,
new HeadersDefaultImpl());
```

8.3.2.14 Searching for the Owner of a Group

```
String queryFilter = "(uid=" +"*)";
Map<String, String> queryParams = new HashMap<String, String>();
queryParams.put("nativequery", queryFilter);

// Get Person owners in the group.
JSONCollection searchResults = gc.searchPersonOwners("testGroup", queryParams,
new HeadersDefaultImpl());

// Get raw JSON array value in the "elements" attribute.
String elementJSONString = searchResults.getJsonArrayElements();
JSONArray ja = null;
ja = new JSONArray(elementJSONString);

// Sample of how to get the members' URIs.  A client could call GET on each of these
// person URIs using the person client API and get details on each member.
Set<String> userUriSet = new HashSet<String>();
final String OWNER_URI_FIELD_NAME = "owner-uri";
for(int i=0; i<ja.length(); i++) {
  JSONObject jo = ja.getJSONObject(i);

  //Get URI field of this user.
  String uri = jo.getString(OWNER_URI_FIELD_NAME);
  if (uri != null && !uri.isEmpty()) {
    userUriSet.add(uri);
  }
}

// Get Group owners in the group.
searchResults = gc.searchGroupOwners("testGroup", queryParams, new HeadersDefaultImpl());
```

8.3.2.15 Removing a Group Owner

```
boolean result = gc.deletePersonOwner("testGroup", "testuser123", new HeadersDefaultImpl());
boolean result2= gc.deleteGroupOwner("testGroup", "testSubGroup", new HeadersDefaultImpl());
```

8.3.2.16 Adding a Group (or a User) to a Group Using addMemberOf

The following sample demonstrates how to use the `addMemberOf` method to make a group a member of another group, or how to make a user a member of a group.

```
// Add group "testSubGroup" to be a member of group "testGroup"
String resultRoleMembership2= gc.addMemberOf("testGroup", "testSubGroup",
new HeadersDefaultImpl());

// Add user "testuser123" to be a member of group "testGroup"
String resultRoleMembership = pc.addMemberOf("testuser123", "testGroup",
new HeadersDefaultImpl());
```

8.3.2.17 Getting the Membership of a Group Using getMemberOf

The following sample demonstrates how to use the `getMemberOf` method to get relationship data about a specified group.

```
// Get relationship data where user "testuser123" is a member of group "testGroup"
String resultRoleMembership = pc.getMemberOf("testuser123", "testGroup", new HeadersDefaultImpl());
```

```
// Get relationship data where group "testsubGroup" is a member of group "testGroup"
String resultRoleMembership2= gc.getMemberOf("testGroup", "testSubGroup",
new HeadersDefaultImpl());
```

8.3.2.18 Searching a Group Using searchMemberOf

```
String queryFilter = "(uid=" +"*)";
Map<String, String> queryParams = new HashMap<String, String>();
queryParams.put("nativequery", queryFilter);

// Search groups of which Person "testuser123" is a member
JSONCollection searchResults = pc.searchMemberOf("testuser123", queryParams,
new HeadersDefaultImpl());

//Get raw JSON array value in "elements" attribute
String elementJSONString = searchResults.getJsonArrayElements();
JSONArray ja = null;
ja = new JSONArray(elementJSONString);

// Sample of how to get the members' URIs. A client could call GET on each of these
// person URIs using the person client API to get details about each member.
Set<String> groupUriSet = new HashSet<String>();
final String GROUP_URI_FIELD_NAME = "group-uri";
for(int i=0; i<ja.length(); i++) {
  JSONObject jo = ja.getJSONObject(i);


  //Get URI field of this user.
  String uri = jo.getString(GROUP_URI_FIELD_NAME);
  if (uri != null && !uri.isEmpty()) {
    groupUriSet.add(uri);
  }
}

// Search Groups of which group "testSbuGroup" is a member.
searchResults = gc.searchMemberOf("testSubGroup", queryParams, new HeadersDefaultImpl());
```

8.3.2.19 Removing a Group (or a User) from a Group Using deleteMemberOf

```
// Delete member "testuser123" from group "testGroup"
boolean result = pc.deleteMemberOf("testuser123","testGroup", new HeadersDefaultImpl());

// Delete member "testSubGroup" from group "testGroup"
boolean result2= gc.deleteMemberOf("testGroup","testSubGroup", new HeadersDefaultImpl());
```

8.3.2.20 Assigning Group Ownership Using addOwnerOf

```
// Add user "testuser123" to be an owner of group "testGroup"
String resultRoleOwnership = pc.addOwnerOf("testuser123", "testGroup", new HeadersDefaultImpl());

// Add group "testSubGroup" to be an owner of group "testGroup"
String resultRoleOwnership2 = gc.addOwnerOf("testGroup", "testSubGroup", new HeadersDefaultImpl());
```

8.3.2.21 Getting Group Ownership Info Using getOwnerOf

```
// Get relationship data where user "testuser123" is an owner of group "testGroup"
String resultRoleOwnership = pc.getOwnerOf("testuser123", "testGroup", new HeadersDefaultImpl());

// Get relationship data where group "testsubGroup" is an owner of group "testGroup"
String resultRoleOwnership2= gc.getOwnerOf("testGroup", "testSubGroup", new HeadersDefaultImpl());
```

8.3.2.22 Searching for the Owner of a Group Using searchOwnerOf

```
String queryFilter = "(uid=" +"*)";
Map<String, String> queryParams = new HashMap<String, String>();
queryParams.put("nativequery", queryFilter);

// Search Groups of which Person "testuser123" is an owner.
JSONCollection searchResults = pc.searchOwnerOf("testuser123", queryParams,
new HeadersDefaultImpl());

// Get raw JSON array value in "elements" attribute.
String elementJSONString = searchResults.getJsonArrayElements();
JSONArray ja = null;
ja = new JSONArray(elementJSONString);


// Sample of how to get the members' URIs. A client could call GET on each of these person URIs
using the person client API to get details about each member.
Set<String> groupUriSet = new HashSet<String>();
final String GROUP_URI_FIELD_NAME = "group-uri";
for(int i=0; i<ja.length(); i++) {
  JSONObject jo = ja.getJSONObject(i);

  // Get URI field of this user.
  String uri = jo.getString(GROUP_URI_FIELD_NAME);
  if (uri != null && !uri.isEmpty()) {
    groupUriSet.add(uri);
  }
}

// Search Groups of which group "testSbuGroup" is an owner.
searchResults = gc.searchOwnerOf("testSubGroup", queryParams, new HeadersDefaultImpl());
```

8.3.2.23 Removing a Group (or a User) from a Group Using deleteOwnerOf

```
// Delete owner "testuser123" from group "testGroup"
boolean result = pc.deleteOwnerOf("testuser123","testGroup", new HeadersDefaultImpl());

// Delete owner "testSubGroup" from group "testGroup"
boolean result2= gc.deleteOwnerOf("testGroup","testSubGroup", new HeadersDefaultImpl());
```

8.3.3 Working With Organizations

An *organization* is a hierarchical group of people that usually includes a manager and reports.

This section presents code samples that cover the following basic scenarios:

- Getting set up
- Creating Helper Utilities
- Verifying a Manager
- Verifying Direct Reports
- Retrieve All Reports Using Scope=All Feature
- Retrieve the Manager Chain Using Scope=toTop Feature
- Retrieve Report Details Using Pre-Fetch Feature
- Retrieve Manager Data using the Pre-Fetch feature
- Deleting a Report From the Manager

8.3.3.1 Getting set up

First import the following Java classes, then declare the "groups" Service URI global variable.

```
import oracle.security.idaas.rest.jaxrs.client.sdk.ClientSDKConfig;
import oracle.security.idaas.rest.jaxrs.client.sdk.Headers;
import oracle.security.idaas.rest.jaxrs.client.sdk.HeadersDefaultImpl;
import oracle.security.idaas.rest.jaxrs.client.sdk.userprofile.PeopleClient;

private static String personServiceURI= "http://hostcomputer.example.com:18001/oic_
rest/rest/userprofile/people";

private static String peopleBaseURI = "/oic_rest/rest/userprofile/people";
```

8.3.3.2 Creating Helper Utilities

The three helper utilities in this section are useful when working with organization data.

Helper Utility for Creating User Data

```
public static String createPersonHelper(String personServiceURI, String username,String
password,Map<String, String> optionalAttributes) {

ClientSDKConfig cc = new ClientSDKConfig(personServiceURI);
PeopleClient pc = new PeopleClient(cc);

//Generate some fake user info.
String uid = username;
String userpassword = password;
String sn = uid;
String cn = uid;
String mail = uid + "@example.com";

try {
//now put these values into the resourceAttrs map, and pass to helper
//these java string names need to match the json field names
```

```
Map<String, Object> resourceAttrs = new HashMap<String, Object>();
resourceAttrs.put("uid", uid);
resourceAttrs.put("password", userpassword);
resourceAttrs.put("lastname", sn);
resourceAttrs.put("commonname", cn);
resourceAttrs.put("mail", mail);
if (optionalAttributes != null && !optionalAttributes.isEmpty()) {
for(Map.Entry<String, String> me : optionalAttributes.entrySet()) {
resourceAttrs.put(me.getKey(), me.getValue());
}
}

String newUser = pc.createUser(resourceAttrs, new HeadersDefaultImpl());

}
```

Helper Utility for Establishing Manager and Reports Relationships

```
private static boolean assignManagerToUser(String personServiceURI, String serviceBaseURI, String
userUID, String theManagerId) {
ClientSDKConfig cc = new ClientSDKConfig(personServiceURI);
PeopleClient pc = new PeopleClient(cc);
final String MANAGER_URI_SEGMENT_NAME = "manager";
//now make payload
final String MANAGER_URI_JSON_ATTRIBUTE_NAME = "manager-uri";
final String REPORTS_URI_JSON_ATTRIBUTE_NAME = "report-uri";
Map<String, Object> resourceAttrs = new HashMap<String, Object>();
resourceAttrs = new HashMap<String, Object>();
//use base URI od people service within json values
String theManagerURIValue = serviceBaseURI + "/" + theManagerId;
resourceAttrs.put(MANAGER_URI_JSON_ATTRIBUTE_NAME, theManagerURIValue);
String theReporteeURIValue = serviceBaseURI + "/" + userUID; //user being added to list of reports
resourceAttrs.put(REPORTS_URI_JSON_ATTRIBUTE_NAME, theReporteeURIValue);

return pc.addUserToOrgChart(userUID, MANAGER_URI_SEGMENT_NAME, resourceAttrs, new
HeadersDefaultImpl());
}
```

Data Preparation Utility

This utility creates users at different hierarchy levels within an organization.

```
String theUIDofManager = null;
Map<String, String> optionalAttributes = new HashMap<String, String>();
optionalAttributes.put("manager", theUIDofManager);
//keep a map of created people in orgchart
Map<String, String> createdPeople= new HashMap<String, String>();
String userPassword = "secret123";
String userId = "ceo"+ "orgcharttestuser"+ "123"; // user is CEO
String person = Util.createPersonHelper(peopleServiceURI, userId, userPassword, optionalAttributes
);

theUIDofManager = userId; //set to previously created user
userId = "director" + "orgcharttestuer" + "123"; // user id DIRECTOR
optionalAttributes = new HashMap<String, String>();//reset for each new user
person = Util.createPersonHelper(peopleServiceURI, userId, userPassword, optionalAttributes);

//now assign this newly created user DIRECTOR's manager to be CEO
```

```
assignManagerToUser(peopleServiceURI, peopleBaseURI, userId, theUIDofManager);

theUIDofManager = userId; //set to previously created user
userId = "developer111" + "orgcharttestuser" + "123"; // user is DEVELOPER111
optionalAttributes = new HashMap<String, String>();//reset for each new user
person = Util.createPersonHelper(peopleServiceURI, userId, userPassword, optionalAttributes);

//now assign this newly created user DEVELOPER111's manager to be DIRECTOR
assignManagerToUser(peopleServiceURI, peopleBaseURI, userId, theUIDofManager);

userId = "developer222"+ "orgcharttestuser"+"123"; // user is DEVELOPER222
optionalAttributes = new HashMap<String, String>();//reset for each new user
person = Util.createPersonHelper(peopleServiceURI, userId, userPassword, optionalAttributes);
//now assign this newly created user DEVELOPER222's manager to be DIRECOTR
assignManagerToUser(peopleServiceURI, peopleBaseURI, userId, theUIDofManager);
```

8.3.3.3 Verifying a Manager

```
//Set empty query parameters and empty headers.
Map<String, String> searchQueryParameters = new HashMap<String, String>();
Headers searchHeaders = new HeadersDefaultImpl();
JSONCollection resultSet = pc.searchManagers("developer222orgcharttestuser123",
searchQueryParameters, searchHeaders);

//get raw JSON array value in "elements" attribute
String elementJSONString = resultSet.getJsonArrayElements();

boolean found = false;
final String MANAGER_URI_ATTRIBUTE_NAME = "manager-uri";
JSONArray ja = new JSONArray(elementJSONString);
for(int i=0; i< ja.length() && found==false; i++) {
    JSONObject elem = ja.getJSONObject(i);//get item from array
    try {
      //The "manager-uri" attribute of this item in element array is
      //expanded automatically so its value is a JSONObject.
      JSONObject managerURIObject = elem.getJSONObject(MANAGER_URI_ATTRIBUTE_NAME);

      //Check if attr is present AND matches some value.
      if(managerURIObject.getString("uri").equalsIgnoreCase("directororgcharttestuser123")) {
        found = true;
      }
    } catch (JSONException je) {
    //An exception is thrown if attribute is not found or is not a JSON object
    //found = false;
    }

//print out each user, until found
}
```

8.3.3.4 Verifying Direct Reports

```
Map<String, String> searchQueryParameters = new HashMap<String, String>();
Headers searchHeaders = new HeadersDefaultImpl();
JSONCollection resultSet = pc.searchReportees("ceoorgcharttestuser123",
searchQueryParameters, searchHeaders);
```

```
//Get raw JSON array value in "elements" attribute.
String elementJSONString = resultSet.getJsonArrayElements();

boolean found = false;
final String REPORTS_URI_ATTRIBUTE_NAME = "report-uri";

JSONArray ja = new JSONArray(elementJSONString);
for(int i=0; i< ja.length() && found==false; i++) {
  JSONObject elem = ja.getJSONObject(i); //Get item from array
  try {
    JSONObject reportURIObject = elem.getJSONObject(REPORTS_URI_ATTRIBUTE_NAME);

    //Check if attr is present AND matches some value.
    if(reportURIObject.getString("uri").equalsIgnoreCase("directororgcharttestuser123")) {
      found = true;
    }
  } catch (JSONException je) {
  //exception is thrown if attribute is not found or is not JSON object
  //found = false;
  }

//Print out each user, until found.
}
```

8.3.3.5 Retrieve All Reports Using Scope=All Feature

The following code sample verifies all of the reports in an organization, including indirect reports.

```
ClientSDKConfig cc = new ClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);

//Now test CEO orgchart by getting reports with scope=all, which should include developer.
String orgChartIdURI = "reports";

//Now do a search and fetch first page o results.
Map<String, String> queryParameters = new HashMap<String, String>();
queryParameters.put(ClientConstants.ATTRIBUTES_TO_ORG_CHART_SCOPE_QUERY_PARAM_NAME, "all");
JSONCollection resultSet = pc.searchReportees("ceoorgcharttestuser123", queryParameters,
new HeadersDefaultImpl());

//Get raw JSON array value in "elements" attribute.
String elementJSONString = resultSet.getJsonArrayElements();
boolean found = false;
JSONArray ja = new JSONArray(elementJSONString);
for (int i=0; i<ja.length(); i++) {
  JSONObject jo = ja.getJSONObject(i);
  Object reportURIObj = jo.get("report-uri");
  if (reportURIObj.toString().indexOf( "developer111orgcharttestuser123") != -1) {
    found = true;
  }
}
```

8.3.3.6 Retrieve the Manager Chain Using Scope=toTop Feature

The following code sample uses the `toTop` attribute to retrieve an array that contains the managers in a management chain.

```
UserProfileClientSDKConfig cc = new UserProfileClientSDKConfig(serviceURI);
```

```
PeopleClient pc = new PeopleClient(cc);

// Now do a search and fetch the first page of results.
Map<String, String> queryParameters = new HashMap<String, String>();
queryParameters.put(ClientConstants.ATTRIBUTES_TO_ORG_CHART_SCOPE_QUERY_PARAM_NAME, "toTop");
JSONCollection resultSet = pc.searchManagers("developer111orgcharttestuser123",
queryParameters, new HeadersDefaultImpl());

// Get raw JSON array value in "elements" attribute.
String elementJSONString = resultSet.getJsonArrayElements();
```

8.3.3.7 Retrieve Report Details Using Pre-Fetch Feature

The following code samples retrieves manager details when the Report ID and the Manager ID are known.

```
ClientSDKConfig cc = new ClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);
final String ATTRIBUTES_TO_PREFETCH_QUERY_PARAM_NAME = ClientConstants.ATTRIBUTES_TO_PRFFETCH_
QUERY_PARAM_NAME;
String attributeToPrefetch = "report-uri";
final String MANAGER_URI_SEGMENT_NAME = "manager";

//Now read/get new user's details.
String reporteeId = "developer111orgcharttestuser123";
String managerId = "directororgcharttestuser123";

//Now GET that user just to check.
Map<String, String> queryParameters = new HashMap<String, String>();
queryParameters.put(ATTRIBUTES_TO_PREFETCH_QUERY_PARAM_NAME, attributeToPrefetch);

//Get raw JSON representation.
String existingManagerRel = pc.getManager(reporteeId, managerId, queryParameters, new
HeadersDefaultImpl());

//Now obtain manager details and retrieve the reports data.
JSONObject jo = new JSONObject(existingManagerRel);
Object managerAttributeValue = jo.get(attributeToPrefetch);
```

8.3.3.8 Retrieve Manager Data using the Pre-Fetch feature

```
UserProfileClientSDKConfig cc = new UserProfileClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);

final String ATTRIBUTES_TO_PREFETCH_QUERY_PARAM_NAME = ClientConstants.ATTRIBUTES_TO_PRFFETCH_
QUERY_PARAM_NAME;
String attributeToPrefetchName = "manager(commonname)";
Map<String, String> queryParameters = new HashMap<String, String>();
queryParameters.put(ATTRIBUTES_TO_PREFETCH_QUERY_PARAM_NAME, attributeToPrefetchName);

// Get the raw JSON representation of the person.
String existingUser = pc.readUser("developer111orgcharttestuser123", queryParameters, new
HeadersDefaultImpl());

// Get the manager attribute, which is expanded by prefetch to include one or more
// sub-attributes, so that manager is a JSON object within the person JSON.
// Now it is a JSONObject.
JSONObject jo = new JSONObject(existingUser);
```

```
Object managerAttributeValue = (Object) jo.get("manager");
System.out.println(CLASS_NAME + "." + METHOD + ": prefetch detail="
+ managerAttributeValue);
```

8.3.3.9 Deleting a Report From the Manager

```
ClientSDKConfig cc = new ClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);
String uidForExistingUser = "developer111orgcharttestuser123";
String theManagerId = "directororgcharttestuser123";
final String REPORTS_URI_SEGMENT_NAME = "reports";
Map<String, String> queryParameters = new HashMap<String, String>(); //None yet.
String existingOrgChartInstanceDetails = pc.getReportee(theManagerId, uidForExistingUser,
queryParameters, new HeadersDefaultImpl());

//Now that we verified it exists, delete this membership in the reports list.
boolean deleteResult = pc.deleteOrgChartInstance(theManagerId, REPORTS_URI_SEGMENT_NAME,
uidForExistingUser, new HeadersDefaultImpl());

//Now try to get/read that user again. This time we should not find the user.
queryParameters = new HashMap<String, String>(); //None yet.
existingOrgChartInstanceDetails = null;
try {
  existingOrgChartInstanceDetails = pc.readOrgChartInstance(theManagerId, REPORTS_URI_SEGMENT_NAME,
uidForExistingUser, queryParameters, new HeadersDefaultImpl());
} catch (OICClientException ce) {
  System.out.println("existingOrgChartInstanceDetails was successfully deleted so not found"
  + " on subsequent read.");
}
```

8.3.4 Searching With Paging Support

```
UserProfileClientSDKConfig cc = new UserProfileClientSDKConfig(serviceURI);
PeopleClient pc = new PeopleClient(cc);

final String SEARCH_PAGE_POSITION_QUERY_PARAM_NAME = "pagePos";
final String SEARCH_PAGE_SIZE_QUERY_PARAM_NAME = "pageSize";
String pageSizeValue = "1"; //Just get one user for this test.
String pageSizePosition = "0";

//Now do a search and fetch first page o results.
Map<String, String> queryParameters = new HashMap<String, String>();
queryParameters.put(SEARCH_PAGE_SIZE_QUERY_PARAM_NAME, pageSizeValue);
queryParameters.put(SEARCH_PAGE_POSITION_QUERY_PARAM_NAME, pageSizePosition);

//Set query params and empty headers.
JSONCollection searchResults = pc.searchUsers(queryParameters, new HeadersDefaultImpl());

//Get raw JSON array value in "elements" attribute
String elementJSONString = searchResults.getJsonArrayElements();
JSONArray ja = null;
ja = new JSONArray(elementJSONString);
boolean justOneFound = false;

//The search returns a set with just one user.
if (ja.length() == Integer.parseInt(pageSizeValue)) {
  justOneFound = true;
}
```

8.4 Invoking Authorization Services With the Java Client SDK

This example demonstrates accessing the Authorization Service, which is protected by the Access Manager Authentication Service.

```
String clientToken = null;
String userToken = null;
ClientSDKConfig cc = null;
AuthenticationClient authNClient = null;
AuthorizationClient authZClient = null;
Headers headers = new HeadersDefaultImpl();
headers.setContractName("Default");

TokenCreateRequest req = null;
AuthenticationResult resultToken = null;


// Create a Client Token.
cc = new ClientSDKConfig("http://hostcomputer.example.com:18001/oic_
rest/rest/oamauthentication/authenticate");

authNClient = new AuthenticationClient(cc);
req = new TokenCreateRequestImpl("USERCREDENTIAL", "profileid1", "secret12",
"CLIENTTOKEN");
headers = new HeadersDefaultImpl();
headers.setContractName("Default");
resultToken = authNClient.createToken(req, headers);
clientToken = resultToken.getValue();
System.out.println("ClientToken from REST Service : " + clientToken);

// Create a User Token.
req = new TokenCreateRequestImpl("USERCREDENTIAL", "jane", "secret12",
"USERTOKEN");
headers = new HeadersDefaultImpl();
headers.setIdaasRestAuthZHeader("TOKEN " + clientToken);
headers.setContractName("Default");

resultToken = authNClient.createToken(req, headers);
userToken = resultToken.getValue();
System.out.println("UserToken from REST Service : " + userToken);


// Access the Authorization Service using the User Token.
cc = new ClientSDKConfig("http://hostcomputer.example.com:18001/idaas_
rest/rest/oamauthorization/authorization");
authZClient = new AuthorizationClient(cc);

headers = new HeadersDefaultImpl();
headers.setAuthZHeader(userToken);
headers.setContractName("Default");

Map<String, String> qp = new HashMap<String,String>();
qp.put("resource", "http://hostcomputer.example.com:18001/index.html");

qp.put("action", "get");
qp.put(ClientConstants.IDAAS_REST_SUBJECT_TYPE_QUERY_PARAM_NAME, "TOKEN");
qp.put(ClientConstants.IDAAS_REST_SUBJECT_VALUE_QUERY_PARAM_NAME, userToken);
AuthorizationDecision ad = authZClient.getAuthzDecision (qp, headers);
System.out.println("AuthZ Decision from REST Service : " + ad.getAllowed());
```

9

Developing Mobile Services Applications with the iOS Client SDK

This chapter describes how to develop Mobile Services applications with the iOS Client SDK. This SDK serves as a Security Layer for developing secure mobile applications on iOS. Every native iOS app must implement this SDK to use Mobile and Social. This chapter provides the following sections:

9.1 Getting Started With the iOS Client SDK

This SDK (`oamms_sdk_for_ios.zip`) is included in the Oracle Access Management distribution package and can also be downloaded from the Oracle Technical Network (OTN) website.

In addition to this *Developer's Guide*, API documentation in HTML format is provided in the SDK. Refer to the API documentation for descriptions of API classes, interfaces, constructors, methods, and fields.

The IDM Mobile iOS Client SDK is provided as a static library. It contains three modules:

- **Authentication Module** - Processes authentication requests on behalf of users, devices, and applications.
- **User Role Module** - Provides User Profile Services that allow users and applications to get User and Group details from a configured Identity store.
- **Secure Storage Module** - Provides APIs to store and retrieve sensitive data using the iOS Keychain feature.

Note: You must have the Xcode IDE (integrated development environment) installed on an Intel-based Mac running Mac OS X Snow Leopard or later to develop applications for iOS mobile devices.

For more information, see the iOS Dev Center website:

https://developer.apple.com/devcenter/ios/index.action

9.1.1 Getting Started Using the iOS Client SDK With XCode

Follow these steps to set up your XCode environment.

1. Add `libIDMMobileSDK.a` to XCode by following these steps:
   a. Download `libIDMMobileSDK.a` to your development environment and add it to a project folder.
   b. Launch XCode and open your project.
   c. Click your project to select it, then click your target and click the **Build Phases** tab.
   d. Expand **Link Binary With Libraries** and click the **+** button.
   e. Select **Other** and choose `libIDMMobileSDK.a`.
2. Download and unzip the `PublicHeaders.zip` and `PublicResources.zip` files.

   The `PublicHeaders.zip` archive contains the IDM Mobile SDK header files

   The `PublicResources.zip` archive contains the IDM Mobile SDK resource bundle.
3. Choose **Add Files to *Your-Project-Name*** to add the contents of `PublicHeaders.zip` and PublicResources.zip to your project.

   You can now start coding using the IDM Mobile iOS Client SDK.

Important: Before linking your project, add as a single line both the `-ObjC` and `-all_load` linker flags to your project. Without these flags your application will crash with a "selector not recognized" runtime exception.

Because `libIDMMobileSDK` extends pre-existing classes with categories, the linker does not know how to associate the object code of the core class implementation with the category implementation. This prevents objects created in the resulting application from responding to a selector that is defined in the category.

For background information and steps that describe how to add flags to your project, see the following page:

http://developer.apple.com/library/mac/#qa/qa1490/

9.2 Invoking Authentication Services With the iOS Client SDK

This section provides sample code that demonstrates how to authenticate with the Mobile and Social server.

The sample code in this section supports the "JWTAuthentication" (JSON Web Token Authentication) service type. Refer to "Configuring Mobile Services" in *Oracle Fusion*

Middleware Administrator's Guide for Oracle Access Management for information about configuring a service provider.

Step 1: Initialize Required Objects and Declare the Endpoints

Create the `OMMobileSecurityService` class as follows and initialize it by providing the required Mobile and Social server details:

```
OMMobileSecurityService *mss = [[OMMoblieSecurityService alloc] initWithURL: mobileSocialServerURL]
                                                                    appName: applicationName
                                                                     domain: domainName
                                                                   delegate: self];
```

The `initWithURL` argument is the URL (including protocol, host name, and port number) required to reach the Mobile and Social server. Only the HTTP and HTTPS protocols are supported.

The `appName` argument is a unique identifier that identifies the application. This String value must match the application "Name" value located in the Application Profile section of the Mobile and Social server administration console. For more information, see "Editing or Creating Application Profiles" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

The `domain` argument is the name of the Internet domain within which the Mobile and Social server is located.

The `delegate` argument should be set to `self`.

Next, call the `OMMobileSecurityService setup` method.

```
[mss setup];
```

The setup method gets the configured security policies and the application profile from the Mobile and Social server. This method also gets a list of the service endpoints (the URLs) that are required for connecting to the authentication, authorization, and user profile services on the Mobile and Social server.

The setup call is an asynchronous call and the iOS Client SDK calls the following method for the specified delegate:

```
didReceiveApplicationProfile: (NSDictionary *)applicationProfile error: (NSError
*)error
```

This method returns an `OMRegistrationService` object that handles the client registration.

Note: The thread that calls `[mss setup]` must have a run loop running. If you invoke `[mss setup]` from a thread other than the main thread, ensure that a run loop is running in default mode.

For more information, see the iOS Developer Library Threading Programming Guide:

http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/Multithreading/RunLoopManagement/RunLoopManagement.html

---

Step 2: Complete the Authentication Process

The `OracleMobileSecurityService` object maintains all of the details regarding your current session. Upon receiving the application profile and the URLs needed to connect to the services, start the authentication process. Call the following methods.

```
NSError *error = nil;
//startAuthenticationProcess API triggers the authentication process.
error = [self.mss startAuthenticationProcess:nil
presenterViewController:loginController];
if (error)
{
   UIAlertView* alertView = [[UIAlertView alloc] initWithTitle:@"Authentication process failed"
   message:[error localizedDescription]
   delegate:self
   cancelButtonTitle:@"OK"
   otherButtonTitles:nil];
   [alertView show];
   [alertView release];
   return;
}
```

The last line starts the authentication process and the iOS Client SDK interacts with the Mobile and Social server to complete the authentication process. If the user is already authenticated and the authentication token is still valid, the Mobile and Social server simply returns the cached token. Otherwise, the server prompts the user to provide login credentials. If the Mobile and Social server is configured to use Knowledge Based Authentication, the iOS Client SDK automatically handles the details.

Next, the iOS Client SDK calls your delegate's `didFinishAuthentication: error:` method. The method returns `OMAuthenticationContext`, which has your token details.

Use the `OMMobileSecurityService` object's `[mobileSecurityService authenticationContext]` method to retrieve `OMAuthenticatonContext` at any time. For details about `OMAuthenticationContext`, see the API documentation.

```
- (void)didFinishAuthentication:(OMAuthenticationContext *)context error:(NSError *)error
{
if (context == nil || error != nil)
{
   NSString *msg = [[NSString alloc] initWithFormat:@"%@-%d: %@",
   [error domain],
   [error code], [error localizedDescription]];
   UIAlertView* alertView = [[UIAlertView alloc] initWithTitle:@"Err" message:msg delegate:self
                            cancelButtonTitle:@"OK" otherButtonTitles:nil];
   [alertView show];
   [msg release];
   [alertView release];
   return;
}
 // If successful, proceed with your remaining actions.
 // This example gets the authenticated user's attributes
 // and presents it using User Profile Viewer.

 OMUserRoleProfileService* ups = [mss userRoleProfileService];
 OMUserManager *um = [ups getuserManager];
 OMUser *user = [um searchUser:context.userName attributes:nil shouldPreFetch:NO error:&error];
 self.user = user;
 [detailPaneController showProfileButton];
```

At this point your application can use the token obtained from the Mobile and Social server to make additional web service calls.

Note: The iOS Client SDK does not consume a significant amount of memory. The SDK stores registration handles, authentication handles, application profiles, and security profiles. If iOS sends a low memory warning notification, the SDK persists its cache and releases its memory. When required, the SDK can read persisted data either from a file or from `KeyChainItem` (covered later), as appropriate.

9.3 Invoking HTTP Basic Authentication With the iOS Client SDK

This section provides sample code that show to implement HTTP Basic Authentication. This authentication model does not require the Mobile and Social server.

The HTTP Basic authentication scheme supports the following functionality:

- Online and offline authentication
- Credential storage for offline authentication and retrieval
- Idle time-out and session time-out
- Maximum number of allowed failure attempts for offline authentication after which online authentication is enforced
- Retrieval of `HTTPCookie` after authentication (for example, `OAM_ID` and `ObSSSOCookie`)
- Logout clears the cookie and any cached information in the IDM Mobile iOS Client SDK

Sample code for HTTP Basic authentication is provided here:

```
OMMobileSecurityConfiguration *conf = [[OMMobileSecurityConfiguration alloc]
    initWithApplicationID:@"TestApp"
    maxFailureAttempts:3
    authenticationScheme:OM_HTTP_BASIC_AUTH_SCHEME];

/* Initialize OMMobileSecurityService by passing in ADFMobileSecurityConfiguration */
OMMobileSecurityService *service =
[[OMMobileSecurityService alloc]initWithAppProfileConfiguration:conf delegate:self];

/* Create an authentication request object that will hold all the inputs for one particular
session*/
OMAuthenticationRequest *request = [[[OMAuthenticationRequest alloc]init]];
    [request setLogoutURL:@"http://hostName.example.com:14100/oam/server/logout"];
    [request setSessionExpiryInSecs:30];
    [request setIdleTimeInSecs:12];
    [request setIsOnlineMode:true];
    NSMutableArray *array = [[[NSMutableArray alloc]init];
    [array addObject:@"OAM_ID"];
    [request setRequiredTokens:array];
    [request setAuthenticationURL:@"http://hostName.example.com:7777/index.html"];
    [request setAuthenticationScheme:OM_HTTP_BASIC_AUTH_SCHEME];
     NSError *error = nil;
    UINavigationController *viewController = [service getAuthenticationViewController:request
error:nil];
    if (viewController != nil)
       [self presentModalViewController:viewController animated:true];
    else if (error != nil)
    {
        UIAlertView *alertView = [[UIAlertView alloc]initWithTitle:@"Could not authenticate"
```

```
                                                          message:[error localizedDescription]
                                                         delegate:self
                                                cancelButtonTitle:@"OK"
                                                otherButtonTitles:nil];
        [alertView show];
        [alertView release];
    }
```

After the authentication process completes, the code calls the delegate's `didFinishAuthentication: error:` method.

9.4 Invoking User Profile Services With the iOS Client SDK

Before working with the code samples in this section, see "Building Applications With User Profile Services" for notes and information that are not specific to this SDK.

The code samples in this section are organized into the following three categories:

- Working With People
- Working With Groups
- Working With Organizations

9.4.1 Working With People

To search and retrieve user details, get the handle of `OMUserManager` from the `OMMobileSecurityService` object. See "Invoking Authentication Services With the iOS Client SDK" for information about the `OMMobileSecurityService` object.

`OMUserManager` provides synchronous and asynchronous APIs to search and get user details.

All asynchronous operations return an `OMAsyncOpHandle` object. You can use this object to cancel the operation before it completes. Cancelling an operation after it completes has no effect on it.

```
- (NSArray *)searchUsersWithFilter: (NSDictionary *)filter
                                       isSimpleSearch: (BOOL)simpleSearch
                                attributesToBeFetched: (NSArray *)attributesToFetch
                                            pageSize: (NSInteger)pageSize
                                        pagePosition: (NSInteger)pagePosition
                                               error: (NSError **)error;

- (OMAsyncOpHandle *)searchUsersAsynchronouslyWithFilter:(NSDictionary *)filter
                                       isSimpleSearch:(BOOL)simpleSearch
                                attributesToBeFetched:(NSArray *)attributesToFetch
                                            pageSize:(NSInteger)pageSize
                                        pagePosition:(NSInteger)pagePosition;

- (OMUser *)searchUser: (NSString *)user attributes: (NSArray *)attributes
                                 shouldPreFetch: (BOOL)preFetch
                                          error: (NSError **)error;
- (OMAsyncOpHandle *)searchUserAsynchronously: (NSString *)user
                                   attributes: (NSArray *)attributes
                               shouldPreFetch: (BOOL)preFetch
                                        error: (NSError **)error;

- (OMAsyncOpHandle *)searchAsynchronouslyUser: (NSString*)user
                                   attributes: (NSArray *)attributes
                               shouldPreFetch: (BOOL)preFetch;
```

```
- (NSError *)deleteUser: (NSString *)userName;

- (OMAsyncOpHandle *)deleteAsynchronouslyUser:(NSString*)userName;

- (NSError *)createUserWithAttributes: (NSDictionary *)attributes;

- (OMAsyncOpHandle *)createUserAsynchronouslyWithAttributes:(NSDictionary *)attributes;

- (OMAsyncOpHandle *)modifyAsynchronouslyUser: (NSString*)user
                                  attributes: (NSDictionary *)attributes;
```

9.4.2 Working With Groups

To search and retrieve group details, get the handle of `OMRoleManager` from `OMMobileSecurityService`. See "Invoking Authentication Services With the iOS Client SDK" for information about the `OMMobileSecurityService` object.

`OMRoleManager` provides synchronous and asynchronous APIs to search for groups, add members to groups, and delete members from groups.

All asynchronous operations return an `OMAsyncOpHandle` object. You can use this object to cancel the operation anytime before it completes. Cancelling an operation after it completes has no effect on it.

```
- (OMRole *)getRoleByName: (NSString *)roleName
                   error: (NSError **)error;

- (OMAsyncOpHandle *)getAsynchronouslyRoleByName: (NSString *)roleName;

- (NSError *)deleteRoleByName: (NSString *)roleName;

- (OMAsyncOpHandle *)deleteAsynchronouslyRoleByName:(NSString*)name;

- (OMUser *)getUserInfo: (NSString *)userName fromRole: (NSString *)roleName
              error: (NSError **)error;

- (OMAsyncOpHandle *)getAsynchronouslyUserInfo:(NSString *)user
                                    fromRole:(NSString *)roleName;

- (NSError *)deleteMember: (NSString *)memberName
                fromRole: (NSString *)roleName;

- (OMAsyncOpHandle *)deleteAsynchronouslyMember:(NSString *)memberName
                                     fromRole:(NSString*)roleName;

- (OMAsyncOpHandle *)createAsynchronouslyRoleWithAttributes:(NSArray*)attributes
                                                 withValues:(NSArray*)values;

- (OMAsyncOpHandle *)modifyAsynchronouslyRole:(NSString*)role
                                   attributes:(NSArray*)attributes
                                       values:(NSArray*)values;

- (OMAsyncOpHandle *)addUserAsynchronouslyToRole:(NSString *)roleName
                                  withAttributes:(NSArray*)attributes
                                      withValues:(NSArray*)values;
```

9.4.3 Working With Organizations

Use the following APIs to request information about managers and their reports.

Get a User's Manager

The following APIs are available in `OMUser`.

```
- (OMUser *)getManager: (NSError **)error;
- (OMAsyncOpHandle *)getManagerAsynchronously;
```

Get a Given User's Reports

The following APIs are available in `OMUser`.

```
- (NSArray *)getReporteesWithAttributes: (NSArray *)attributes returningError: (NSError **)error;
- (OMAsyncOpHandle *)getReporteesAsynchronouslyWithAttributes:(NSArray *)attributes;
```

9.5 Using the Credential Store Service (KeyChain)

The Credential Store service provides APIs to store and retrieve sensitive data using iOS Keychain Services.

Start with the `OMMobileSecurityService` object and get an `OMCredentialStore` handle. Use `OMCredentialStore` to write to and retrieve sensitive data from `KeyChainItem`.

The following code snippets illustrate how to use `OMCredentialStore`.

Add a User Name and Password

This example adds a user name and password to a given key in `KeyChainItem`.

```
- (void)addCredential:(NSString *)userName pwd:(NSString *)password url:(NSString *)key;
```

Add a User Name, Password, and Tenant Name

This is a variation of the previous `addCredential` function.

```
- (void)addCredential:(NSString *)userName
                 pwd:(NSString *)password
          tenantName:(NSString *)tenantName
                 url:(NSString *)key;
```

Delete a Credential

This example deletes the credential from KeyChainItem. Because there is not a true delete operation, the user name and password are instead set to null.

```
- (void)deleteCredential:(NSString*)key;
```

Update a User Name and Password

This example updates the user name and password given the user and key values. Because there is not a true update operation, `updateCredential` calls `addCredential`.

```
- (void)updateCredential:(NSString*)userName pwd:(NSString*)password  url:(NSString*)key;
```

Update a User Name, Password, and Tenant Name

This is a variation of the previous `updateCredential` function.

```
- (void)updateCredential:(NSString *)userName
                     pwd:(NSString *)password
              tenantName:(NSString *)tenantName
                     url:(NSString *)key;
```

Get a User Name and Password

This example retrieves the user name, password, and tenant name for a given key.

```
- (OMCredential *)getCredential:(NSString*)key;
```

Store a Property in KeyChainItem

This example stores a property in `KeyChainItem`.

```
- (void)storeProperty: (NSString *)property withKey: (NSString *)key;
```

Store Multiple Properties in KeyChainItem

This is a variation of the previous `storeProperty` function.

```
- (void)storeProperty: (NSString *)property
              withKey: (NSString *)key
            withLabel: (NSString *)label
      withDescription: (NSString *)description;
```

9.6 Invoking the Mobile Single Sign-on Agent App

This section describes how to use the iOS Client SDK to interact with the mobile single sign-on agent app. For conceptual information about mobile single sign-on in Mobile and Social, see the "Introducing Mobile Single Sign-on (SSO) Capabilities" and "Understanding Mobile and Social" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

9.6.1 Invoking the Mobile Single Sign-on Agent App From a Web Browser

Web apps can also use the single sign-on authentication features provided by the mobile SSO agent. This functionality requires Access Manager.

1. Log on to the Oracle Access Management Administration Console and click the **Policy Configuration** tab.
2. Under **Shared Components** click **Authentication Schemes**, then click the Create button.

   The "Create Authentication Scheme" tab opens.
3. Create a new Authentication Scheme by completing the form as follows:
   - **Name:** MobileSSOScheme
   - **Authentication Level:** 2
   - **Challenge Method:** FORM
   - **Challenge Redirect URL:** `/oam/server/`
   - **Authentication Module:** LDAP
   - **Challenge URL:** `/mobilesso?serviceDomain=`*MobileServiceDomain*

     where *MobileServiceDomain* is the name of the domain that is configured for single sign-on.
   - **Context Type:** customWar
   - **Context Value:** `/oic_rest`
4. In the Oracle Access Management Administration Console, do the following:
   a. Create a new Authentication Scheme in an Application Domain:

      Authentication Scheme: MobileSSOScheme

      (*MobileSSOScheme* is the scheme that was created in step one.)

   b. Create an HTTP Resource, for example `/mobileapp`, and protect the resource using the created Authentication Scheme (MobileSSOScheme). This is the URI

that will be accessed from the mobile web browser (mobile Safari for iOS) and protected by a Webgate.

9.7 Invoking Webgate Protected Resources

You can use the Mobile and Social SDK to authenticate against Access Manager using the Mobile and Social service. After authenticating against Access Manager, the SDK gets a token and persists it in the cookie store so that any Access Manager protected app can use the embedded web browser. Access Manager protected REST web services, however, cannot be accessed using the web browser.

The Mobile and Social SDK provides the `OMRESTRequest` class to access REST web services protected by Access Manager. First, use the SDK to authenticate against the OAM server using Mobile and Social services.

Next, initialize the `OMRESTRequest` object by passing a `OMMobileSecurityService` object and a delegate object. You can use either of the following methods:

```
executeRESTRequest: convertDataToJSON: isJsonRepresentation returningResponse: error:
```

- or -

```
executeRESTRequestAsynchronously: convertDataToJSON:
```

The former is a synchronous call and the latter is an asynchronous call. The asynchronous call returns the result through the following `OMRESTRequestDelegate` method:

```
didFinishExecutingRESTRequest: data: urlResponse: error: asyncHandle:
```

The following example demonstrates the asynchronous API of the `OMRESTRequest` object.

```
- (void)someMethod
{
    OMMobileSecurityService *mss = ...;
    ...
    //Initialize OMRESTRequest object. In this example, instead of using
    //"initWithMobileSecurityService: delegate:" method, we use init method
    //and set the properties
    OMRESTRequest *restReq = [[OMRESTRequest alloc] init];
    restReq.delegate = self;
    restReq.mobileService = mss;

    NSURL *url = [[NSURL alloc] initWithString:@"http://myresturl.example.com/resturl"];
    NSMutableDictionary *dictionary = [[NSMutableDictionary alloc] initWithCapacity:1];

    //It is important to set the User-Agent to the value configured in OAM 11g R2
    //Webgate user defined parameters.
    [dictionary setObject:@"OAMMS-Agent" forKey:@"User-Agent"];
    NSMutableURLRequest *urlRequest = [[NSMutableURLRequest alloc] initWithURL:url];
    [urlRequest setAllHTTPHeaderFields:dictionary];
    [url release];
    [dictionary release];
    [urlRequest setHTTPMethod:@"GET"];
    OMAsyncOpHandle *opHandle = [restReq executeRESTRequestAsynchronously:urlRequest
                                                        convertDataToJSON:FALSE];
    [urlRequest release];
    OMLog(@"%@", opHandle);
}
```

```
-(void) didFinishExecutingRESTRequest:(OMRESTRequest *)RESTRequest
                                 data:(id)data
                          urlResponse:(NSURLResponse *)urlResponse
                                error:(NSError *)error
                          asyncHandle:(OMAsyncOpHandle *)handle
{
    if (error)
    {
        //In case of error, show the error message
        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"REST Request Error"
                                                            message:[error localizedDescription]
                                                           delegate:self
                                                  cancelButtonTitle:@"OK"
                                                  otherButtonTitles:nil];
        [alertView show];
        [alertView release];
    }
    else
    {
        //Show the result in the UIAlertView
        NSString *disp = nil;
        if ([data isKindOfClass:[NSDictionary class]])
        {
            NSDictionary *dict = (NSDictionary *)data;
            disp = [[dict OMJSONRepresentation] retain];
        }
        else
        {
            disp = [[NSString alloc] initWithData:data
                                         encoding:NSASCIIStringEncoding];
        }
        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"Received"
                                                            message:disp
                                                           delegate:self
                                                  cancelButtonTitle:@"OK"
                                                  otherButtonTitles:nil];
        [disp release];
        [alertView show];
        [alertView release];
    }
}
```

Note: `OMRESTRequest` can obtain the required token from Access Manager only if the REST web service is protected using an Oracle Access Management 11g R2 Webgate.

The user defined parameter of the Access Management 11g R2 Webgate must contain the `User-Agent` property. The same `User-Agent` property value must be specified by the mobile application in its header.

`OMRESTRequest` can be initialized without `OMMobileSecurityService`, as well. In such cases, the `OMRESTRequest` APIs will just return the URL values.

9.7.1 Understanding the OMRESTRequest API Flow

The following steps describe the internal flow of the `OMRESTRequest` API:

1. The `OMRESTRequest` API invokes the URL provided by the mobile application.

2. The Oracle Access Management 11g R2 Webgate returns a 401 error with the following details:

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: OAM-Auth realm="<WebGateName>:<AuthenticationLevel>
<RelativeRESTURL>", request-ctx="<RequestContext>"
```

3. The Mobile and Social SDK maintains a cache of Access Tokens that it has obtained during the application's lifetime. If the Access Token for this Webgate is already present in the cache, the SDK injects the Access Token into the application request.

4. If an Access Token for the Webgate is not available in the Mobile and Social SDK cache, it sends a REST request to the Mobile and Social server to obtain the Access Token for the Webgate.

5. If the request is valid, Mobile and Social returns an Access Token in the response.

6. The Mobile and Social SDK injects the token returned by the Mobile and Social server.

9.8 Using the iOS SDK to Create a Custom Mobile Single Sign-on Agent App

This section contains information to get you started creating a mobile single sign-on app.

To serve as a mobile single sign-on agent, the app must include logic to handle authentication requests coming from other apps (the *mobile SSO clients*). Mobile and Social provides a sample SSO agent app that illustrates the required logic. You can adapt this logic to any business app and enable the app to function as a mobile SSO agent. To get started, open `OICSSOAPP.zip`.

Note that the app delegate of the iOS mobile SSO app should implement the `openURL` method to handle SSO requests coming from other apps. The URL scheme should also be defined in the iOS app and on the Mobile and Social server. Finally, when adding the Application Profile to a Service Domain on the Mobile and Social server, configure the Mobile Single Sign-on (SSO) Configuration attributes (*Participate in Single Sign-on* and *Agent Priority*).

Note: For information about configuring iOS specific settings on the Mobile and Social server, see the following topics in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*:

- See "Editing or Creating Application Profiles" for information about specific iOS application settings.
- See "Editing or Creating the Service Domain" for information about configuring an SSO-enabled application as either a mobile SSO agent or a mobile SSO client.

10

Developing Applications Using the Internet Identity Services Client SDK

This chapter describes how to use the Internet Identity Services Client SDK to integrate Mobile and Social with supported web and mobile applications. This chapter includes the following topics:

10.1 Before you Begin

Before reading this chapter you should read "Understanding Mobile and Social" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*. This Developer's Guide assumes that you are already familiar with and understand Mobile and Social terminology and concepts.

10.2 Introduction to Developing Internet Identity Services Applications

This section covers concepts and requirements that apply to Internet Identity Services application development in general.

Internet Identity Services supports the following integration scenarios:

- Integrating with a website running on a Java-based application server
- Integrating with a web application that uses Oracle Access Management services, such as Access Manager and SSO services
- Integrating with an application that runs on iOS mobile devices

Mobile and Social features a prebuilt login page for Internet Identity Services. This page supports local authentication so that users with existing accounts can log in and it provides Internet Identity Provider support so that new or existing users can authenticate using an Internet Identity Provider, such as Yahoo, Google, Facebook, LinkedIn, and Twitter. You can use the prebuilt login page for both local user authentication and Internet Identity Provider authentication, or you can choose to use the Mobile and Social login page for Internet Identity provider authentication only,

while keeping the web application's local user authentication mechanism in place. The look and feel of the prebuilt login page can be customized as needed.

To facilitate the creation of end-user accounts, end-users who authenticate using an Internet Identity Provider can be prompted to create a local account. Mobile and Social retrieves the end-user's profile from the Identity Provider, and built-in user registration functionality will pre-populate the user's data.

Note: OAuth providers such as Facebook, Twitter, and LinkedIn require that applications register each Mobile and Social instance to get the consumer key and secret values.

10.2.1 About the Internet Identity Services Client SDK

The primary Java package that you will use when working with the Internet Identity Services Client SDK is the `oracle.security.idaas.rp.client` package. (The "rp" stands for *relying party*.)

In addition, the following libraries are required to be available during the compilation and execution phases. These libraries must also be available in the class path of the application server when the client code is included in a web application that may be compiled at run time by the web container. These libraries are provided in the product package, `oamms_sdk_for_java.zip`.

The following libraries are included in `oamms_sdk_for_java.zip`, as well as license information and API documentation.

Mobile and Social Libraries

- `oic_clientsdk.jar`
- `oic_common.jar`
- `oic_sae.jar`
- `ojdl.jar`

Third-Party Archives

- `jersey-archive-1.9.1`

10.3 Getting the List of Identity Providers for an Application

The `RPClient` class located in the `oracle.security.idaas.rp.client` package is required to get the list of configured Internet Identity Providers for an application.

The `RPClient` class takes two parameters: `applicationID` and `properties`.

The first parameter, `applicationID`, is a unique identifier that identifies the application. This String value must match the application "Name" value located in the Application Profile section of the Mobile and Social server administration console. For more information, see the "Configuring Internet Identity Services" chapter in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

The second parameter is a URI that maps to a properties file. This properties file lists the configuration properties that the application and the Mobile and Social server require to connect and securely exchange data. The properties file must be in a location that is accessible to the application at run time.

The following table lists the required and optional configuration properties that the Mobile and Social server accepts. All properties are Strings and, unless otherwise noted, are optional.

Table 10–1 Configuration Properties Required by the RPClient Class

Property Name	Required	Description	Comment
`rp.server.hosturl`	Required	The URL (including protocol, host name, and port number) required to reach the Mobile and Social server. Only the HTTP and HTTPS protocols are supported.	
`rp.server.idp.service`	Required	The relative path required to reach the identity providers service.	The current service path is: `/oic_rp/rest/identityproviders` This path is appended to the `rp.server.hosturl` property.
`rp.server.init.service`	Required	The relative path required to reach the Relying Party (RP) service.	The current service path is: `/oic_rp/RPInitServlet`
`rp.server.connection.timeout`		The duration in milliseconds after which the connection is interrupted if the server stops responding. Null or empty means infinite (no time-out).	
`rp.server.connection.sae.sharedsecret`	Required	The secret used to secure communication with the server.	Base64 encoded String.
`rp.server.connection.sae.algorithm`		The algorithm used to secure communication with the server. Defaults to `SAE` if omitted.	Supported values include: **`SAE`** - Secured Attribute Exchange. **`DES`** - Data Encryption Standard.
`rp.server.connection.sae.keystrength`		The key length used to encrypt the key. Defaults to `128` if omitted.	
`rp.server.connection.sae.cryptotype`		The type of cryptography. Defaults to symmetric if omitted.	
`rp.server.connection.sae.keystorefile`		The file name containing the encryption keys.	
`rp.server.connection.sae.keystoretype`		Determines the type of keystore.	
`rp.server.connection.sae.keystorepass`		The keystore password.	
`rp.server.connection.sae.privatekeyalias`		The private key alias.	
`rp.server.connection.sae.publickeyalias`		The public key alias.	

Table 10–1 (Cont.) Configuration Properties Required by the RPClient Class

Property Name	Required	Description	Comment
`rp.server.connection.sae.privatekeypass`		The private key password.	
`rp.server.connection.sae.certclass`		The class name implementing the Cert interface.	
`proxy.protocol`		The protocol to use with a proxy server.	Supported values include: **http** **socks** **direct**
`proxy.host`		The host name of the proxy server.	
`proxy.port`		The proxy server port number.	
`proxy.username`		The user name required to authenticate with the proxy server.	
`proxy.password`		The password required to authenticate with the proxy server.	

The following sample code consists of a single class that outputs the available Internet Identity Providers and their corresponding URLs. You can use this code to verify an Internet Identity Service platform configuration. Simply provide the required `applicationId` input parameter.

First, import the following class dependencies:

```
// Java imports
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.net.MalformedURLException;

// Mobile and Social imports
import oracle.security.idaas.rp.client.RPClient;
import oracle.security.idaas.rp.client.RPClientConfigUtil;
import oracle.security.idaas.rp.client.RPClientException;
```

Next define the following constants:

```
public class SampleOicClient {
    // Define the default properties file name
    private static final String PROP_FILE_NAME = "SampleOicClient.properties";
    // Pre-define the Client SDK properties
    private final static String PEOPLE_SERVICE = "/oic_rest/rest/userprofileservice/people";
    private final static String TOKEN_SERVICE = "/oic_rest/rest/tokenservice1/tokens";
```

The `createPropertiesFile()` function creates a default properties file if a local file is not found. You still need to provide the required values, however. Refer to Table 10–1 for details.

```
    private static void createPropertiesFile() {
        try {
```

```
            // Create file
            FileWriter fstream = new FileWriter(PROP_FILE_NAME);
            BufferedWriter out = new BufferedWriter(fstream);
            out.write("");
            out.write("rp.server.hosturl=http://hostcomputer.example.com:18001\n");
            out.write("rp.server.idp.service=/oic_rp/rest/identityproviders\n");
            out.write("rp.server.init.service=/oic_rp/RPInitServlet\n");

            out.write("rp.server.connection.timeout=60000\n");
            out.write("rp.server.connection.sae.sharedsecret=sharedSecret1\n");
            out.write("rp.server.connection.sae.algorithm=AES\n");
            out.write("rp.server.connection.sae.keystrength=128\n");
            out.write("rp.server.connection.sae.cryptotype=symmetric\n");
            out.write("rp.server.connection.sae.keystorefile=\n");
            out.write("rp.server.connection.sae.keystoretype=\n");
            out.write("rp.server.connection.sae.keystorepass=\n");
            out.write("rp.server.connection.sae.privatekeyalias=\n");
            out.write("rp.server.connection.sae.publickeyalias=\n");
            out.write("rp.server.connection.sae.privatekeypass=\n");
            out.write("rp.server.connection.sae.sigvalidityduration=\n");
            out.write("rp.server.connection.sae.certclass=\n");

            out.write("#proxy configuration\n");
            out.write("proxy.host=\n");
            out.write("proxy.port=\n");
            out.write("#http|socks|direct\n");
            out.write("proxy.protocol=\n");
            out.write("proxy.username=\n");
            out.write("proxy.password=\n");

            out.close();
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
```

The following code outputs the available Internet Identity Providers based on the required `applicationID` identifier provided.

```
public static void main(String[] args) {
        RPClient client = null;
        int exitStatus = 0;
        String ret = null;
        File prop = new File(PROP_FILE_NAME);
        String applicationName = null;

        //Check the arguments: applicationID is mandatory
        if (args.length < 1 || args[0].isEmpty()) {
            System.err.println("Invalid number of arguments. Specify the name of the application
(the applicationID) to be used to connect to the Mobile and Social Server.\n");
            exitStatus = 1;
        } else {
            applicationName = args[0];

            // Check if a properties file is available
            if (prop.exists()) {
                RPClientConfigUtil conf = null;

                // Read the configuration using the provided utility class
                try {
```

```
                conf = new RPClientConfigUtil(prop.toURI().toURL());
            } catch (MalformedURLException e) {
                System.err.println("Malformed URL:" + e.getMessage());
                exitStatus = 1;
            } catch (IOException ioe) {
                System.err.println("IO Exception:" + ioe.getMessage());
                exitStatus = 1;
            }
            System.out.println("RPClient :\n=========\n");
            try {

                // Initiate the interface with the Mobile and Social Server using
                // the configuration properties and the applicationID.
                client = new RPClient(applicationName, conf);

                ret = "The application name is [" + applicationName + "]\n";
                ret += "The retrieved IDP information is:\n\n";
                for (String idp : client.getIDPList()) {
                    // Display the IDP name
                ret += "    IDP name : " + idp + "\n";
                    // DISPLAY the IDP reference URL
                ret += "    IDP Href : " + client.getHrefByIdpName(idp);
                    ret += "\n";
                }
            } catch (RPClientException rpce) {
                System.err.println("Client Exception:" + rpce.getMessage());
                exitStatus = 1;
            }
            System.out.println(ret);
            System.out.println("\nClient SDK :\n============\n");
            System.out.println("    CreateToken :\n    ============\n");
            new CreateToken(conf.get("rp.server.hosturl") + TOKEN_SERVICE);
            System.out.println("\n    People :\n    ========\n");
            new CreateUser(conf.get("rp.server.hosturl") + PEOPLE_SERVICE);
        } else {
            //No properties file is available, so create a default one
            createPropertiesFile();
            System.out.println("The " + PROP_FILE_NAME + " properties file has not been found.
 A default one has been created at this location.");
            System.out.println("Please edit the file and provide the required values. Then
 restart this utility.\n");
            exitStatus = 2;
        }
    }
    System.exit(exitStatus);
  }
}
```

10.4 Integrating Internet Identity Services With a Web Application Running on a Server

The Internet Identity Services SDK supports web applications, such as portal sites and consumer-driven web sites that run on Java-compliant application servers.

To integrate Internet Identity Services with a web application, first define the web application on the Mobile and Social server, then integrate the Internet Identity Services login page with the web application. Next, configure User Registration (optional) and handle the final return response.

This section covers the following topics:

- Defining the Web Application on the Mobile and Social Server
- Integrating the Internet Identity Services Login Page With the Web Application
- Handling User Registration
- Handling the Final Return Response

10.4.1 Defining the Web Application on the Mobile and Social Server

Use the Mobile and Social system administration console to define the web application on the Mobile and Social server. See "Editing or Creating Application Profiles" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management* for help completing this task.

Following is a brief description of some of the items that you need to configure on the Mobile and Social server:

- **Application Name** - Provide the context name of the web application.
- **Application Return URL** - Provide the URL that Mobile and Social should use to send back authentication responses.
- **Shared Secret** - Provide the security secret that the web application and the Mobile and Social server share to facilitate secure communication. The shared secret is also needed during User registration.
- **Required Identity Providers** - Choose the Identity Providers that the end-user can pick from to authenticate to the application.
- **Application User Profile Attribute Mappings** - Map the user profile attributes that the Identity Provider returns to the user profile attributes that are local to the application.
- **User Registration and Registration URL** - Indicate if the system should prompt Users who do not have a local account to register. Provide the URL to which the server should redirect Users after authentication when the Service Provider completes.

10.4.2 Integrating the Internet Identity Services Login Page With the Web Application

To integrate Internet Identity Services, use the Internet Identity Services Client SDK (`oic_clientssdk.jar`) and modify its login page (`login.jsp`).

There are two ways that you can integrate the Internet Identity Services login page with a web application: (1) Add the pre-built login page hosted on the Mobile and Social server to the web application using the HTML `<iframe>` tag, or (2) Build a custom login page using Internet Identity provider data provided by Mobile and Social.

10.4.2.1 Adding the Pre-built Internet Identity Services Login Page

To add the login page hosted on the Mobile and Social server to the web application, first get a secure token using the SDK. The web application needs a Secured Attribute Exchange (SAE) token, which is based on the shared secret that is known to the web application and the Mobile and Social server.

The following sample code shows how to initialize the Internet Identity Service client SDK and get the `saeToken`. This code can be added to a JSP page, for example `login.jsp`.

The `RPClient` class takes two parameters: `applicationID` and `properties`.

```
RPClient rpClient = new RPClient("sampleapp",properties);
Map<String, String> attrs = new HashMap<String, String>();
attrs.put("applicationID","sampleapp");
String saeToken =
rpClient.getSaeToken(attrs,properties.getProperty("rp.server.connection.sae.sharedsecret"),
properties.getProperty("rp.server.connection.sae.sharedsecret"));
```

The `getSaeToken` method gets the SAE token for the application.

The `sae.sharedsecret` property from the properties file makes up the second and third parameters of the `rpClient.getSaeToken` method. (For details, see Section 10.3, "Getting the List of Identity Providers for an Application.") The first instance of the "SAE secret" is used to sign the attributes, and the second instance is used to encrypt them. If the second instance of the "SAE secret" (that is, the third parameter of the method) is null, the attributes are signed but not encrypted.

Next, use an HTML `<iframe>` tag to embed the Internet Identity Service login page:

```
<iframe
src="http://oc.example.com:24666/oic_rp/login.jsp?applicationID=sampleapp&saeToken=<%=saeToken%>"
scrolling="no"
frameBorder="no"
allowtransparency="true"
style="width:720px;height:440px;">
</iframe>
```

The following screen capture shows a sample login screen that has used an iframe to integrate the prebuilt login page hosted on the Mobile and Social server. This page has been configured to support both local user authentication and Internet Identity Provider authentication.

Figure 10–1 Pre-built Login Screen With Local Login Support

The next screen capture shows the same page as the previous example with only Internet Identity Provider support enabled. In this example, the web application would implement its local user authentication mechanism separately.

Figure 10–2 Pre-built Login Screen Without Local Login Support

10.4.2.2 Building a Custom Login Page

If you need greater flexibility building your login page, use the approach outlined in this section.

The code in the following example initializes the Internet Identity Service client SDK, invokes the REST endpoint, gets the identity provider data, then builds the login page using an HTML table to display the Identity Provider logos in table rows.

```
String ret =" ";
RPClient client = null;
try {
     client = new RPClient("sampleportal", "rpclient.properties");
     for (String idp: client.getIDPList()){
        ret += "\n<TR><TD><a href='" + client.getHrefByIdpName(idp)
                + "'><img src='images/" + idp.toLowerCase()
                + ".gif' alt=" + idp + " title=" + idp
                + "border='0'></img></a></TD></TR>";
}
} catch(Exception e){
      e.printStackTrace();
}
```

The output from the Mobile and Social server looks like this.

```
<table cellpadding="6" cellspacing="6" align="center">
<th colspan=2>Sign in with any Account</th>
<TR>

<td>
<a href='http://rp.example.com:24666/oic_rp/init?applicationID=sampleportal
&saeToken=RU5DUllQVEVENjIwODgxM0ZCNTAxOEZENUZBRTA2MzYxOTJBQzM3MDIwQjc5NEE1RDdFMjczREYxNUYw
```

```
Mzk0NjQwRjRFRTQwNzBCMzc0OEMyRUVENTkyOTVCMkI5NUU0MzM2QTk5MzYyRjJCQjg5MDJDNDcwNDlFNTFFMTIzMU
Q5OTY1RTZBMjA3QzM3N0FCNDlBMDlFQjVFQUI2RDlDRTU1RERGOTExNEIyMThFNzBGMjYzRkI3MkRGNEIwMjlENTBFQ
zFEMTM1RkUzRjU5RjcxQkMxQTg2QkNBNzAzQTUwOTBCRUJBOEY3REM5RUU3RjIyQjEwQ0Q5QzNCQjA0RDVDRDBGQUNF
NkM1M0ZGQzJCNDk4NERBRDNGNkI4REY0QkU3QzZCMDU4QTRBREQxNTI4NzdCMTkxRkU4MTdGRTYzNEQ0OTdFN0MxQzk
3M0MzQkFFOEVCQzEwQzg0NDIzMDQ1NDAyNUZCRQ=='><img src='images/facebook.gif' alt=Facebook
title=Facebook border='0'></img></a></td>
<td>
<a href='http://rp.example.com:24666/oic_rp/init?applicationID=sampleportal
&saeToken=RU5DUllQVEVENjIwODgxM0
xNUYwMzk0NjQwRjRFRTQwNzBCMzc0OEMyZCNTAxOEZENUZBRTA2MzYxOTJBQzM3MDIwQjc5NEE1RDdFMjczREYRUVENT
RUVENTkyOTVCMkI5NUU0MzM2QTk5MzYyOTM3REY0NzJFQTIzQTVDNDY4RjRCREJFRTM4OEU2RDI2QUI3QTc4QjE3RUNDO
DY4NDU2MDZCQjk4Q0IyNjg3QTNFMUQzOTVENTM5NjEzRTZDMTM3RjVBNDJFRUZENzUzOERDOEVDQkUzOEY5NEM5QjU2Qz
E4NjVGRTA2MzVCMDBBNUYyNTgzRDU0OEI4ODE4RUI4ODgxMkUyMEM3RDU3RUIwQjMyRTk2RkI3ODc5RkI1MzU5QjhFNDU1
RjZDQzZEQTlEQTVCRDkwQjIxQzEyRDUzNEIzNTMyNTgwRTZCOTM3NzZDM0UyNTg2QTE3MTZFMDc3MTJFNDAxMDI3OTg1Qw==">
<img src='images/twitter.gif' alt=Twitter title=Twitter border='0'></img></a></td>
</TR><TR>
<td>
<a href='http://rp.example.com:24666/oic_rp/init??applicationID=sampleportal
&saeToken=RU5DUllQVEVENjIwODgxM0ZCNTAxOEZENUZBRTA2MzYxOTJBQzM3MDIwQjc5NEE1RDdFMjczREYxNUYwMzk
0NjQwRjRFRTQwNzBCMzc0OEMyRUVENTkyOTVCMkI5NUU0MzM2QTk5MzYyNThGRDM1MjRDRDQ2QTQ5RTRFMEZEMTY3OEEyQ
TAxM0Y4NDQ1MjUwODkxRTlBRDgzMDdBNDRDRjEzMkE4MEJCNkU5ODhCMjcyRkNBNUMwREJFRjA4MDAzQkMwRTAzQzNGNkUy
OEJEMzMxMDcwNTlEMDdGQjREMzFEQjdCRjRDM0YxNUQ4OTI2QTY4OUM1NDIwNjk5MDY3RUM0M0YyNjA0QjBBRDc3MkZGOThB
QjUxRTJDQUFFOEYwOEQ0QTE3Njc4MDM1NjYxNzY5MzY4Mjk0MjVFRDFGNDhEMDAzQTFBMjU1MEQ1QUE1RkM3MkNDMUNGMjUwN0J
CMEI1NDkzM0Q5NQ=='><img src='images/linkedin.gif' alt=LinkedIn title=LinkedIn border='0'></img>
</a></td>
<td>
<a href='http://rp.example.com:24666/oic_rp/init??applicationID=sampleportal
&saeToken=RU5DUllQVEVEMDI5M0FFNjIzMEVEOUZEQkEzMEU0QzAxMDlCOTg1ODlDQ0I5QkQxM0JGQjhGQkY5QzAzNDZER
EZGN0I1RDBBMjQ3NzkxNjdBOERFNTUzN0IxMzk0QTdENUUyNDQxOTdBNkE4MDEwOTlGOEJDNTIyQTQwMEU3OTM3OUUxQTRFR
UYyNjY0MDc3Nzc0OEMzNDJCMzhCOUJDREZBQzdCNENFQkI5NTBCNDRCOTQyQjU5NkYwMEQ2MUY3MUMxNkJEOUIyQzk5MTk1R
DRGNzQ4OTg0QzFDMjFEMzQwOUQ0RUIxQTRFOTZGMTFBN0ExODg3MTZCQTc5QUE2QTU5RDk3MUMzOUQ1MkY1NEM2Q0I1OUFCNz
FBOTQ3M0VBQkE3QzUzQzc0MzM4ODk3RUY3NEJFQkJFODg2QjE4QkU5RUFGQURFRDE3NUMyRTg0NjRDRjNDQzJGN0Y1QTU4RDlD
OTIzMDM5OQ=='><img src='images/google.gif' alt=Google title=Google border='0'></img></a></td>
</TR><TR>
<td>
<a href='http://rp.example.com:24666/oic_rp/init??applicationID=sampleportal
&saeToken=RU5DUllQVEVEMDI5M0FFNjIzMEVEOUZEQkEzMEU0QzAxMDlCOTg1ODlDQ0I5QkQxM0JGQjhGQkY5QzAzNDZEREZGN
0I1RDBBMjQ3NzkxNjdBOERFNTUzN0IxMzk0QTdENUUyNDQxOTdBMDQ5MjMxQUJDMUZDMzVFNjI5NkVBREZBNzk0MUJDQTY2OTE3
RkRCQTUyRUFERTVGNEIyN0NCMzMzMThBQTM2MDlGOTE5NDY4OTBCN0EyNzdGRUY2Qjc1RTBENjY4MDEyNTVGOTA3Mzc5MTZGQkU
1MkM0RkQwMEJFQjAzOUQ4NEQyOEYyNDU4MDRDMjUzQjg2ODlGRkM1MkFGRTEwOTAwMDM2NDExRDIwMzkyMDBBMzYxMkZGRjU4Rj
VNkFBQ0QwQTgyMzA2NDA2REEyMzkxMzU2NjAzNDIzNzE5MDdDM0RFRURGNTE5NjI4MzQ2RTVCNzZCOTg5Q0NFMkM2RTkwMQ==">
<img src='images/yahoo.gif' alt=Yahoo title=Yahoo border='0'></img></a></td></TR>
<!------End Providers------>
</table>
```

10.4.3 Handling User Registration

To facilitate the creation of local end-user accounts, Mobile and Social can prompt users to create a local account. After the User authenticates with an Identity Provider, Mobile and Social can redirect to a custom User registration page or to a built-in registration page that is included with the Mobile and Social Server.

This section covers both approaches:

- Using a Custom User Registration Page
- Using the Mobile and Social Built-in User Registration Page

10.4.3.1 Using a Custom User Registration Page

Use the information in this section to configure a custom User Registration Page for use with Mobile and Social.

Configure the User Registration Properties on the Server

Use the Mobile and Social system administration console to configure the following User Registration properties for the application:

- User Registration (Choose "Enabled")
- Registration URL
- Attributes (Located in the "Application User Attribute" section)
- Attribute Mapping (Located in the "Registration Service Details with Application User Attribute Mapping" section)

For information about each field, see "Editing or Creating Application Profiles" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

Decrypt the saeToken

If **User Registration** is enabled for the application, Mobile and Social redirects to the configured **Registration URL**. Mobile and Social does a POST to the URL and includes two parameters: `saeToken`, which contains the User profile data encrypted with the shared secret, and `Application`, which is the application name.

Use the `RPClient` API in the `oic_clientsdk` to decrypt the `saeToken`.

For example:

```
String saeToken = request.getParameter("saeToken");
Map<String, UserAttribute> regAttrMap = null;
if (saeToken != null) {
    Map<String, String> saeAttrs = client.getAttrFromSaeToken(saeToken,
                                   ”shared secret value”,”shared secret value”);
    System.out.println("register: saeAttrs :" + saeAttrs);
    String regAttrs = saeAttrs.get ("reg_attrs");
    String selectedIDP = saeAttrs.get ("oicInternetIdentityProvider");
    String state = saeAttrs.get ("return_url");
}
```

The following block shows all of the attributes of `saeToken`:

```
saeAttrs :{readonly_fields=uid,mail, password_field=password,
registerReadOnlyToken=RU5DUllQVEVEODk4QjlGQzU0RjNDNTQyNTY0NTU1MzMzOTU3RDU3QjkwNzd
EMTBGREJDMTlBQjM4NDIzNTFFRkI0OUNDQjg1M0JFREQ1NTdCM0I5RkY4MEYyNjFDMEE1NUFBRjhDQjA4
QTQ1M0IyMDg0MzFCNzgxQjg4Nzg4QzJFNEU5MzJFNUM0MjgxMEQxNjEzNkFERjE1MUE0QTMzM0E3MTMyN
zM5NUExN0U3MzA2MjZCRjZDQzgxN0I2MjIwNzEyNEQ2REVE,
reg_attrs=uid:UserId:example@gmail.com:,mail:Email Address::,
timezone:Time Zone::,postaladdress:Country:US:,preferredlanguage:Language:en-US:,
lastname:Last Name:doe:,commonname:First Name:john:,password:Password::,,
username_attr=uid, mandatory_fields=uid,mail,password,password,
state=f166f9aa12edaaeffce703276de2d73c30dbddd0,
oicInternetIdentityProvider=Google,
return_url=http://host.example.com:18001/oic_
rp/popup?state=f166f9aa12edaaeffce703276de2d73c30dbddd0}
```

The application needs to process the value of `reg_attrs`. The other `saeToken` attributes should be ignored.

```
reg_attrs=uid:UserId:example@gmail.com:,mail:Email Address::,
```

```
timezone:Time Zone::,postaladdress:Country:US:,preferredlanguage:Language:en-US:,
lastname:Last Name:doe:,commonname:First Name:john:,password:Password::,,
```

The value is a comma-separated `{User Attribute Name:User Attribute Label:User Attribute Value}` set.

The application can redirect to the Mobile and Social Return URL by appending `oicUserRegister=done` to the URL.

For example:

```
response.sendRedirect(http://oic.host.com:18001/oic_rp/popup?state=f166f9aa12edaaeffce703276de2d73c
30dbddd0&oicUserRegister=done);
```

Mobile and Social creates a User Token based on the Identity Provider authentication and returns it to the application.

10.4.3.2 Using the Mobile and Social Built-in User Registration Page

Use the information in this section to enable the built-in User Registration page. This page is shown in Figure 10–3.

Configure the User Registration Properties on the Server

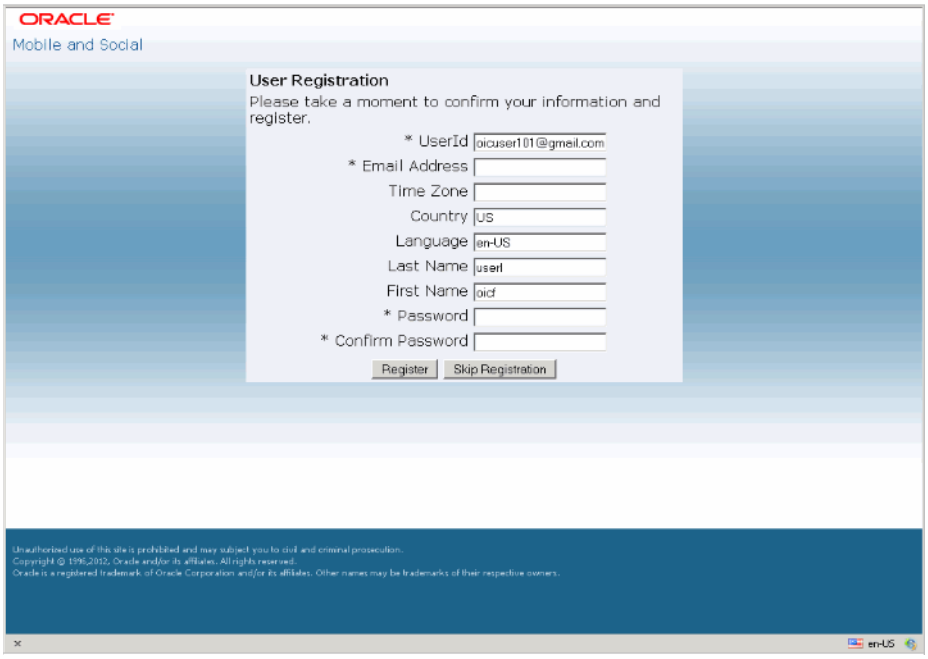
Use the Mobile and Social system administration console to configure the following User Registration properties for the application:

- User Registration (Choose **Enabled**)
- Registration URL (Set to the URL provided with the default Mobile and Social Internet Identity Services application, OAMApplication. For example: `http://host.example.com:port/oic_rp/register.jsp`
- Attributes (Located in the **Application User Attribute** section)
- Attribute Mapping (Located in the **Registration Service Details with Application User Attribute Mapping** section)

For information about each field, see "Editing or Creating Application Profiles" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

If **User Registration** is enabled for the application, Mobile and Social redirects to the built-in Mobile and Social User Registration Page. The User can complete the form and register (required if Access Manager is protecting the resource) or skip registration. Mobile and Social then redirects to the application's configured Return URL.

Figure 10–3 The Mobile and Social Built-In User Registration Page



10.4.4 Handling the Final Return Response

After User authentication—and optionally after User registration—Mobile and Social redirects back to the application.

Depending on whether the user chose to log in locally or to log in using a Identity Provider, the return response is slightly different.

Local Login Return Response

If the User opted to log in locally, Mobile and Social redirects back to the application's Return URL with the `saeToken` parameter. The `saeToken` contains the Mobile and Social generated User Token data, which has been encrypted using the Shared Secret.

Use the `RPClient` API in the `oic_clientsdk` to decrypt the `saeToken`.

For example:

```
String saeToken = request.getParameter("saeToken");
if (saeToken != null) {
    Map<String, String> saeAttrs = client.getAttrFromSaeToken(saeToken,
                                   "shared secret value","shared secret value");
    System.out.println("register: saeAttrs :" + saeAttrs);
    String uid = saeAttrs.get ("uid");
    String authType = saeAttrs.get ("authType");
    String oicUserToken = saeAttrs.get ("oicLocalLoginUserToken ");
}
```

The following block shows the saeToken response attributes:

```
{uid=weblogic, authType=local,
oicLocalLoginUserToken=eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6ImJhc2VfZG9tYW
luIn0.eyJleHAiOjEzMzkwODkzMzAxMTgsImF1ZCI6InJlc3Rfc2VydmVyIiwiaXNzIjoiSW50ZXJuZXR
JZGVudGl0eUF1dGhlbnRpY2F0aW9uIiwicHJuIjoid2VibG9naWMiLCJqdGkiOiIyZTVhZmJhZS03ZTQz
LTQwYmMtODIwZS1mODVhN2MyODE2ZWUiLCJvcmFjbGUub2ljLnRva2VuLnR5cGUiOiJVU0VSVE9LRU4iL
CJpYXQiOjEzMzkwODU3MzAxMTgsIm9yYWNsZS5vaWMudG9rZW4udXNlcl9kbiI6InVpZD13ZWJsb2dpYy
```

```
xvdT1wZW9wbGUsb3U9bXlyZWFsbSxkYz1iYXNlX2RvbWFpbiJ9.UIZuur_e_uWyFzoig
_mBolPXKIz5BnmmszmB65zsAVNISQ9dui0PRd6F85N-0ZBPKQMf8Xnch0c_mjIt6eXwUD6YD_
Wo0f44svmzM43X74Q2VkMgkH2xM1-3lZEMQb03zZ55CrR5UXpsDKvF54bSa6-jbg2ODZAb2ralEqKiVd0}
```

See Section 10.4.4.1, "Secured Attribute Exchange (SAE) Token Response Attributes," for information about response attributes.

Identity Provider Login Return Response

If the User opted to log in using an Identity Provider, Mobile and Social redirects back to the application's Return URL (using HTTP POST). The response includes the `saeToken` parameter, which contains the Mobile and Social generated relying-party User Token data that has been encrypted using the Shared Secret, and the User profile data returned by the Identity Provider.

Use the `RPClient` API in the `oic_clientsdk` to decrypt the `saeToken`.

For example:

```
String saeToken = request.getParameter("saeToken");
if (saeToken != null) {
    Map<String, String> saeAttrs = client.getAttrFromSaeToken(saeToken,
                                   ”shared secret value”,”shared secret value”);
    System.out.println("register: saeAttrs :" + saeAttrs);
    String uid = saeAttrs.get ("uid");
    String authType = saeAttrs.get ("oicInternetIdentityProvider ");
    String oicUserToken = saeAttrs.get ("internet_identity_user_token");
}
```

The following block shows the saeToken response attributes:

```
{uid=example@gmail.com, mail=, timezone=, postaladdress=US,
internet_identity_user_token=eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6ImJhc2dn
VfZG9tYWluIn0.eyJleHAiOjEzMzkwODk4OTM4NjQsIm9yYWNsZS5vaWMudG9rZW4ucnAuaWRwX3VyaSI
6Imh0dHBzOi8vd3d3Lmdvb2dsZS5jb20vYWNjb3VudHMvbzgvaWQiLCJvcmFjbGUub2ljLnRva2VuLnJw
LnVzZXJfaWRfdXJpIjoibWFpbHRvOm9pY3VzZXIxMDFAZ21haWwuY29tIiwiYXVkIjoicmVzdF9zZXJ2Z
XIiLCJpc3MiJJbnRlcm5ldElkZW50aXR5QXV0aGVudGljYXRpb24iLCJwcm4iOiJtYWlsdG86b2ljdXNl
cjEwMUBnbWFpbC5jb20iLCJqdGkiOiI0ZTQxNzhkYy1hOWRhLTQxODMtODVkMi1hNDM0MDM5YmZhNzUiL
CJvcmFjbGUub2ljLnRva2VuLnR5cGUiOiJVU0VSVE9LRU4iLCJpYXQiOjEzMzkwODYyOTM4NjQsIm9yYW
NsZS5vaWMudG9rZW4udXNlcl9kbiI6Im1haWx0bzpvaWN1c2VyMTAxQGdtYWlsLmNvbSBodHRwczovL3d
3dy5nb29nbGUuY29tL2FjY291bnRzL284L2lkIn0.K4d_gewCE6N61Ng8DsB31BIEEG1S1wMHWpt5CC8u
yfdMgtpEjzs77yFYzNz5vS6x6lsa8fH2FrSgOlxoyoiOnRF8UfJgCzk3VJIDNoDd_2BsYrBQVLGQzWTbs
avO5x09macq6rmOOT1Jbg9napfol8plqgTHPZgm3CH7Rt7jmxA, preferredlanguage=en-US,
lastname=doe, commonname=johndoe, oicInternetIdentityProvider=Google, password=}
```

See Section 10.4.4.1, "Secured Attribute Exchange (SAE) Token Response Attributes," for information about response attributes.

10.4.4.1 Secured Attribute Exchange (SAE) Token Response Attributes

The following table describes the `saeToken` response attributes. These attributes can be used in your code as needed.

Table 10–2 Secured Attribute Exchange (SAE) Token Response Attributes

Attribute	Description
`uid`	The User ID of the User who completed the local login on the Mobile and Social hosted login page.
`authType`	The type of login that the User selected. For example, `authType=local`.

Table 10–2 (Cont.) Secured Attribute Exchange (SAE) Token Response Attributes

Attribute	Description
`oicInternetIdentityProvider`	The Identity Provider the User selected. For example, `oicInternetIdentityProvider=Google`.
`oicLocalLoginUserToken`	The Mobile and Social generated User token for Users who login locally. You can use this User token in your application to access User profile and other REST services in Mobile and Social.
`internet_identity_user_token`	The Mobile and Social generated User token for Users who login with an Identity Provider. You can use this User token in your application to access User profile and other REST services in Mobile and Social.

10.5 Integrating With an Access Manager Protected Web Application

You do not have to write code to integrate Internet Identity Services with web applications that are integrated with Access Manager. To complete this integration, use the Mobile and Social and the Access Manager system administration consoles. For instructions, see "Configuring Internet Identity Services" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

10.6 Integrating Internet Identity Services With a Mobile Application

Internet Identity Service provides a mobile-friendly login page if you use a mobile browser to view the login page hosted on the Mobile and Social server. Mobile and Social auto-detects the mobile device and displays the appropriate page. Further configuration is not required.

If you integrate the Internet Identity Services login page in a native mobile app, you can use either the hosted login page or a custom login page that is installed on the device. The code running on the mobile device does not need to know which Identity providers are enabled on the Mobile and Social server. You can add and remove Identity providers on the server without having to update the code that runs on the mobile device.

10.6.1 Defining the Mobile Application on the Mobile and Social Server

Use the Mobile and Social system administration console to define an application profile for the mobile application in Mobile and Social. See "Editing or Creating Application Profiles" *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management* for help completing this task.

Following is a brief description of some of the items that you need to configure on the Mobile and Social server:

- **Application Name** - Provide the name of the application.
- **Mobile Application Return URL** - Provide the mobile application's return URL. Mobile and Social uses this URL to send back authentication responses.
- **Shared Secret** - Provide the security secret that the mobile application and the Mobile and Social server share to facilitate secure communication.
- **Required Identity Providers** - Choose the Identity Providers that the end-user can pick from to authenticate to the application.
- **User Attribute Mappings** - Map the user profile attributes that the Identity Provider returns to the user profile attributes that are local to the application.

11

Extending the Capabilities of the Mobile and Social Server

This chapter discusses how to extend the Mobile and Social Java interfaces to add new authentication Services Providers for Mobile Services. This chapter includes the following topics:

- Section 11.1, "Create a new Authentication Services Provider for Mobile Services"
- Section 11.2, "Create a new Identity Service Provider for Internet Identity Services"

11.1 Create a new Authentication Services Provider for Mobile Services

This section covers the following topics:

- Developing the Custom Authentication Service Provider
- Building the Custom Authentication Service Provider
- Deploying the Custom Authentication Service Provider

11.1.1 Developing the Custom Authentication Service Provider

To create a custom authentication Service Provider you need to write two custom classes:

- **TokenService** - Implement this interface first. This is a basic custom token provider that works with non-mobile applications.
- **MobileCompositeTokenServiceProvider** - Extend this class to support mobile applications. Here you are re-purposing the custom token provider you created to support Mobile SSO. If you do not need to support mobile applications, you do not need to extend this class.

11.1.1.1 Implementing the TokenService Interface

Refer to the TokenService Java documentation for details about the API to be implemented. Note that you will also need to implement the `LifecycleServiceProvider` and `UserAuthenticator` interfaces.

In the custom token provider, you must implement the `createTokens()` method to reuse this authentication Service Provider to support mobile clients.

When returning a `Token` object (under successful conditions) or throwing a `RESTUnauthorizedException` (under unsuccessful conditions), a `PluginContext` object needs to be created. This object needs to be included in the returned `Token` object or the thrown `RESTUnauthorizedException`.

- Each Service Domain has a Security Handler plug-in. The Security Handler Plug-in tracks user behavior patterns and, if necessary, can issue an authentication challenge (for example, a knowledge-based authentication challenge). The `OAAMSecurityHandlerPlugin` is included with Mobile and Social. Oracle Adaptive Access Manager integration is required for knowledge-based authentication (KBA) challenges.
- After an authentication Service Provider token operation, a Security Handler Plug-in is typically invoked. Data in this `PluginContext` is used to communicate with the security plug-in. If the `Token` object or `RESTUnauthorizedException` object does not contain a `PluginContext` object, the configured security plug-in is not invoked.
- A `PluginContext` object is created through the `PluginDataFactory` API. The `PluginContext` API collects security data, such as the type of security event, the User ID, the client application ID, and their corresponding ID authentication status and types. Refer to the Javadocs for details.

11.1.1.2 Extending the MobileCompositeTokenServiceProvider

Extend this class to reuse the custom token Service Provider to support mobile devices.

Implement the `getComponentTokenServiceProviderClass()` API. Refer to the following sample code:

```
protected Class getComponentTokenServiceProviderClass() {
  return CustomTokenProvider.class;
  // CustomTokenProvider is the class name you implemented. Change the name to
  // the name that you used when implementing the TokenService.
}
```

For more information, see the Java documentation for this class.

11.1.2 Building the Custom Authentication Service Provider

Build the custom authentication Service Provider as follows.

11.1.2.1 To Build the Custom Authentication Service Provider

1. Gather the `oic_rest.jar` file, the `oic_common.jar` file, and any additional JAR files needed for your custom code.

   For example:

   ```
   com/example/tokenprovider/MyTokenProvider.java implementing TokenService
   com/example/tokenprovider/MobileMyTokenProvider.java extending
   MobileCompositeTokenServiceProvider
   ```

2. Build the custom token provider.

   For example:

   ```
   javac -cp ./oic_rest.jar:./oic_common.jar
   com/example/tokenprovider/MyTokenProvider.java

   javac -cp ./oic_rest.jar:./oic_common.jar:.
   com/example/tokenprovider/MobileMyTokenProvider.java
   ```

3. Build the JAR file.

   For example:

```
jar cvf mytokenpro.jar com/example/tokenprovider/*.class
```

11.1.3 Deploying the Custom Authentication Service Provider

Deploy the custom authentication Service Provider as follows.

11.1.3.1 To Deploy the Custom Authentication Service Provider

1. Copy `mytokenpro.jar` to your deployment's `fmwconfig/oic/plugins` directory.

   The JAR files here are dynamically picked up by Mobile and Social. If additional JAR files are needed for the custom Service Provider, then those files need to be available in the CLASSPATH of the container.

2. To configure your custom token provider from the Administration console, choose **System Configuration > Mobile and Social > Mobile Services > Service Providers > Authentication Service Providers**.

   Create a new Service Provider, for example *MyTokenProvider*.

3. Configure your custom token Provider for mobile SSO applications.

   If you implemented `MobileCompositeTokenServiceProvider`, from the Administration console choose **System Configuration > Mobile and Social > Mobile Services > Service Providers > Authentication Service Providers**.

   Create a new Service Provider, for example *MobileMyTokenProvider*.

4. Configure the authentication Service instances, which use the custom token providers as defined in steps 2 and 3.

   From the Administration console choose **System Configuration > Mobile and Social > Mobile Services > Service Domains >** *Select a Service Domain* **> Authentication Services**.

   Create a new instance, for example *MyTokenService*.

5. Define an authentication service instance, which is using custom mobile token providers as defined in steps 2 and 3.

   From the Administration console choose, **System Configuration > Mobile and Social > Mobile Services > Service Domains >** *Select a Service Domain* **> Authentication Services**.

   Create a new instance, for example *MobileMyAuthnService*.

The custom authentication Service Providers can now be used in the deployment.

11.2 Create a new Identity Service Provider for Internet Identity Services

Mobile and Social provides support for the following Identity Providers: Facebook, Google, LinkedIn, Twitter, and Yahoo. You can add additional OpenID and OAuth service providers by implementing the `IdentityProvider` Java interface, and then use the System Administration Console to add the provider to your Mobile and Social deployment.

This section covers the following topics:

- Developing the Custom Identity Service Provider
- Building the Custom Identity Service Provider
- Deploying the Custom Identity Service Provider

11.2.1 Developing the Custom Identity Service Provider

The interface has three methods:

- `authenticateUser()` - This method initiates the process of authenticating the User with the Identity Provider. After authentication, the Identity Provider uses the Return URL sent in the authentication request to return Identity profile information to the Mobile and Social server.

  There are two return URL options:

  - `https://host.example.com:port/oic_rp/popup` - Use this option if the Identity Provider login page opens in a pop-up window.
  - `https://host.example.com:port/oic_rp /return` - Use this option if the Identity Provider login page opens in the same browser window as the application's login page.
- `getAccessToken()` - If the Identity Provider uses the OAuth protocol, the Mobile and Social server needs to get an Access Token using this method. The Mobile and Social server uses the Access Token to get a User Token.
- `getUserProfile()` - This method gets the User profile from the Identity Provider.

11.2.2 Building the Custom Identity Service Provider

Build the custom Identity Service Provider as follows.

11.2.2.1 To Build the Custom Identity Service Provider

1. Gather the `oic_rp.jar` file, the `oic_common.jar` file, and the `j2ee.jar` file.
2. Build the class.

   For example if the Identity Provider name is XYZ:

   ```
   javac -cp ./j2ee.jar:./oic_rp.jar:./oic_common.jar
   com/xyz/custom/idp/XYZImpl.java
   ```

   Add any additional JAR files as required by your custom code.
3. Build the JAR file.

   For example:

   ```
   jar cvf xyz-idp.jar com/xyz/custom/idp/XYZImpl.class
   ```

11.2.3 Deploying the Custom Identity Service Provider

Deploy the custom authentication Service Provider as follows. The following steps use *XYZProvider* as an example.

11.2.3.1 To Deploy the Custom Identity Service Provider

1. Copy `xyz-idp.jar` to your deployment's `fmwconfig/oic/plugins` directory.

   The JAR files here are dynamically picked up by Mobile and Social. If additional JAR files are needed for the custom Service Provider, then those files need to be available in the CLASSPATH of the container.
2. To configure your custom Identity Provider from the Administration console, choose **System Configuration > Mobile and Social > Internet Identity Services.**

In the **Internet Identity Providers** section click **Create** to add the new Internet Identity Provider, for example *XYZProvider*.

- Define any attributes needed under **Protocol Attributes**. These attributes are consumed in the custom implementation.
- Define any User attributes in the **User Attributes Returned** section. These attributes are consumed in the custom implementation as part of the `getUserProfile()` method logic.

Or, instead of using the Administration console, add the following XML to `oic_rp.xml`:

```
    <InternetIdentityProvider description="XYZ OAuth Provider"
name="XYZProvider">
        <icon>XYZ.gif</icon>
        <protocolType>OAuth</protocolType>
        <userAttribute>
            <name>id</name>
            <value>id</value>
        </userAttribute>
        <userAttribute>
            <name>first_name</name>
            <value>first_name</value>
        </userAttribute>
        <userAttribute>
            <name>last_name</name>
            <value>last_name</value>
        </userAttribute>
        <userAttribute>
            <name>email</name>
            <value>email</value>
        </userAttribute>
        <userAttribute>
            <name>location</name>
            <value>location</value>
        </userAttribute>
        <userAttribute>
            <name>birthday</name>
            <value>birthday</value>
        </userAttribute>
        <userAttribute>
            <name>gender</name>
            <value>gender</value>
        </userAttribute>
        <userAttribute>
            <name>language</name>
            <value>language</value>
        </userAttribute>
        <userAttribute>
            <name>country</name>
            <value>country</value>
        </userAttribute>
        <userAttribute>
            <name>profile_image_url</name>
            <value>profile_image_url</value>
        </userAttribute>
        <providerImplClass>com.xyz.custom.idp.XYZImpl</providerImplClass>
    </InternetIdentityProvider>
```

3. Create or Edit the Application Profile that will use the custom Identity Provider.

For instructions, see "Editing or Creating Application Profiles" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

4. In the **Application User Attribute Vs. Internet Identity Provider User Attributes Mapping** section select XYZProvider and define the User attribute mapping.

12

Sending Mobile and Social REST Calls With cURL

This chapter uses cURL to demonstrate the REST calls that the Mobile and Social client sends to the Mobile and Social server. This chapter includes the following topics:

- Request and Response Header Attribute Name Reference
- Mobile and Social REST Security Filter Reference
- Mobile Services REST Reference: Authentication and Authorization
- Mobile Services REST Reference: Commands for Mobile Single Sign-on Tokens
- Mobile Services REST Reference: Commands for User Profile Services
- Practical Examples

Notes About Using cURL

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 Mobile and Social client interacts with the Mobile and Social server. It can also be a helpful troubleshooting tool.

Note: 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.

Request and Response Header Attribute Name Reference

This section documents the request and response attribute names that are reserved for use with Mobile and Social REST Services. These attributes can be included in a query parameter, in an HTTP header, or in the JSON body portion of the header as noted.

Note: All attribute names and values are case-sensitive.

The following attribute names are documented in this section:

- X-IDAAS-REST-VERSION
- X-IDAAS-SERVICEDOMAIN
- X-IDAAS-REST-AUTHORIZATION
- AUTHORIZATION
- X-Idaas-Rest-Subject-Type
- X-Idaas-Rest-Subject-Value
- X-Idaas-Rest-Subject
- X-Idaas-Rest-Subject-Username
- X-Idaas-Rest-Subject-Password
- X-Idaas-Rest-New-Token-Type-To-Create
- X-Idaas-Rest-Application-Context
- X-Idaas-Rest-Application-Resource
- X-Idaas-Rest-User-Principal
- X-Idaas-Rest-Provider-Type

X-IDAAS-REST-VERSION

Use this attribute to specify the specific version of the SDK that the client application is compatible with. If you do not specify an SDK version, the Mobile and Social server defaults to using the latest SDK version.

Where to use This Attribute

- HTTP header
- Query parameter

Attribute Type

- Request
- Response

Sample cURL Command

```
-H "X-IDAAS-REST-VERSION:v1"
```

Sample Request

```
curl -i
-H "Content-Type: application/json http://host.us.example.com:14100/oic_rest
/rest/jwtauthentication/authenticate
-d '{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"profileid1",
"X-Idaas-Rest-Subject-Password":"secret12",
"X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTTOKEN"}'
-H "X-IDAAS-REST-VERSION:v1"
```

Sample Response

```
HTTP/1.1 200 OK Date: Tue, 05 Jun 2012 11:23:19 GMT Transfer-Encoding: chunked
Content-Type: application/json
X-IDAAS-REST-VERSION: v1
Set-Cookie: JSESSIONID=5Z4sPNsHVmrplgs8HNDbQGxddC7TJQS7s4QspYvMpcMJJLC2nGx5!1574
236250;
path=/;
HttpOnly
X-ORACLE-DMS-ECID:a393487d2600b00c:-7abb0b83:137b52ee014:-8000-00000000000026aa
X-Powered-By: Servlet/2.5 JSP/2.1
```

Comments

The attribute value must be a string representation of the protocol version, for example `v1`.

X-IDAAS-SERVICEDOMAIN

Use to specify a Service Domain value. If a Service Domain value is not provided, the system will use the "Default" Service Domain.

Where to use This Attribute

- HTTP header

Attribute Type

- Request only

Sample cURL Command

```
-H "X-IDAAS-SERVICEDOMAIN: Default"
```

Sample Request

```
curl -i
-H "Content-Type: application/json"--request POST
http://host.us.example.com:14100/oic_rest/rest/jwtauthentication/authenticate
-d '{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"profileid1",
"X-Idaas-Rest-Subject-Password":"secret12",
"X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTTOKEN"}'
-H "X-IDAAS-REST-VERSION:v1"
-H "X-IDAAS-SERVICEDOMAIN: Default"
```

Comments

The attribute value must be a string representation of the target Service Domain, for example `MyMobileServiceDomain`.

X-IDAAS-REST-AUTHORIZATION

Use to specify an *application* credential in the HTTP request header.

Use the following format:

`-H "X-IDAAS-REST-AUTHORIZATION: <AuthenticationScheme-Name> <Credential Value>"`

where *AuthenticationScheme-Name* is one of the following:

- `HTTP Basic`
- `UIDPassword`
- `Token`

Where to use This Attribute

- HTTP header

Attribute Type

- Request only

Sample cURL Commands

```
-H "X-IDAAS-REST-AUTHORIZATION: Token eyJhbG56I4OTg5OTk3M...fW1VGmunfzqZ-bG4rM"

-H "X-IDAAS-REST-AUTHORIZATION: Basic fn49xkOVXunF%2B5zMQUiGUlwTXPYiKw"

-H "X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred=\"Tp8aUEeptClBz6h9cH8F%2Fwk976\""
```

Sample Request

```
curl -i -H "Content-Type: application/json" --request POST
http://host.us.example.com:14100/oic_rest/rest/jwtauthentication/authenticate
-d '{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"sampleuser",
"X-Idaas-Rest-Subject-Password":"password123",
"X-Idaas-Rest-New-Token-Type-To-Create":"USERTOKEN"}'
-H "X-IDAAS-REST-VERSION:v1"
-H "X-IDAAS-SERVICEDOMAIN: Default"
-H "X-IDAAS-REST-AUTHORIZATION: Token eyJhbGciOiJSUzUxMiIsInR5cSldUQXV0aGVudGljYXR
CI6IkpXVCIsImtpZCI6Im9yYWtleSJ9.eyJleHAiOjEzMzg4OTg5OTk3MzIsIzZXJ2ZXIxIiwiaXNzIjoi
joiY2I2MWU5YTQtZjJmYS00ZDQzLWFlOTYtZWQ5MjZlMGQ2NDZlIiwib3JhY2xlLm9pYy50b2tlbi50eXB
lIjoiQ0xJRU5UVE9LRU4iLCJpYXQiOjEzMzg4OTUzOTk3MzIsIm9yYWNsZS5vaWMudG9rZW4udXNlcl9kb
iI6InVpZD1wcm9maWxlaWQxLG91PXBlb3BsZSxvdT1teXJlYWxtLGRjPWJhc2VfZG9tYWsZSxvdT1teXJl
YWxtLGRjPWJhc2VfZG9tYWluIn0.kN17W0N3GEmdccm7GoUOT4iP23yWb6LloleOJ0grZkeiijXE-t8Kfy
N6Jq1m8EKzdYgiKFwdb-SO9MpOVMyPgxSRER9mn_3kkcKNagl7yIgu0EJUOS3Hudy2Suv0Th5b6fDgXLIY
LkBA0cC1WlP5RgW1VGmuBX7RnfzqZ-bG4rMiLCJwcm4iOiJwcm9maWxlaWQxIiwianRpI"
```

Comments

The client application must send a security credential using the X-IDAAS-REST-AUTHORIZATION header if you select the **Secured Application** option for either **User Profile Services** or **Authorization Services** on the Service Domain Configuration "Service Protection" tab. The server accepts credentials sent using any of the three valid security schemes (HTTP Basic, UIDPassword, or Token).

AUTHORIZATION

Use to specify a *user* credential in the HTTP request header. Use the `AUTHORIZATION` header if a User Token is required and you are using either a JWTAuthentication or an OAMAuthentication token format. The User Token value has to be the User token issued by the authentication Service Provider.

Use the following format:

`-H "AUTHORIZATION:<User Token Value>"`

Where to use This Attribute

- HTTP header

Attribute Type

- Request only

Sample cURL Command

```
-H "AUTHORIZATION:eyJhbGciOiJSUzUxMiIsInR5cmtpZCI6Im9g5OTk3M...sW1VGmunfzqZ-bG4rM"
```

Sample Request

```
curl -i --request GET
"http://host.us.example.com:14100/oic_rest/rest/userprofile/people/weblogic/"
-H
"AUTHORIZATION:eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6Im9yYWtleSJ9.eyJleHi
EzMzg4OTk3MTMxMzcsImF1ZCI6Im9hbV9zZXJ2ZXIxIiwiaXNzIjoiSldUQXV0aGVudGljYXRpb24iLCJw
cm4iOiJ3ZWJsb2dpYyIsImp0aSI6IjNlMjdiZjc4LTg3NDQtNDFkMS05MzlmLTlkZGY0N2VkNGFlNyIsIm
YWNsZS5vaWMudG9rZW4udHlwZSI6IlVTRVJUT0tFTiIsImlhdCI6MTMzODg5NjExMzEzNywib3JhY2xlLm
9pYy50b2tlbi51c2VyX2RuIjoidWlkPXdlYmxvZ2ljLG91PXBlb3BsZSxvdT1teXJlYWxtLGRjPWJhc2V6
ZG9tYWluIn0.hHmAa5Syw3AcqRPwIq_XLx6DcMzCBzvDXGFYvwAf9nqVgxgvLTJJfxZzofS5Ut272b0dFG
sv3qakeDm2NTgg6fR2YKH5BxAHnEmq0IAmhLuyWdux_rMZNB-wP8h5JD26UQf_nnBBWApvgULeM2mWQEzY
RVDMpN9K7pycNrsGKOj8U"
```

Comments

The client application must send a security credential using the AUTHORIZATION header if you select the **Secured User** option for either **User Profile Services** or **Authorization Services** on the Service Domain Configuration "Service Protection" tab. The server accepts tokens only.

X-Idaas-Rest-Subject-Type

The type of the subject (either `USERCREDENTIAL`, `UID`, `UIDASSERTION`, or `TOKEN`).

Where to use This Attribute

- Query parameter
- JSON body

Attribute Type

- Request only

Sample cURL Command

```
-d '{"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL"}'

-d '{"X-Idaas-Rest-Subject-Type":"UID"}'

-d '{"X-Idaas-Rest-Subject-Type":"UIDASSERTION"}'
```

Sample Request 1

```
curl -H "Content-Type: application/json" --request GET
"http://host.us.example.com:14100/oic_rest/rest/jwtauthentication/validate?
X-Idaas-Rest-Subject-Value=eyJhbGciOiJSUzU...I_A0PM&
X-Idaas-Rest-Subject-Type=TOKEN"
```

Sample Request 2

```
curl -i -H "Content-Type: application/json" --request POST
http://host.us.example.com:14100/oic_rest/rest/jwtauthentication/authenticate
-d '{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"profileid1",
"X-Idaas-Rest-Subject-Password":"secret12345",
"X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTTOKEN"}'
```

Comments

The attribute value must be one of the following:

- `USERCREDENTIAL`
- `UID`
- `UIDASSERTION`
- `TOKEN`

X-Idaas-Rest-Subject-Value

The string value of the subject. Include this attribute when the value of `X-Idaas-Rest-Subject-Type` is either `TOKEN`, `UID`, or `UIDASSERTION`.

Where to use This Attribute

- Query parameter
- JSON body

Attribute Type

- Request only

Sample cURL Command

Sample Request 1

```
curl -H "Content-Type: application/json" --request GET
"http://host.example.com:14100/oic_rest/rest/jwtauthentication/validate?
X-Idaas-Rest-Subject-Value~=eyJhbGciOiJSUzU...PM&
X-Idaas-Rest-Subject-Type~=TOKEN"
```

Sample Request 2

```
curl -H "Content-Type: application/json" --request POST
http://localhost:18001/oic_rest/rest/jwtauthentication/access
-d '{
"X-Idaas-Rest-Subject-Type":"TOKEN",
"X-Idaas-Rest-Subject-Value":"vTBI8jN...%3D",
"X-Idaas-Rest-Application-Context":"75sSbBZZKJiUOAWikZxsKA==",
"X-Idaas-Rest-Application-Resource":"http:/host.example.com:7779/index.html",
"X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN"}'
```

X-Idaas-Rest-Subject

Use to supply both the subject type and string value in the header when the subject type is of type `TOKEN`.

Where to use This Attribute

- HTTP header

Attribute Type

- Request only

Sample cURL Command

Sample Request

```
curl -H "Content-Type: application/json" --request GET
http://host.example.com:14100/oic_rest/rest/jwtauthentication/validate
-H
"X-Idaas-Rest-Subject: TOKEN eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6Im9yYWtl
eSJ9.eyJleHAiOjEzMzg5MDEzMzUyMjUsImF1ZCI6Im9hbV9zZXJ2ZXIxIiwiaXNzIjoiSldUQXV0aGVu
dGljYXRpb24iLCJwcm4iOiJ3ZWJsb2dpYyIsImp0aSI6ImUzNDZiYjJiLTQyZmYtNGRjMC1hOTZkLWYyY
2U5MjM0NTM0YSIsIm9yYWNsZS5vaWMudG9rZW4udHlwZSI6IlVTRVJUT0tFTiIsImlhdCI6MTMzODg5Nz
czNTIyNSwib3JhY2xlLm9pYy50b2tlbi51c2VyX2RuIjoidWlkPXdlYmxvZ2ljLG91PXBlb3BsZSxvdT1
teXJlYWxtLGRjPWJhc2VfZG9tYWluIn0.GZ3-X4NRGdQ99MB63B5MmPuyE5M2kFwqHMQ97AXwBjYElMep
ZdziTEgDeYLKJuVB83plSGwpfQEDdzlxR3Sy7tRXbfV3EdK1lpbUyUyEEIwAfuu4xtbNERKrPw3pJoPtU
q0TCd0BV2sRdyy1zuSBdU2J6zUjG8rW-PYDWI_A0PM"
```

X-Idaas-Rest-Subject-Username

Use to supply the user name as a string only if the `X-Idaas-Rest-Subject-Type` value is `USERCREDENTIAL`.

Where to use This Attribute

- JSON body

Attribute Type

- Request only

Sample cURL Command

Sample Request

```
curl -i -H "Content-Type: application/json" --request POST
http://host.example.com:14100/oic_rest/rest/jwtauthentication/authenticate
-d '{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"sampleuser",
"X-Idaas-Rest-Subject-Password":"password123",
"X-Idaas-Rest-New-Token-Type-To-Create":"USERTOKEN"}'
```

X-Idaas-Rest-Subject-Password

Use to supply the password as a string only if the `X-Idaas-Rest-Subject-Type` value is `USERCREDENTIAL`.

Where to use This Attribute

- JSON body

Attribute Type

- Request only

Sample cURL Command

Sample Request

```
curl -i -H "Content-Type: application/json" --request POST
http://host.example.com:14100/oic_rest/rest/jwtauthentication/authenticate
-d '{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"sampleuser",
"X-Idaas-Rest-Subject-Password":"password123",
"X-Idaas-Rest-New-Token-Type-To-Create":"USERTOKEN"}'
```

X-Idaas-Rest-New-Token-Type-To-Create

Use to provide the token types to be created. Multiple token types can be specified in a request.

Where to use This Attribute

- JSON body

Attribute Type

- Request only

Sample cURL Command

Sample Request

```
curl -i -H "Content-Type: application/json" --request POST
http://host.example.com:14100/oic_rest/rest/jwtauthentication/authenticate
-d '{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"sampleuser",
"X-Idaas-Rest-Subject-Password":"password123",
"X-Idaas-Rest-New-Token-Type-To-Create":"USERTOKEN"}'
```

Comments

The attribute value must be one of the following:

- `CLIENTREGHANDLE`
- `CLIENTTOKEN`
- `USERTOKEN`
- `USERTOKEN::OAMMT`
- `ACCESSTOKEN`

X-Idaas-Rest-Application-Context

Use to specify the application context for which an Access Token is needed. The supplied value must be string.

Where to use This Attribute

- JSON body

Attribute Type

- Request only

Sample cURL Command

Sample Request 1

```
curl -H "Content-Type: application/json"
--request POST http://localhost:18001/oic_rest/rest/jwtauthentication/access
-d '{
"X-Idaas-Rest-Subject-Type":"TOKEN",
"X-Idaas-Rest-Subject-Value":"vTBI8jN8eYIsfAp%2BZqe...GkiaHv7TjAGZ5XFSQk5A%3D%3D",
"X-Idaas-Rest-Application-Context":"75sSbBZZKJiUOAWikZxsKA==",
"X-Idaas-Rest-Application-Resource":"http://somehost.example.com:7779/index.html",
"X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN"}'
```

X-Idaas-Rest-Application-Resource

Use to specify the target resource for which an Access Token is needed. The supplied value must be string.

Where to use This Attribute

- JSON body

Attribute Type

- Request only

Sample cURL Command

Sample Request 1

```
curl -H "Content-Type: application/json"
--request POST http://localhost:18001/oic_rest/rest/jwtauthentication/access
-d '{
"X-Idaas-Rest-Subject-Type":"TOKEN",
"X-Idaas-Rest-Subject-Value":"vTBI8jN8eYIsfAp%2BZqe...GkiaHv7TjAGZ5XFSQk5A%3D%3D",
"X-Idaas-Rest-Application-Context":"75sSbBZZKJiUOAWikZxsKA==",
"X-Idaas-Rest-Application-Resource":"http://somehost.example.com:7779/index.html",
"X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN"}'
```

X-Idaas-Rest-User-Principal

Used to return the principal User.

Where to use This Attribute

- JSON body

Attribute Type

- Response only

Sample cURL Command

Sample Response

```
HTTP/1.1 200 OK Date: Tue, 05 Jun 2012 11:35:13 GMT
Transfer-Encoding: Content-Type: application/json X-IDAAS-REST-VERSION: v1
Set-Cookie: JSESSIONID=
TCjjPNnRvL6fvhJpMSjLhHYrFyMKqwcFxTNL1RQzyvkSJ7G2TLj4!1574236250;
path=/; HttpOnly X-ORACLE-DMS-ECID: a393487d2600b00c:-7abb0b83:137b52ee014:
-8000-00000000000026f5 X-Powered-By: Servlet/2.5 JSP/2.1
{
"X-Idaas-Rest-Token-Value":"eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6Im9yYWtle
SJ9.eyJleHAiOjEzMzg4OTk3MTMxMzcsImF1ZCI6Im9hbV9zZXJ2ZXIxIiwiaXNzIjoiSldUQXV0aGVud
GljYXRpb24iLCJwcm4iOiJ3ZWJsb2dpYyIsImp0aSI6IjNlMjdiZjc4LTg3NDQtNDFkMS05MzlmLTlkZG
d0N2VkNGFlNyIsIm9yYWNsZS5vaWMudG9rZW4udHlwZSI6IlVTRVJUT0tFTiIsImlhdCI6MTMzODg5NjE
xMzEzNywib3JhY2xlLm9pYy50b2tlbi51c2VyX2RuIjoidWlkPXdlYmxvZ2ljLG91PXBlb3BsZSxvdT1t
eXJlYWxtLGRjPWJhc2VfZG9tYWluIn0.hHmAa5Syw3AcqRPwIqXLx6DcMzCBzvDXGFYvwAf9nqVgxgvLT
JfxZzofS5Ut272b0dFGsv3qakeDm2NTgg6fR2YKH5BxAHnEmq0IAmhLuyWdux_rMZNB-wP8h5JD26UQf
nnBBWApvgULeM2mWQEzYRVDMpN9K7pycNrsGK8U",
"X-Idaas-Rest-User-Principal":"jdoe",
"X-Idaas-Rest-Provider-Type":"JWT",
"X-Idaas-Rest-Token-Type":"USERTOKEN"
}
```

X-Idaas-Rest-Provider-Type

Used to return the token provider type. Valid values include `OAM_10G`, `OAM_11G`, and `JWT`.

Where to use This Attribute

- JSON body

Attribute Type

- Response

Sample cURL Command

Sample Response

```
HTTP/1.1 200 OK Date: Tue, 05 Jun 2012 11:35:13 GMT
Transfer-Encoding: chunked Content-Type: application/json X-IDAAS-REST-VERSION: v1
Set-Cookie:JSESSIONID=TCjjPNnRvL6fvhJpMSjLhHYrFyMKqwcFxTNL1RQzyvkSJ7G2TLj4!157423;
path=/; HttpOnly X-ORACLE-DMS-ECID:
a393487d2600b00c:-7abb0b83:137b52ee014:-8000-00000000000026f5
X-Powered-By: Servlet/2.5 JSP/2.1
{
"X-Idaas-Rest-Token-Value":"eyJhbGciOiJSUzUxMiIsInR
5cCI6IkpXVCIsImtpZCI6Im9yYWtleSJ9.eyJleHAiOjEzMzg4OTk3MTMxMzcsImF1ZCI6Im9hbV9
zZXJ2ZXIxIiwiaXNzIjoiSldUQXV0aGVudGljYXRpb24iLCJwcm4iOiJ3ZWJsb2dpYyIsImp0aSI6IjN
lMjdiZjc4LTg3NDQtNDFkMS05MzlmLTlkZGY0N2VkNGFlNyIsIm9yYWNsZS5vaWMudG9rZW4
udHlwZSI6IlVTRVJUT0tFTiIsImlhdCI6MTMzODg5NjExMzEzNywib3JhY2xlLm9pYy50b2tlbi51c2
VyX2RuIjoidWlkPXdlYmxvZ2ljLG91PXBlb3BsZSxvdT1teXJlYWxtLGRjPWJhc2VfZG9tYWluIn0.h
HmAa5Syw3AcqRPwIq_XLx6DcMzCBzvDXGFYvwAf9nqVgxgvLTJJfxZzofS5Ut272b0dFGsv3q
akeDm2NTgg6fR2YKH5BxAHnEmq0IAmhLuyWdux_rMZNB-wP8h5JD26UQf_nnBBWApvgULeM
2mWQEzYRVDMpN9K7pycNrsGK8U",
"X-Idaas-Rest-User-Principal":"weblogic",
"X-Idaas-Rest-Provider-Type":"JWT",
"X-Idaas-Rest-Token-Type":"USERTOKEN"
}
```

Mobile and Social REST Security Filter Reference

The authorization schemes in this section are used to protect the Mobile and Social REST Services.

The following calls are demonstrated:

- Authorize With UIDPASSWORD
- Authorize With HTTP Basic
- Authorize With an Access Manager Token

Authorize With UIDPASSWORD

Shows how to send the REST call required for UIDPASSWORD authentication.

cURL Command

```
curl --request GET
"localhost:18001/idaas_rest/rest/authorizationservice3/authorization?
resource=http://is-x86-05.us.example.com:7779/index.html&
action=GET&X-Idaas-Rest-Subject-Value=
ZNsJcMMM3ow83Zr5D8KqCPnhBGmui4RnBvUXJ5dqC7OfwZIv6FDcYWwfPuHupxN%2B
fs5qN0I6AWIZBX%2F2KQNNQ5bPDN1XqeE8y7OPPoy4znteEfCaRHb7UA1ia1ox%2BW8
5LbknXCLaZ5q%2FN4I0IcXP%2B13FGX9r9LROQ3OZZVNMLhfx3KabZcIVmSHBkK%2F
ARGYEJQv6RO%2FPCMN2YYTJgWxGr20rWeG8NLbzgN%2FPyADxxlPLvkxH2YCVHHH
7bLBfOp3p83IbJ%2FC%2Bm9sCd4YjlSlhsMUXKtvZ1LnJME4UymuR5tXuw2B0Yr25OHxU
bMreIGgRYZXFonmjhAovKhXqIgzpIg%3D%3D&
X-Idaas-Rest-Subject-Type=TOKEN"
-H "X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred=\"
Tp8aUEeptClBz6A6h9cH8F%2FwcZJvLok976\""
-H "Authorization: gdX4z0leySgt0DiPeItsQfBweYZIfZ2dm7fVypNz%2Bf6pbrzF7P4
AvUzPXIzLf2lL0zHuvNI%2B77OsUESM99U6zQjytC%2FgrAD6O2QdSe2VUNGjjw8Di5ev1
gSI0m5a5VQO9rmGNlB1xndnPYoaX0nDpi3eGAyQNw3PUAbEGYglsDMR1js2jsiXKyexryn
8k1coc3EHGqk%2ByqfEXzfzGjwEB4ipnSGg2c4a9BX2BKjKLoOD0PdNVc2nf6f%2F7T2Ck
hA%2BSFowwE%2BEIzvQ7cVbeRYqco2eYCJhs8GS8Haq9T2dnhIAa4tux9MyxVLRNRtDd
q39HDr5hvUI7OpHQHNUMeRcPQ%3D%3D"
```

Expected Output

```
{
"Allowed":"true"
}
```

Comments

- In a request, use the `X-IDAAS-SERVICEDOMAIN` header name to specify a Service Domain value. The `X-IDAAS-SERVICEDOMAIN` name can be used as a query parameter or a header. If a Service Domain value is not provided, the system will use the "Default" Service Domain.

Authorize With HTTP Basic

Shows how to send the REST call required for HTTP Basic authorization.

cURL Command

```
curl --request GET
"localhost:18001/idaas_rest/rest/authorizationservice3/authorization?
resource=http://is-x86-05.us.example.com:7779/index.html
&action=GET&
X-Idaas-Rest-Subject-Value=
ZNsJcMMM3ow83Zr5D8KqCPnhBGmui4RnBvUXJ5dqC7OfwZIv6FDcYWwfPuHupxN%2Bfs5
qN0I6AWIZBX%2F2KQNNQ5bPDN1XqeE8y7OPPoy4znteEfCaRHb7UA1ia1ox%2BW85Lbkn
XCLaZ5q%2FN4I0IcXP%2B13FGX9r9LROQ3OZZVNMLhfx3KabZcIVmSHBkK%2FARGYEJ
Qv6RO%2FPCMN2YYTJgWxGr20rWeG8NLbzgN%2FPyADxxlPLvkxH2YCVHHH7bLBfOp3p
83IbJ%2FC%2Bm9sCd4YjlSlhsMUXKtvZ1LnJME4UymuR5tXuw2B0Yr25OHxUbMreIGgRYZ
XFonmjhAovKhXqIgzpIg%3D%3D&
X-Idaas-Rest-Subject-Type=TOKEN"
-H "X-IDAAS-REST-AUTHORIZATION: Basic Tp8aUEeptClBz6A6h9cH8F%2FwcZJvLok976"
-H "Authorization: TOKEN gdX4z0leySgt0DiPeItsQfBweYZIfZ2dm7fVypNz%2Bf6pbrzF7P4A
vUzPXIzLf2lL0zHuvNI%2B77OsUESM99U6zQjytC%2FgrAD6O2QdSe2VUNGjjw8Di5ev1gS
I0m5a5VQO9rmGNlB1xndnPYoaX0nDpi3eGAyQNw3PUAbEGYglsDMR1js2jsiXKyexryn8k1
coc3EHGqk%2ByqfEXzfzGjwEB4ipnSGg2c4a9BX2BKjKLoOD0PdNVc2nf6f%2F7T2CkhA%2B
SFowwE%2BEIzvQ7cVbeRYqco2eYCJhs8GS8Haq9T2dnhIAa4tux9MyxVLRNRtDdq39HDr5hv
UI7OpHQHNUMeRcPQ%3D%3D"
```

Expected Output

```
{
"Allowed":"true"
}
```

Comments

- A contract name can be specified as a query parameter or a header with a name of `X-IDAAS-CONTRACT`. Otherwise, Mobile and Social assumes the contract is a "default" contract.
- HTTPBasic has to be configured for client with an encrypted password in the client definition as shown here:

```
<IdaasClient description="OIC Client 1" name="clientid1">
<authnService>sampletokenservice</authnService>
<param>
<name>userId4BasicAuth</name>
<value>rest_client1</value>
</param>
<param>
<name>sharedSecret4BasicAuth</name>
<value>9Qo9olLIl5gDwESYR0hOgw==</value>
</param>
</IdaasClient>
```

Authorize With an Access Manager Token

Shows how to send the REST call required for Access Manager authorization.

cURL Command

```
curl --request GET
"localhost:18001/idaas_rest/rest/authorizationservice3/authorization?
resource=http://is-x86-05.us.example.com:7779/index.html
&action=GET&
X-Idaas-Rest-Subject-Value=ZNsJcMMM3ow83Zr5D8KqCPnhBGmui4RnBvUXJ5dqC7OfwZIv6
FDcYWwfPuHupxN%2Bfs5qN0I6AWIZBX%2F2KQNNQ5bPDN1XqeE8y7OPPoy4znteEfCaRHb
7UA1ia1ox%2BW85LbknXCLaZ5q%2FN4I0IcXP%2B13FGX9r9LROQ3OZZVNMLhfx3KabZcIV
mSHBkK%2FARGYEJQv6RO%2FPCMN2YYTJgWxGr20rWeG8NLbzgN%2FPyADxxlPLvkxH2
YCVHHH7bLBfOp3p83IbJ%2FC%2Bm9sCd4YjlSlhsMUXKtvZ1LnJME4UymuR5tXuw2B0Yr25
OHxUbMreIGgRYZXFonmjhAovKhXqIgzpIg%3D%3D
&X-Idaas-Rest-Subject-Type=TOKEN"
-H "X-IDAAS-REST-AUTHORIZATION: TOKEN Tp8aUEeptClBz6A6h9cH8F%2FwcZJvLok976
c5q0SitrrgSCJ5FQk58KMtUg2FCPLbjZbP2%2B3P5zZPiSCeHwNua%2FBHdIDCOnUYOXNg
4uBKA7t7O4jGRfn49xkOVXunF%2B5zMQUiGUlwTXPYiKwooAknkeHs3HIq6s2if%2FHpuPH
curRa%2BdyfjWfYWTpqPeo%2FzyHHzDH1wF8hM6k6YwJ%2FpxD8avuXogP%2Bp5j2tCZ0
aAhonseNMcKvGTRBoV1shGnotK9gt01nDgc2LWA5oidJgxlcaWDw3%2FXZhvgudkLwl0jxEw
0K%2BzffyeZs0gfUkZJBnsm8qh2KP%2BiCPzT7HPVPF%2FyYCg%3D%3D"
-H "Authorization: TOKEN gdX4z0leySgt0DiPeItsQfBweYZIfZ2dm7fVypNz%2Bf6pbrzF7P4AvU
zPXIzLf2lL0zHuvNI%2B77OsUESM99U6zQjytC%2FgrAD6O2QdSe2VUNGjjw8Di5ev1gSI0m5
a5VQO9rmGNlB1xndnPYoaX0nDpi3eGAyQNw3PUAbEGYglsDMR1js2jsiXKyexryn8k1coc3EH
Gqk%2ByqfEXzfzGjwEB4ipnSGg2c4a9BX2BKjKLoOD0PdNVc2nf6f%2F7T2CkhA%2BSFowwE
%2BEIzvQ7cVbeRYqco2eYCJhs8GS8Haq9T2dnhIAa4tux9MyxVLRNRtDdq39HDr5hvUI7OpHQ
HNUMeRcPQ%3D%3D"
```

Expected Output

```
{
"Allowed":"true"
}
```

Comments

- A contract name can be specified as a query parameter or a header with a name of `X-IDAAS-CONTRACT`. Otherwise, Mobile and Social assumes the contract is a "default" contract.
- The client has to be defined with a token service name that can validate the token for the request as shown here:

  ```
  <IdaasClient description="OIC Client 5" name="clientid5">
  <authnService>oamsdktokenservice</authnService>
  </IdaasClient>
  ```

Mobile Services REST Reference: Authentication and Authorization

The cURL commands in this section show the REST calls used to request security tokens from the Mobile and Social server. Some REST calls use the POST method, whereas others use GET.

The following calls are demonstrated:

- Authentication for a Client Token
- Authentication for a User Token
- Authentication for an Access Token
- Get or Validate a (Client) Token
- Authorization

Authentication for a Client Token

Shows how to send the REST call to request a client token.

cURL Command

```
curl -H "Content-Type: application/json" --request POST
http://localhost:18001/idaas_rest/rest/tokenservice1/tokens
-d '{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"client1",
"X-Idaas-Rest-Subject-Password":"secret12",
"X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTTOKEN"}'
```

Expected Output

```
{"X-Idaas-Rest-Token-Value":"kubExOtDjCtL5Q0R1QhAgL5zNVmDFYKG1Y0AUe+P9HKvnz4gIDVx
YIMNxxyfJJpmkT5XtYKkDgW295juWEcK7c7LmPBkxE6MytcfvKh4HzWIUGEgS2uKej3PQJG49RpZ6UxAP
ZbGYWj7fpjZoqBhtPiCtyacI0C22bl2/DbbRCVx4341z68j5YiTgOklGC6lIucSorlM7pBI54bxygFZsr
F1DVKxL+RNhrobYsN6I7fFLR4fL+iO/BZcbwM/4SNDuCIC82eOxPI/mTcRraz0cLw9tcLbw7c11MjC2eu
EBSGUjGcNmxpbhiJIt7SIBzJczzNsaBnH+2fKx/VTeVVvGQgGAf19e5b1Drj5QyNhj2I=",
"X-Idaas-Rest-Token-Type":"CLIENTTOKEN",
"X-Idaas-Rest-User-Principal":"client-1",
"X-Idaas-Rest-Provider-Type":"OAM_11G"}
```

Comments

- A contract name can be specified as a query parameter or a header with a name of `X-IDAAS-CONTRACT`. Otherwise, Mobile and Social assumes the contract is a "default" contract.

Authentication for a User Token

Shows how to send a REST call requesting a User token.

cURL Command

```
curl -H "Content-Type: application/json"
--request POST http://localhost:18001/idaas_rest/rest/tokenservice1/tokens
-d '{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"tester1",
"X-Idaas-Rest-Subject-Password":"secret12",
"X-Idaas-Rest-New-Token-Type-To-Create":"USERTOKEN"}'
```

Expected Output

```
{"X-Idaas-Rest-Token-Value":"adc3bfbExOtDjCtL5Q0R1QhAgL5zNVmDFYKG1Y0AUe+P9HKvnz4g
IDVxYIMNxxyfJJpmkT5XtYKkDgW295juWEcK7c7LmPBkxE6MytcfvKh4HzWIUGEgS2uKej3PQJG49RpZ6
UxAPZbGYWj7fpjZoqBhtPiCtyacI0C22bl2/DbbRCVx4341z68j5YiTgOklGC6lIucSorlM7pBI54bxyg
FZsrF1DVKxL+RNhrobYsN6I7fFLR4fL+iO/BZcbwM/4SNDuCIC82eOxPI/mTcRraz0cLw9tcLbw7c11Mj
C2euEBSGUjGcNmxpbhiJIt7SIBzJczzNsaBnH+2fKx/VTeVVvGQgGAf19e5b1Drj5QyNhj2I=",
"X-Idaas-Rest-Token-Type":"USERTOKEN",
"X-Idaas-Rest-User-Principal":"user-1",
"X-Idaas-Rest-Provider-Type":"OAM_11G"}
```

Comments

- A contract name can be specified as a query parameter or a header with a name of `X-IDAAS-CONTRACT`. Otherwise, Mobile and Social assumes the contract is a "default" contract.

Authentication for an Access Token

Shows how to send a REST call requesting an access token.

cURL Command

```
curl -H "Content-Type: application/json"
--request POST http://localhost:18001/idaas_rest/rest/tokenservice1/tokens
-d '{
"X-Idaas-Rest-Subject-Type":"TOKEN",
"X-Idaas-Rest-Subject-Value":
"vTBI8jN8eYsmHCUzC4kITJ8xnv0WCcGaETq5OEco9lErOzn13EZIUpyKk4
xucZ9lfbe1b367GuCwPUceldPs8v8W4JtSWJO0IKhBc0o4EaBh4jZlRDcd9y
mNEbh%2BrdaG2rJ0%2BGFGglqe%2BNg0JUBVoKemdoe%2BylkQLj9RT4
yV%2FxOtI7eeXdIOtKD%2BURIIAp%2BZqeDmYHX%2BeTJqOi9dvhUv8b5
AniFPTX3xaO5%2Fovs8250aojYyz465Td0ZmchMyTn%2BSwSPoD81GEiV
MddlapJnred%2BVdStBv%2FHsawIgErLLsCNjlidd90z%2FDVGy31XSOuz3
GkwmE0BoM%2BSiAGZ5XFSQA%3D%3D",
"X-Idaas-Rest-Application-Context":"75sSbBZZKJiUOAWikZxsKA==",
"X-Idaas-Rest-Application-Resource":"http:/wengate123.us.example.com:7779/index.ht
ml",
"X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN"}'
```

Expected Output

```
{"X-Idaas-Rest-Token-Value":"R1QhAgL5zNVmDFYKG1Y0AUe+P9HKvnz4gIDVxYIMNxxyfJJpmkT5
XtYKkDgW295juWEcK7c7LmPBkxE6MytcfvKh4HzWIUGEgS2uKej3PQJG49RpZ6UxAPZbGYWj7fpjZoqBh
tPiCtyacI0C22bl2/DbbRCVx4341z68j5YiTgOklGC6lIucSorlM7pBI54bxygFZsrF1DVKxL+RNhrobY
sN6I7fFLR4fL+iO/BZcbwM/4SNDuCIC82eOxPI/mTcRraz0cLw9tcLbw7c11MjC2euEBSGUjGcNmxpbhi
JIt7SIBzJczzNsaBnH+2fKx/VTeVVvGQgGAf19e5b1Drj5QyNhj2I=",
"X-Idaas-Rest-Token-Type":"ACCESSTOKEN",
"X-Idaas-Rest-User-Principal":"user-1",
"X-Idaas-Rest-Provider-Type":"OAM_11G"}
```

Comments

- A contract name can be specified as a query parameter or a header with a name of `X-IDAAS-CONTRACT`. Otherwise, Mobile and Social assumes the contract is a "default" contract.

Get or Validate a (Client) Token

Shows how to send the REST call required to request (get) a client token.

cURL Command

```
curl --request GET http://localhost:18001/idaas_rest/
rest/mobilesecret1/tokens/info -H "X-Idaas-Rest-Subject: TOKEN someTokenValue"
```

Expected Output

```
{"X-Idaas-Rest-Token-Value":"QA8wjxWGSf3VMggfxFFYW4Yrre0DuG7hOagET4yfF3PX
bbUUsgh7uJUOEX5aZAQPsrV90J20gtALfhiUI32gbxooeqppGnQSLnk0ehpN4%2B6%2BCgR2nOMrYzoLi
U7%2FvrnoG7894eUfxHwmvZESQw4w4ez6L%2BOcaHF2tc05F4zkqi6%2BveSL4uFdiaMh9pJ2k%2BXF%2
FWn2Q8IfOWBdk2IzWeFhwi35CzMLJrNiAST%2BdMWhteIKcNEFbvS1WFaYR8Fjzx%2FpuU3%2FdTaG2gX
xDJxE%2BpI2bpanks4fdZwaFmkLCraUfJFdtiGgOk2SIVIwi4UYCBAbM9XZJ5nyjtmxpqEESKJSGQ%3D%
3D",
"X-Idaas-Rest-Token-Type":"USERTOKEN",
"X-Idaas-Rest-User-Principal":"testuser",
"X-Idaas-Rest-Provider-Type":"JWT"
}
```

Comments

- A contract name can be specified as a query parameter or a header with a name of `X-IDAAS-CONTRACT`. Otherwise, Mobile and Social assumes the contract is a "default" contract.

Authorization

Shows how to send the REST call required to request a client token.

cURL Command

```
curl --request GET "localhost:18001/idaas_rest/
rest/authorizationservice1/authorization?
resource=http://webgate123.us.example.com:7779/index.html&
action=GET&X-Idaas-Rest-Subject-Value=
ZNsJcMMM3ow83Zr5D8KqCPnhBGmui4RnBvUXJ5dqC7OfwZIv6FDcYWwf
PuHupxN%2Bfs5qN0I6AWIZBX%2F2KQNNQ5bPDN1XqeE8y7OPPoy4znte
EfCaRHb7UA1ia1ox%2BW85LbknXCLaZ5q%2FN4I0IcXP%2B13FGX9r9LR
OQ3OZZVNMLhfx3KabZcIVmSHBkK%2FARGYEJQv6RO%2FPCMN2YYTJ
gWxGr20rWeG8NLbzgN%2FPyADxxlPLvkxH2YCVHHH7bLBfOp3p83IbJ%2
FC%2Bm9sCd4YjlSlhsMUXKtvZ1LnJME4UymuR5tXuw2B0Yr25OHxUbMreI
GgRYZXFonmjhAovKhXqIgzpIg%3D%3D&
X-Idaas-Rest-Subject-Type=TOKEN"
```

Expected Output

```
{
"Allowed":"true"
}
```

Comments

- A contract name can be specified as a query parameter or a header with a name of `X-IDAAS-CONTRACT`. Otherwise, Mobile and Social assumes the contract is a "default" contract.

Mobile Services REST Reference: Commands for Mobile Single Sign-on Tokens

The cURL commands in this section show the REST calls that the mobile single sign-on agent sends to the Mobile and Social server to request client, user, and access tokens, and to create client registration handles.

The following calls are demonstrated:

- Create a Client Registration Handle for a Mobile Single Sign-on Agent App
- Create a Client Registration Handle for a Mobile Single Sign-on Client App (User Name Scenario)
- Create a Client Registration Handle for a Mobile Single Sign-on Client App (User Token Scenario)
- Create a Request for a User Token
- Create a Request for an Access Token
- The Single Sign-on Agent Request to Create an Access Token for its own use
- Verify a Client Reg Handle

Create a Client Registration Handle for a Mobile Single Sign-on Agent App

Shows how to create a client registration handle for a mobile single sign-on (SSO) agent app based on a user name and password. In this example, the mobile single sign-on agent app is named *MobileAgent1*.

cURL Command

```
curl -H "Content-Type: application/json" --request POST
http://localhost:18001/idaas_rest/rest/mobilejwtauthentication/register -d
'{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"theUserName",
"X-Idaas-Rest-Subject-Password":"thePassword",
"X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTREGHANDLE",
"deviceProfile" : { ... },
"clientId": "MobileAgent1" }'
```

Expected Output

```
{"X-Idaas-Rest-Token-Value":"eyJ0b2tl...",
"X-Idaas-Rest-Token-Type":"CLIENTREGHANDLE",
handles : {
"oaam.session" : { ... } ,
"oaam.device" : { ... }
}
}
```

Comments

- The value of `CLIENTREGHANDLE` is shortened for display purposes.
- The user name and password ("theUserName" and "thePassword" in this example) is a security credential that signifies an authenticated user authorized for such a device.

Create a Client Registration Handle for a Mobile Single Sign-on Client App (User Name Scenario)

This example shows how the mobile single sign-on agent creates a client registration handle for a mobile business app (the client app) utilizing a user name and password. In this example, the request originated with the mobile business app, which is named *MobileExpenseReport1*.

cURL Command

```
curl -H "Content-Type: application/json" --request POST
http://localhost:18001/idaas_rest/rest/mobilejwtauthentication/register -H
"X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD ..." -d
'{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"theUserName",
"X-Idaas-Rest-Subject-Password":"thePassword",
"X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTREGHANDLE",
"deviceProfile" : { ... },
handles : {
"oaam.session" : "...",
"oaam.device" : "..." },
"clientId": "MobileExpenseReport1" } '
```

Expected Output

```
{"X-Idaas-Rest-Token-Value":"ey...",
"X-Idaas-Rest-Token-Type":"CLIENTREGHANDLE",
handles : {
"oaam.session" : { ... } ,
"oaam.device" : { ... }
}
}
```

Comments

- The value of `CLIENTREGHANDLE` and other tokens is shortened for display purposes.
- If the `clientId` is not a mobile SSO agent (for example, `MobileExpenseReport1`), then the caller needs to add a header to the HTTP request that contains the client reg handle obtained previously for a Mobile Agent, for example `-H "X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD...."`

Create a Client Registration Handle for a Mobile Single Sign-on Client App (User Token Scenario)

This example is similar to the previous example. Instead of a user name and password, however, a user token is submitted. The user token is a security credential that signifies that an authenticated user authorized the device. As with the previous example, the request originated with the mobile business app, which is named *MobileExpenseReport1*.

cURL Command

```
curl -H "Content-Type: application/json" --request POST
http://localhost:18001/idaas_rest/rest/mobilejwtauthentication/register -H
"X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD ..." -d
'{
"X-Idaas-Rest-Subject-Type":"TOKEN",
"X-Idaas-Rest-Subject-Value":"ey...",
"X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTREGHANDLE",
"deviceProfile" : { ... },
handles : {
"oaam.session" : "...",
"oaam.device" : "..." },
"clientId": "MobileExpenseReport1" } '
```

Expected Output

```
{"X-Idaas-Rest-Token-Value":"ey...",
"X-Idaas-Rest-Token-Type":"CLIENTREGHANDLE",
handles : {
"oaam.session" : { ... } ,
"oaam.device" : { ... }
}
}
```

Comments

- The value of `CLIENTREGHANDLE` and other tokens is shortened for display purposes.
- When registering the client application, the user token can only represent a user registration if the `Mobile.reauthnForRegNewClientApp` configuration value is set to `false` in the corresponding mobile agent client application profile.
- The HTTP header `X-IDAAS-REST-AUTHORIZATION` has a `UIDPASSWORD` scheme value that contains the client reg handle of the mobile agent app (for example, MobileAgent1).

Create a Request for a User Token

This example shows the REST call that the mobile single sign-on agent sends to the Mobile and Social server to request that a user token be created.

cURL Command

```
curl -H "Content-Type: application/json" --request POST
http://localhost:18001/idaas_rest/rest/mobilejwtauthentication/authenticate -H
'X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred="..." ' -d
'{
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
"X-Idaas-Rest-Subject-Username":"theUserName",
"X-Idaas-Rest-Subject-Password":"thePassword",
"X-Idaas-Rest-New-Token-Type-To-Create":"USERTOKEN",
"handles" : { ... },
"deviceProfile" : { ... } }'
```

Expected Output

```
{"X-Idaas-Rest-Token-Value":"ey...",
"X-Idaas-Rest-Token-Type":"USERTOKEN",
handles : {
"oaam.session" : { ... } ,
"oaam.device" : { ... }
}
}
```

Comments

- Token values are shortened for display purposes.
- An SSO agent app (*MobileAgent1,* for example) requests a User token with a user name and password.
  The HTTP header `X-IDAAS-REST-AUTHORIZATION` has a `UIDPASSWORD` scheme value that contains the client reg handle of the SSO agent app (*MobileAgent1*).

Create a Request for an Access Token

This example shows a mobile SSO agent request for an access token on behalf of a mobile business app. The mobile SSO agent is named *MobileAgent1*, and the business app is named *MobileExpenseReport1*.

cURL Command

Mobile OAMAuthentication Example

```
curl -H "Content-Type: application/json" -H
'X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred="..." ' -H
'X-IDAAS-REST-AGENT-AUTHORIZATION: UIDPASSWORD cred="..." '
--request POST
http://localhost:18001/idaas_rest/rest/mobileoamauthentication/access -d
'{
"X-Idaas-Rest-Subject-Type":"TOKEN",
"X-Idaas-Rest-Subject-Value":"... USER TOKEN VALUE...",
"X-Idaas-Rest-Application-Context":"75sSbBZZKJiUOAWikZxsKA==",
"X-Idaas-Rest-Application-Resource":
"http:/wengate123.us.example.com:7779/index.html",
"X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN",
"handles" : { ... },
"deviceProfile" : { ... }
}'
```

Mobile JWTAuthentication Example

```
curl -H "Content-Type: application/json" -H
'X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred="..." ' -H
'X-IDAAS-REST-AGENT-AUTHORIZATION: UIDPASSWORD cred="..." '
--request POST
http://localhost:18001/idaas_rest/rest/mobilejwtauthentication/access -d
'{
"X-Idaas-Rest-Subject-Type":"TOKEN",
"X-Idaas-Rest-Subject-Value":"... USER TOKEN VALUE ...",
"X-Idaas-Rest-Application-Resource":"...",
"X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN",
"handles" : { ... },
"deviceProfile" : { ... }
}'
```

Expected Output

```
{"X-Idaas-Rest-Token-Value":"...",
"X-Idaas-Rest-Token-Type":"ACCESSTOKEN",
handles : {
"oaam.session" : { ... } ,
"oaam.device" : { ... }
}
}
```

Comments

- This HTTP request carries two headers: The first contains the client registration handle of the SSO Agent app, and the second contains the client registration handle of the Business app.

  The header `X-IDAAS-REST-AGENT-AUTHORIZATION` contains the client reg handle of the SSO agent app (*MobileAgent1*).

The header `X-IDAAS-REST-AUTHORIZATION` contains the client reg handle of the Business app (*MobileExpenseReport1*).

- The Mobile and Social server component (specifically, the Mobile Services component) will verify the validity of both handles. It will ensure both apps are listed in the target service domain. The underlying Token / Authentication Service will vend out an Access Token upon verifying the validity of the User Token Value.
- In the case of Access Manager, the `X-Idaas-Rest-Application-Resource` field refers to a resource protected by a particular WebGate. It also has an `X-Idaas-RESt-Application-Context` field that corresponds to the Access Manager Application Context.
- Token values are shortened for display purposes.

The Single Sign-on Agent Request to Create an Access Token for its own use

This example shows a mobile SSO agent request for an access token for its own use. The mobile SSO agent requires an access token before it can request tokens on behalf of client apps.

cURL Command

Mobile OAMAuthentication Example

```
curl -H "Content-Type: application/json" -H
'X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred="..." '
--request POST http://localhost:18001/idaas_
rest/rest/mobileoamauthentication/access -d
'{
"X-Idaas-Rest-Subject-Type":"TOKEN",
"X-Idaas-Rest-Subject-Value":"... USER TOKEN VALUE...",
"X-Idaas-Rest-Application-Context":"75sSbBZZKJiUOAWikZxsKA==",
"X-Idaas-Rest-Application-Resource":"http:/wengate123.us.example.com:7779/index.ht
ml",
"X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN",
"handles" : { ... },
"deviceProfile" : { ... }
}'
```

Mobile JWTAuthentication Example

```
curl -H "Content-Type: application/json" -H
'X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred="..." '
--request POST http://localhost:18001/idaas_
rest/rest/mobilejwtauthentication/access -d
'{
"X-Idaas-Rest-Subject-Type":"TOKEN",
"X-Idaas-Rest-Subject-Value":"... USER TOKEN VALUE ...",
"X-Idaas-Rest-Application-Resource":"...",
"X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN",
"handles" : { ... },
"deviceProfile" : { ... }
}'
```

Expected Output

```
{
"X-Idaas-Rest-Token-Value":"...",
"X-Idaas-Rest-Token-Type":"ACCESSTOKEN",
handles : {
"oaam.session" : { ... } ,
"oaam.device" : { ... }
}
}
```

Comments

- This HTTP request carries ONE header, `X-IDAAS-REST-AUTHORIZATION`, that contains the client reg handle of the SSO agent app (*MobileAgent1*).

  There is no `X-IDAAS-REST-AGENT-AUTHORIZATION` header in this request.

- The Mobile and Social server component (specifically, the Mobile Services component) will verify the validity of both handles. It will ensure that the

MobileAgent1 app is listed in the target service domain and that it is marked as an SSO-capable app (that is, the app is listed with an SSO Priority).

- Token values are shortened for display purposes.

Verify a Client Reg Handle

This example shows a client reg handle verification request. The Mobile and Social server has token and handle verification logic, so the mobile client does not need to make this verification call.

When the request is sent to the Mobile and Social server to create a User Token or an Access Token, the service verifies the one or two HTTP headers that contain the client reg handles: `X-IDAAS-REST-AUTHORIZATION` and `X-IDAAS-REST-AGENT-AUTHORIZATION`.

cURL Command

```
curl --request
GET http://localhost:18001/idaas_rest/rest/mobileservice1/tokens/info -H
"X-Idaas-Rest-Subject: TOKEN ey..." -H
"X-IDAAS-REST-AUTHORIZATION: TOKEN ey..."
```

Expected Output

```
{
"X-Idaas-Rest-Token-Value":"eyJl...",
"X-Idaas-Rest-Token-Type":"CLIENTREGHANDLE"
}
```

Comments

- The `CLIENTREGHANDLE` values are repeated under two different HTTP headers. If an administrator uses an explicit service binding not requiring a Client Token to perform a verify token operation, the second HTTP header can be dropped.
- The `CLIENTREGHANDLE` value is shortened for display purposes.
- Token values are shortened for display purposes.

Mobile Services REST Reference: Commands for User Profile Services

The cURL commands in this section show the REST calls that are sent from a client application to the Mobile and Social server to perform User Profile Services transactions with a connected Directory server.

User Profile cURL commands are grouped into the following sections:

Basic User Operations

Basic user operations commands include the following:

- Create a User
- Read a User
- Update a User
- Delete a User

Create a User

Shows how to create a user profile in a remote directory.

cURL Command

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{"uid":"John","description":"test user","lastname":"Anderson",
"commonname":"John Anderson","firstname":"John"}'
```

Expected Output

```
{"uid":"John","guid":"FE1D7BD0590111E1BFDCF77FB8E715D5","
description":"test user","name":"John","lastname":"Anderson",
"commonname":"John Anderson","loginid":"John","firstname":"John",
"uniquename":"FE1D7BD0590111E1BFDCF7FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/John"}
```

Read a User

Shows how to retrieve a user profile in a remote directory.

cURL Command

```
curl -i --request GET http://localhost:14100/idaas_rest/
rest/userprofile/people/John/
```

Expected Output

```
{"uid":"John","guid":"FE1D7BD0590111E1BFDCF77FB8E715D5","description":"test user",
"name":"John","lastname":"Anderson","commonname":"John Anderson","loginid":"John",
"firstname":"John","uniquename":"FE1D7BD0590111E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/John"}
```

Update a User

Shows how to update a user profile record in a remote directory.

cURL Command

```
curl -H "Content-Type: application/json" --request PUT
http://localhost:14100/idaas_rest/rest/userprofile/people/John/ -d
'{"description":"test user1"}'
```

Expected Output

```
{"uid":"John","guid":"FE1D7BD0590111E1BFDCF77FB8E715D5",
"description":"test user1","name":"John","lastname":"Anderson",
```

"commonname":"John Anderson","loginid":"John","firstname":"John",
"uniquename":"FE1D7BD0590111E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/John"}

Delete a User

Shows how to remove a user profile record in a remote directory.

cURL Command

```
curl -i --request DELETE http://localhost:14100/
idaas_rest/rest/userprofile/people/John/
```

Expected Output

No response.

Basic Group Operations

Basic group operations commands include the following:

- Create a Group
- Read a Group
- Update a Group
- Delete a Group

Create a Group

Shows how to create a group profile in a remote directory.

cURL Command

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"group1 testuing","commonname":"group1"}'
```

Expected Output

```
{"guid":"2259C6C0592011E1BFDCF77FB8E715D5","description":"group1 testing",
"name":"group1","commonname":"group1",
"uniquename":"2259C6C0592011E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1"}
```

Read a Group

Shows how to retrieve a group profile in a remote directory.

cURL Command

```
curl -i --request GET "http://localhost:14100/idaas_rest/
rest/userprofile/groups/group1/"
```

Expected Output

```
{"guid":"2259C6C0592011E1BFDCF77FB8E715D5","description":"group1 testing",
"name":"group1","commonname":"group1",
"uniquename":"2259C6C0592011E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1"}
```

Update a Group

Shows how to update a group profile in a remote directory.

cURL Command

```
curl -H "Content-Type: application/json" --request PUT
http://localhost:14100/idaas_rest/rest/userprofile/groups/group1/ -d
'{"description":"group11 testing"}'
```

Expected Output

```
{"guid":"2259C6C0592011E1BFDCF77FB8E715D5","description":"group11 testing",
"name":"group1","commonname":"group1",
"uniquename":"2259C6C0592011E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1"}
```

Delete a Group

Shows how to delete a group profile in a remote directory.

cURL Command

```
curl -H "Content-Type: application/json" --request PUT
http://localhost:14100/idaas_rest/rest/userprofile/groups/group1/ -d
'{"description":"group11 testing"}'
```

Expected Output

```
{"guid":"2259C6C0592011E1BFDCF77FB8E715D5","description":"group11 testing",
"name":"group1","commonname":"group1",
"uniquename":"2259C6C0592011E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1"}
```

"memberOf" Relationship Operations

The "members" and "memberOf" logical entity relationships both point to the same "member" attribute in the LDAP "group" entity. Both logical entity relationships can be used to add, delete, read, and search a user with respect to a group.

This section includes the following operations:

- Create a "memberOf" Relationship
- Read a "memberOf" Relationship
- Delete a "memberOf" Relationship

Create a "memberOf" Relationship

Shows how to make a user a member of a group.

cURL Command

Create User "John"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{"uid":"John"Anderson","commonname":"John Anderson","firstname":"John"}'
```

Create Group "Group1"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"group1 testing","commonname":"group1"}'
```

Create a MemberOf Relationship

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/John/memberOf/ -d
'{"group-uri":"\/idaas_rest\/rest\/userprofile\/group\/group1",
"person-uri":"\/idaas_rest\/rest\/userprofile\/people\/John"}'
```

Expected Output

```
{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1",
"person-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/John\/memberOf\/group1"}
```

Read a "memberOf" Relationship

Shows how to retrieve a "memberOf" relationship profile for the specified user.

cURL Command

```
curl -i --request GET "http://localhost:14100/idaas_rest
/rest/userprofile/people/John/memberOf/group1/"
```

Expected Output

Either of the following:

- HTTP Status 200 (The request has succeeded.)
- No response.

Delete a "memberOf" Relationship

Shows how to delete a "memberOf" relationship.

cURL Command

Delete the MemberOf Relationship

```
curl -i --request DELETE "http://localhost:14100/idaas_rest/
rest/userprofile/people/John/memberOf/group1/"
```

Delete User "John"

```
curl -i --request DELETE http://localhost:14100/idaas_rest/
rest/userprofile/people/John/
```

Delete the Group "group1"

```
curl -i --request DELETE "http://localhost:14100/idaas_rest/
rest/userprofile/groups/group1"
```

Expected Output

Either of the following:

- HTTP Status 200 (The request has succeeded.)
- No response.

"members" Relationship Operations

The "members" and "memberOf" logical entity relationships both point to the same "member" attribute in the LDAP "group" entity. Both logical entity relationships can be used to add, delete, read, and search a user with respect to a group.

This section includes the following operations:

Create a "members" Relationship

Shows how to assign a user to a group.

cURL Command

Create User "John"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{"uid":"John"Anderson","commonname":"John Anderson","firstname":"John"}'
```

Create Group "Group1"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"group1 testuing","commonname":"group1"}'
```

Create a Members Relationship

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/group1/members -d
'{"group-uri":"\/idaas_rest\/rest\/userprofile\/group\/group1",
"person-uri":"\/idaas_rest\/rest\/userprofile\/people\/John"}'
```

Expected Output

```
{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1",
"person-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/group1\/members\/John"}
```

Read a "members" Relationship

Shows how to read a "members" relationship.

cURL Command

```
curl -i --request GET "http://localhost:14100/idaas_rest/
rest/userprofile/people/group1/members/John"
```

Expected Output

```
{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1",
"person-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/group1\/members\/John"}
```

Delete a "members" Relationship

Shows how to delete a "members" relationship profile.

cURL Command

Delete the Members Relationship

```
curl -i --request DELETE "http://localhost:14100/idaas_rest/
rest/userprofile/people/group1/members/John/"
```

Delete User "John"

```
curl -i --request DELETE http://localhost:14100/idaas_rest/
rest/userprofile/people/John/
```

Delete Group "Group1"

```
curl -i --request DELETE "http://localhost:14100/idaas_rest/
rest/userprofile/groups/group1/"
```

Expected Output

HTTP Status 200 (The request has succeeded.)

"manager" Relationship Operations

This section includes the following operations:

- Create a "manager" Relationship
- Read a "manager" Relationship
- Delete a "manager" Relationship

Create a "manager" Relationship

Shows how to assign a manager to a user.

cURL Command

Create User "John"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{"uid":"John"Anderson","commonname":"John Anderson","firstname":"John"}'
```

Create User "Alan"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{"uid":"Alan","description":"Manager User","lastname":"Doe",
"commonname":"Alan Doe","firstname":"Alan"}'
```

Create a Manager Relationship

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/John/manager/ -d
'{"report-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"manager-uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan"}'
```

Expected Output

```
{"report-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/John\/manager\/Alan",
"manager-uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan"}
```

Read a "manager" Relationship

Shows how to read a *manager* relationship profile.

cURL Command

```
curl -i --request GET "http://localhost:14100/idaas_rest/
rest/userprofile/people/John/manager/Alan"
```

Expected Output

```
{"report-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/John\/manager\/Alan",
"manager-uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan"}
```

Delete a "manager" Relationship

Shows how to delete the manager relationship.

cURL Command

Delete the Manager Relationship

```
curl -i --request DELETE "http://localhost:14100/
idaas_rest/rest/userprofile/people/John/manager/Alan"
```

Delete User "John"

```
curl -i --request DELETE http://localhost:14100/
idaas_rest/rest/userprofile/people/John/
```

Delete User "Alan"

```
curl -i --request DELETE "http://localhost:14100/
idaas_rest/rest/userprofile/people/Alan/"
```

Expected Output

No response.

"reports" Relationship Operations

This section includes the following operations:

- Create a "reports" Relationship
- Read a "reports" Relationship
- Delete a "reports" Relationship

Create a "reports" Relationship

Shows how to create a reports-to relationship.

cURL Command

Create User "John"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{"uid":"John"Anderson","commonname":"John Anderson","firstname":"John"}'
```

Create User "Alan"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{"uid":"Alan","description":"Manager User","lastname":"Doe",
"commonname":"Alan Doe","firstname":"Alan"}'
```

Create a Reports Relationship

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/Alan/reports/ -d
'{"report-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"manager-uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan"}'
```

Expected Output

```
{"report-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan\/reports\/John",
"manager-uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan"}
```

Read a "reports" Relationship

Shows how to read a reports-to relationship.

cURL Command

```
curl -i --request GET "http://localhost:14100/idaas_rest/
rest/userprofile/people/Alan/reports/John"
```

Expected Output

```
{"report-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan\/reports\/John",
"manager-uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan"}
```

Delete a "reports" Relationship

Shows how to delete a reports-to relationship.

cURL Command

Delete the Reports Relationship

```
curl -i --request DELETE "http://localhost:14100/idaas_rest/
rest/userprofile/people/Alan/reports/John"
```

Delete User "John"

```
curl -i --request DELETE http://localhost:14100/idaas_rest/
rest/userprofile/people/John/
```

Delete User "Alan"

```
curl -i --request DELETE "http://localhost:14100/idaas_rest/
rest/userprofile/people/Alan/"
```

Expected Output

No response.

"ownerOf" Relationship Operations

This section includes the following operations:

- Create an "OwnerOf" Relationship
- Read an "OwnerOf" Relationship
- Delete an "OwnerOf" Relationship

Create an "OwnerOf" Relationship

Shows how to create an ownerOf relationship.

cURL Command

Create User "John"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{"uid":"John"Anderson","commonname":"John Anderson","firstname":"John"}'
```

Create Group "group1"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"group1 testuing","commonname":"group1"}'
```

Create an "ownerOf" Relationship

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/John/ownerOf/ -d
'{"group-uri":"\/idaas_rest\/rest\/userprofile\/group\/group1",
"owner-uri":"\/idaas_rest\/rest\/userprofile\/people\/John"}'
```

Expected Output

```
{"report-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan\/reports\/John",
"manager-uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan"}
```

Read an "OwnerOf" Relationship

Shows how to read an ownerOf relationship.

cURL Command

```
curl -i --request GET "http://localhost:14100/idaas_rest/
rest/userprofile/people/John/ownerOf/group1"
```

Expected Output

```
{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1",
"owner-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/John\/ownerOf\/group1"}
```

Delete an "OwnerOf" Relationship

Shows how to delete an ownerOf relationship.

cURL Command

Delete the "ownerOf" Relationship

```
curl -i --request DELETE "http://localhost:14100/idaas_rest/
```

```
rest/userprofile/people/John/ownerOf/group1"
```

Delete User "John"

```
curl -i --request DELETE http://localhost:14100/idaas_rest/
rest/userprofile/people/John/
```

Delete Group "group1"

```
curl -i --request DELETE "http://localhost:14100/idaas_rest/
rest/userprofile/groups/group1"
```

Expected Output

No response.

"personOwner" Relationship Operations

This section includes the following operations:

- Create an "OwnerOf" Relationship
- Read an "OwnerOf" Relationship
- Delete an "OwnerOf" Relationship

Create a "personOwner" Relationship

Shows how to create a personOwner relationship.

cURL Command

Create User "John"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{"uid":"John"Anderson","commonname":"John Anderson","firstname":"John"}'
```

Create Group "group1"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"group1 testing","commonname":"group1"}'
```

Create a "personOwner" Relationship

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/group1/personOwner -d
'{"group-uri":"\/idaas_rest\/rest\/userprofile\/group\/group1",
"owner-uri":"\/idaas_rest\/rest\/userprofile\/people\/John"}'
```

Expected Output

```
{"report-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan\/reports\/John",
"manager-uri":"\/idaas_rest\/rest\/userprofile\/people\/Alan"}
```

Read a "personOwner" Relationship

Shows how to read a personOwner relationship.

cURL Command

```
curl -i --request GET "http://localhost:18001/idaas_rest/
rest/userprofile/groups/group1/personOwner/John"
```

Expected Output

```
{"owner-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1\/personOwner\/John"
```

Delete a "personOwner" Relationship

Shows how to delete a personOwner relationship.

cURL Command

Delete the "personOwner" Relationship

```
curl -i --request DELETE "http://localhost:18001/idaas_rest/
```

```
rest/userprofile/groups/group1/personOwner/John"
```

Delete User "John"

```
curl -i --request DELETE http://localhost:14100/idaas_rest/
rest/userprofile/people/John/
```

Delete Group "group1"

```
curl -i --request DELETE "http://localhost:14100/idaas_rest/
rest/userprofile/groups/group1/"
```

Expected Output

No response.

"groupOwner" Relationship Operations

This section includes the following operations:

- Create a "groupOwner" Relationship
- Read a "groupOwner" Relationship
- Delete a "groupOwner" Relationship

Create a "groupOwner" Relationship

Shows how to create a groupOwner relationship.

cURL Command

Create Group "XYZ"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"XYZ Group","commonname":"XYZ"}'
```

Create Group "ABC"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"ABC Group","commonname":"ABC"}'
```

Create a "groupOwner" Relationship

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/XYZ/groupOwner -d
'{"group-uri":"\/idaas_rest\/rest\/userprofile\/group\/XYZ",
"owner-uri":"\/idaas_rest\/rest\/userprofile\/group\/ABC"}'
```

Expected Output

```
{"owner-uri":"\/idaas_rest\/rest\/userprofile\/groups\/ABC",
"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ\/groupOwner\/ABC"}
```

Read a "groupOwner" Relationship

Shows how to read a groupOwner relationship.

cURL Command

```
curl -i --request GET "http://localhost:14100/
idaas_rest/rest/userprofile/groups/XYZ/groupOwner/ABC"
```

Expected Output

```
{"owner-uri":"\/idaas_rest\/rest\/userprofile\/people\/John",
"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/group1\/personOwner\/John"
```

Delete a "groupOwner" Relationship

Shows how to delete a groupOwner relationship.

cURL Command

Delete the "groupOwner" Relationship

```
curl -i --request DELETE "http://localhost:14100/
```

```
idaas_rest/rest/userprofile/groups/XYZ/groupOwner/ABC"
```

Delete Group "XYZ"

```
curl -i --request DELETE http://localhost:14100/
idaas_rest/rest/userprofile/groups/XYZ/
```

Delete Group "ABC"

```
curl -i --request DELETE "http://localhost:14100/
idaas_rest/rest/userprofile/groups/ABC/"
```

Expected Output

No response.

"groupOwnerOf" Relationship Operations

This section includes the following operations:

- Create a "groupOwnerOf" Relationship
- Read a "groupOwnerOf" Relationship
- Delete a "groupOwnerOf" Relationship

Create a "groupOwnerOf" Relationship

Shows how to create a groupOwnerOf relationship.

cURL Command

Create Group "XYZ"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"XYZ Group","commonname":"XYZ"}'
```

Create Group "ABC"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"ABC Group","commonname":"ABC"}'
```

Create a "groupOwnerOf" Relationship

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ABC/groupOwnerOf -d
'{"group-uri":"\/idaas_rest\/rest\/userprofile\/group\/XYZ",
"owner-uri":"\/idaas_rest\/rest\/userprofile\/group\/ABC"}'
```

Expected Output

```
{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ",
"owner-uri":"\/idaas_rest\/rest\/userprofile\/groups\/ABC",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/ABC\/groupOwnerOf\/XYZ"}
```

Read a "groupOwnerOf" Relationship

Shows how to read a groupOwnerOf relationship.

cURL Command

```
curl -i --request GET "http://localhost:14100/
idaas_rest/rest/userprofile/groups/ABC/groupOwnerOf/XYZ"
```

Expected Output

```
{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ",
"owner-uri":"\/idaas_rest\/rest\/userprofile\/groups\/ABC",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/ABC\/groupOwnerOf\/XYZ"
```

Delete a "groupOwnerOf" Relationship

Shows how to delete a groupOwnerOf relationship.

cURL Command

Delete the "groupOwnerOf" Relationship

```
curl -i --request DELETE "http://localhost:14100/
```

```
idaas_rest/rest/userprofile/groups/ABC/groupOwnerOf/XYZ"
```

Delete Group "XYZ"

```
curl -i --request DELETE http://localhost:14100/
idaas_rest/rest/userprofile/groups/XYZ/
```

Delete Group "ABC"

```
curl -i --request DELETE "http://localhost:14100/
idaas_rest/rest/userprofile/groups/ABC/"
```

Expected Output

No response.

"groupMemberOf" Relationship Operations

This section includes the following operations:

- Create a "groupMemberOf" Relationship
- Read a "groupMemberOf" Relationship
- Delete a "groupMemberOf" Relationship

Create a "groupMemberOf" Relationship

Shows how to create a groupMemberOf relationship.

cURL Command

Create Group "XYZ"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"XYZ Group","commonname":"XYZ"}'
```

Create Group "iCloud"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"iCloud Group","commonname":"iCLOUD"}'
```

Create a "groupMemberOf" Relationship

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/XYZ/groupMemberOf -d
'{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/iCLOUD",
"member-uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ"}'
```

Expected Output

```
{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/iCLOUD",
"member-uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ\/groupMemberOf\/iCLOUD"}
```

Read a "groupMemberOf" Relationship

Shows how to read a groupMemberOf relationship.

cURL Command

```
curl -i --request GET "http://localhost:14100/
idaas_rest/rest/userprofile/groups/XYZ/groupMemberOf/iCLOUD"
```

Expected Output

```
{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/iCLOUD",
"member-uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ\/groupMemberOf\/iCLOUD"
```

Delete a "groupMemberOf" Relationship

Shows how to delete a groupMemberOf relationship.

cURL Command

Delete the "groupMemberOf" Relationship

```
curl -i --request DELETE "http://localhost:14100/
```

```
idaas_rest/rest/userprofile/groups/XYZ/groupMemberOf/iCLOUD"
```

Delete Group "XYZ"

```
curl -i --request DELETE http://localhost:14100/
idaas_rest/rest/userprofile/groups/XYZ/
```

Delete Group "iCLOUD"

```
curl -i --request DELETE "http://localhost:14100/
idaas_rest/rest/userprofile/groups/iCLOUD/"
```

Expected Output

No response.

"groupMembers" Relationship Operations

This section includes the following operations:

- Create a "groupMembers" Relationship
- Read a "groupMembers" Relationship
- Delete a "groupMembers" Relationship

Create a "groupMembers" Relationship

Shows how to create a groupMembers relationship.

cURL Command

Create Group "XYZ"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"XYZ Group","commonname":"XYZ"}'
```

Create Group "iCloud"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/ -d
'{"description":"iCloud Group","commonname":"iCLOUD"}'
```

Create a "groupMembers" Relationship

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/groups/iCLOUD/groupMembers -d
'{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/iCLOUD",
"member-uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ"}'
```

Expected Output

```
{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/iCLOUD",
"member-uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/iCLOUD\/groupMembers\/XYZ"}
```

Read a "groupMembers" Relationship

Shows how to read a groupMembers relationship.

cURL Command

```
curl -i --request GET "http://localhost:14100/
idaas_rest/rest/userprofile/groups/iCLOUD/groupMembers"
```

Expected Output

```
{"group-uri":"\/idaas_rest\/rest\/userprofile\/groups\/iCLOUD",
"member-uri":"\/idaas_rest\/rest\/userprofile\/groups\/XYZ",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/iCLOUD\/groupMemberOf\/XYZ"
```

Delete a "groupMembers" Relationship

Shows how to delete a groupMembers relationship.

cURL Command

Delete the "groupMembers" Relationship

```
curl -i --request DELETE "http://localhost:14100/idaas_rest/rest/
```

```
userprofile/groups/iCLOUD/groupMembers"
```

Delete Group "XYZ"

```
curl -i --request DELETE http://localhost:14100/
idaas_rest/rest/userprofile/groups/XYZ/
```

Delete Group "iCLOUD"

```
curl -i --request DELETE "http://localhost:14100/
idaas_rest/rest/userprofile/groups/iCLOUD/"
```

Expected Output

No response.

Search User Operations

This section includes the following operations:

- Search Users
- Search Users With PageSize and PagePos
- Search Users With a Search Parameter and Without a Search Filter
- Search Users With a Search Filter
- Search Groups
- Search Relationships

Search Users

Shows how to get a list of all users.

cURL Command

```
curl -i --request GET http://localhost:14100/idaas_rest/rest/userprofile/people
```

Expected Output

```
{"next":"\/idaas_rest\/rest\/userprofile\/people?pageSize=10&pagePos=1",
"elements":[{"uid":"OracleSystemUser","guid":"E9A3B390581611E19F08FB1E3902A71C",
"description":"Oracle]]]] application software system user.",
"name":"OracleSystemUser","lastname":"OracleSystemUser",
"commonname":"OracleSystemUser","loginid":"OracleSystemUser",
"uniquename":"E9A3B390581611E19F08FB1E3902A71C",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/OracleSystemUser"},
{"uid":"weblogic","guid":"E9A4C500581611E19F08FB1E3902A71C",
"description":"This user is the default administrator.","name":"weblogic",
"lastname":"weblogic","commonname":"weblogic","loginid":"weblogic",
"uniquename":"E9A4C500581611E19F08FB1E3902A71C",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/weblogic"},
{"uid":"alice","guid":"D8D1907158F511E1BFDCF77FB8E715D5",
"description":"This test user is alice.","name":"alice","lastname":"alice",
"commonname":"alice","loginid":"alice",
"uniquename":"D8D1907158F511E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/alice"},
{"uid":"sean","guid":"D8D5AF2058F511E1BFDCF77FB8E715D5",
"description":"This test user is sean.","name":"sean","lastname":"sean",
"commonname":"sean","loginid":"sean",
"uniquename":"D8D5AF2058F511E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/sean"},
{"uid":"wei","guid":"D8D6245058F511E1BFDCF77FB8E715D5",
"description":"This test user is wei.","name":"wei","lastname":"wei",
"commonname":"wei","loginid":"wei",
"uniquename":"D8D6245058F511E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/wei"},
{"uid":"malla","guid":"D8D64B6058F511E1BFDCF77FB8E715D5",
"description":"This test user is malla.","name":"malla","lastname":"malla",
"commonname":"malla","loginid":"malla",
"uniquename":"D8D64B6058F511E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/malla"},
{"uid":"alan","guid":"D8D6998058F511E1BFDCF77FB8E715D5",
"description":"This test user is alan.","name":"alan","lastname":"alan",
"commonname":"alan","loginid":"alan",
"uniquename":"D8D6998058F511E1BFDCF77FB8E715D5",
```

```
"uri":"\/idaas_rest\/rest\/userprofile\/people\/alan"},
"uri":"\/idaas_rest\/rest\/userprofile\/people?pageSize=10&pagePos=0"}
```

Search Users With PageSize and PagePos

Shows how to get a list of users while specifying a page size and the page position.

cURL Command

```
curl -i --request GET "http://localhost:14100/
idaas_rest/rest/userprofile/people?pagePos=0&pageSize=1"
```

Expected Output

```
{"next":"\/idaas_rest\/rest\/userprofile\/people?pageSize=1&pagePos=1",
"elements":[{"uid":"OracleSystemUser","guid":"E9A3B390581611E19F08FB1E3902A71C",
"description":"Oracle]] application software system user.",
"name":"OracleSystemUser","lastname":"OracleSystemUser",
"commonname":"OracleSystemUser","loginid":"OracleSystemUser",
"uniquename":"E9A3B390581611E19F08FB1E3902A71C",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/OracleSystemUser"}],
"uri":"\/idaas_rest\/rest\/userprofile\/people?pageSize=1&pagePos=0"}
```

Search Users With a Search Parameter and Without a Search Filter

Shows how to get a list of users while specifying a search parameter but not a search filter.

cURL Command

```
curl -i --request GET "http:/localhost:14100/idaas_rest/rest/userprofile/people/
?pagePos=0&pageSize=10&searchparam.name=John*"
```

Expected Output

```
{"elements":[{"uid":"John","guid":"E932E4F0590911E1BFDCF77FB8E715D5",
"description":"test user","name":"John","lastname":"Anderson",
"commonname":"John Anderson","loginid":"John","firstname":"John",
"uniquename":"E932E4F0590911E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/John"}],
"uri":"\/idaas_rest\/rest\/userprofile\/people?pageSize=10
&searchparam.name=John+Anderson&pagePos=0"}
```

Search Users With a Search Filter

Shows how to get a list of users while specifying the default "out-of-the-box" simple AND search filter.

cURL Command

```
curl -i --request GET "http:/localhost:14100/idaas_rest/rest/userprofile/
people?searchFilter=SimpleOR&searchparam.uid=John&searchparam.lastname=TEST"
```

Expected Output

```
{"elements":[{
"uid":"John",
"guid":"E932E4F0590911E1BFDCF77FB8E715D5",
"description":"test user",
"name":"John",
"lastname":"Anderson",
"commonname":"John Anderson",
```

```
"loginid":"John",
"firstname":"John",
"uniquename":"E932E4F0590911E1BFDCF77FB8E715D5",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/John"}],
"uri":"\/idaas_rest\/rest\/userprofile\/people?pageSize=10
&searchFilter=SimpleOR&searchparam.lastname=TEST&searchparam.uid=John&pagePos=0"}
```

Search Groups

Shows how to get Group information.

cURL Command

```
curl -i --request GET "http:/localhost:14100/idaas_rest/rest/userprofile/
groups/?pagePos=0&pageSize=2"
```

Expected Output

```
{"next":"\/idaas_rest\/rest\/userprofile\/groups?pageSize=2&pagePos=1",
"elements":[{
"guid":"7CF7EC60724811E1BFB5AB6A1E4E415B",
"description":"AdminChannelUsers]] can access the admin channel.",
"name":"AdminChannelUsers",
"commonname":"AdminChannelUsers",
"uniquename":"7CF7EC60724811E1BFB5AB6A1E4E415B",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/AdminChannelUsers"},
{"guid":"7CF7EC61724811E1BFB5AB6A1E4E415B",
"description":"Administrators can view and modify all resource attributes and
start and stop servers.",
"name":"Administrators",
"commonname":"Administrators",
"uniquename":"7CF7EC61724811E1BFB5AB6A1E4E415B",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/Administrators"}],
"uri":"\/idaas_rest\/rest\/userprofile\/groups?pageSize=2&pagePos=0"}
```

Search Relationships

Given the name of a person in an organization, allows you to search for the person's manager.

cURL Command

```
curl -i --request GET "http:/localhost:14100/idaas_rest/rest/userprofile/
people/JohnD/manager/?pagePos=0&pageSize=2"
```

Expected Output

```
{"elements":[{
"report-uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD\/manager\/SusanS",
"manager-uri":{
   "uid":"SusanS",
   "manager":"\/idaas_rest\/rest\/userprofile\/people\/SusanS\/manager",
   "state":"CA",
   "lastname":"Smith",
   "firstname":"Susan",
   "loginid":"SusanS",
   "uniquename":"5B543C30790511E1AF41BD17BAB1A1C1",
   "uri":"\/idaas_rest\/rest\/userprofile\/people\/SusanS",
   "country":"USA",
```

```
   "guid":"5B543C30790511E1AF41BD17BAB1A1C1",
   "title":"Sr]]. Director, Development ",
   "name":"SusanS",
   "commonname":"Susan Smith"}
}],
"uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD\/manager?pageSize=2
&pagePos=0"}
```

The "attrsToFetch" Query Parameter Feature

Use the `attrsToFetch` query parameter to retrieve a specific set of attributes instead of the full set of attributes that the system returns otherwise. To specify multiple attributes use a comma-separated list of attribute names.

For example:

```
.../people/alice?attrsToFetch=uid,email
```

The `attrsToFetch` query parameter can be used with any Search, Read, User, Group, or Relationship operation.

This section includes the following examples:

- Read a User With attrsToFetch
- Search Groups With attrsToFetch
- Search a Relationship With attrsToFetch

Read a User With attrsToFetch

This example shows how to retrieve the User's common name only. Without the `attrsToFetch` parameter, the system would retrieve the full set of User attributes.

cURL Command

```
curl -i --request GET
"http://host:10/idaas_rest/rest/userprofile/people/Alice/?attrsToFetch=commonname"
```

Expected Output With attrsToFetch

```
{
"commonname":"Alice Mac",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/Alice"}
```

Expected Output Without attrsToFetch

```
{
"uid":"Alice",
"guid":"C04020C078FE11E1AF41BD17BAB1A1C1",
"description":"Alice User",
"name":"Alice",
"lastname":"Mac",
"commonname":"Alice Mac",
"loginid":"Alice",
"firstname":"Alice",
"uniquename":"C04020C078FE11E1AF41BD17BAB1A1C1",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/Alice"}
```

Search Groups With attrsToFetch

This example shows how to search Groups and retrieve only the name of each Group. Without the `attrsToFetch` parameter, the system would retrieve every attribute of each Group.

cURL Command

```
curl -i --request GET
"http:/host:10/idaas_rest/rest/userprofile/groups?pagePos=0&pageSize=2
&attrsToFetch=name"
```

Expected Output With attrsToFetch

```
{"next":
```

```
"\/idaas_rest\/rest\/userprofile\/groups?pageSize=2&attrsToFetch=name&pagePos=1",
"elements":[{
"name":"AdminChannelUsers",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/AdminChannelUsers"},
{
"name":"Administrators",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/Administrators"
}],
"uri":"\/idaas_rest\/rest\/userprofile\/groups?pageSize=2&attrsToFetch=name
&pagePos=0"}
```

Expected Output Without attrsToFetch

```
{"next":
"\/idaas_rest\/rest\/userprofile\/groups?pageSize=2&pagePos=1",
"elements":[{
"guid":"7CF7EC60724811E1BFB5AB6A1E4E415B",
"description":"AdminChannelUsers can access the admin channel.",
"name":"AdminChannelUsers",
"commonname":"AdminChannelUsers",
"uniquename":"7CF7EC60724811E1BFB5AB6A1E4E415B",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/AdminChannelUsers"},
{
"guid":"7CF7EC61724811E1BFB5AB6A1E4E415B",
"description":"Administrators can view and modify all resource attributes and
start and stop servers.",
"name":"Administrators",
"commonname":"Administrators",
"uniquename":"7CF7EC61724811E1BFB5AB6A1E4E415B",
"uri":"\/idaas_rest\/rest\/userprofile\/groups\/Administrators"
}],
"uri":"\/idaas_rest\/rest\/userprofile\/groups?pageSize=2&pagePos=0"}
```

Search a Relationship With attrsToFetch

This example shows how to retrieve the name of the Groups that a User is a member of. Without the `attrsToFetch` parameter, the system would retrieve the full set of Group attributes for each Group.

cURL Command

```
curl -i --request GET
"http://host:10/idaas_rest/rest/userprofile/people/weblogic/memberOf?
pagePos=0&pageSize=2&attrsToFetch=name"
```

Expected Output With attrsToFetch

```
{"next":
"\/idaas_rest\/rest\/userprofile\/people\/weblogic\/memberOf?
pageSize=2&attrsToFetch=name&pagePos=1",
"elements":[
   {
   "group-uri":
     {
      "name":"Administrators",
      "uri":"\/idaas_rest\/rest\/userprofile\/groups\/Administrators"
     },
   "person-uri":"\/idaas_rest\/rest\/userprofile\/people\/weblogic",
   "uri":"\/idaas_rest\/rest\/userprofile\/people\/weblogic\/memberOf\/
          Administrators"
   },
   {
```

```
   "group-uri":
     {
      "name":"OAAMEnvAdminGroup",
      "uri":"\/idaas_rest\/rest\/userprofile\/groups\/OAAMEnvAdminGroup"
     },
   "person-uri":"\/idaas_rest\/rest\/userprofile\/people\/weblogic",
   "uri":"\/idaas_ rest\/rest\/userprofile\/people\/weblogic\/memberOf\/
  OAAMEnvAdminGroup"
}],
"uri":"\/idaas_rest\/rest\/userprofile\/people\/weblogic\/memberOf?
pageSize=2&attrsToFetch=name&pagePos=0"}
```

Expected Output Without attrsToFetch

```
{"next":
"\/idaas_rest\/rest\/userprofile\/people\/weblogic\/memberOf?
pageSize=2&pagePos=1",
"elements":[
   {
   "group-uri":
     {
      "guid":"7CF7EC61724811E1BFB5AB6A1E4E415B",
      "description":"Administrators can view and modify all resource attributes
                     and start and stop servers.",
      "name":"Administrators",
     "commonname":"Administrators",
     "uniquename":"7CF7EC61724811E1BFB5AB6A1E4E415B",
     "uri":"\/idaas_rest\/rest\/userprofile\/groups\/Administrators"
     },
   "person-uri":"\/idaas_rest\/rest\/userprofile\/people\/weblogic",
   "uri":"\/idaas_rest\/rest\/userprofile\/people\/weblogic\/memberOf\/
         Administrators"
   },
   {
   "group-uri":
    {
     "guid":"7CF83A81724811E1BFB5AB6A1E4E415B",
     "description":"EnvAdminGroup",
     "name":"OAAMEnvAdminGroup",
     "commonname":"OAAMEnvAdminGroup",
     "uniquename":"7CF83A81724811E1BFB5AB6A1E4E415B",
     "uri":"\/idaas_rest\/rest\/userprofile\/groups\/OAAMEnvAdminGroup"
     },
   "person-uri":"\/idaas_rest\/rest\/userprofile\/people\/weblogic",
   "uri":"\/idaas_rest\/rest\/userprofile\/people\/weblogic\/memberOf\/
         OAAMEnvAdminGroup"
   }],
"uri":"\/idaas_rest\/rest\/userprofile\/people\/weblogic\/memberOf?
pageSize=2&pagePos=0"}
```

The "prefetch" Query Parameter Feature

Use the `prefetch` query parameter to expand a query to retrieve a collection of attributes linked to the User or Group or Relationship that is the subject of the query. To specify multiple attributes use a comma-separated list of attribute names.

For example:

```
.../people/alice?prefetch=attr1,attr2(b1,b2),attr3(b1,b2,b3)
```

If you do not specify the prefetch query parameter, the system returns the requested URI only.

You can use the `prefetch` query parameter with any User, Group, or Relationship profile operation, but not a Search operation.

So for example, you can use `prefetch` with instance resources such as the following:

- `.../people/alice`
- `.../groups/Admin`
- `.../people/alice/memberOf/Admin`

But you *cannot* use `prefetch` with collection resources, such as the following:

- `.../people`
- `.../groups`
- `.../people/alice/memberOf`

This section includes one example:

- Read a User With prefetch

Read a User With prefetch

This example shows how to retrieve the collection of "manager" attributes for the specified user in addition to the full set of User attributes that is returned by default.

cURL Command

```
curl -i --request GET
"http://localhost:16191/idaas_rest/rest/userprofile/people/JohnD/
?prefetch=manager"
```

Expected Output With prefetch

```
{
"uid":"JohnD",
"manager":
   {"elements":
    [{
     "report-uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD",
     "uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD\/manager\/SusanS",
     "manager-uri":
      {
       "uid":"SusanS",
       "manager":"\/idaas_rest\/rest\/userprofile\/people\/SusanS\/manager",
       "state":"CA",
       "lastname":"Smith",
       "firstname":"Susan",
       "loginid":"SusanS",
       "uniquename":"5B543C30790511E1AF41BD17BAB1A1C1",
       "uri":"\/idaas_rest\/rest\/userprofile\/people\/SusanS",
```

```
       "country":"USA",
       "guid":"5B543C30790511E1AF41BD17BAB1A1C1",
       "title":"Sr]]. Director, Development ",
       "name":"SusanS",
       "commonname":"Susan Smith"
      }
    }],
    "uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD\/manager
          ?pageSize=0&pagePos=-1"
   },
"state":"CA",
"lastname":"Doe",
"firstname":"John",
"loginid":"JohnD",
"uniquename":"2F23AC90790511E1AF41BD17BAB1A1C1",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD",
"country":"USA",
"guid":"2F23AC90790511E1AF41BD17BAB1A1C1",
"title":"Director, Development ",
"name":"JohnD",
"commonname":"John Doe"}
```

Expected Output Without prefetch

```
{
"uid":"JohnD",
"manager":"\/idaas_rest\/rest\/userprofile\/people\/JohnD\/manager",
"state":"CA",
"lastname":"Doe",
"firstname":"John",
"loginid":"JohnD",
"uniquename":"2F23AC90790511E1AF41BD17BAB1A1C1",
"uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD",
"country":"USA",
"guid":"2F23AC90790511E1AF41BD17BAB1A1C1",
"title":"Director, Development ",
"name":"JohnD",
"commonname":"John Doe"}
```

The "scope" Query Parameter Feature

Use the `scope` query parameter to retrieve a nested level of attributes in a relationship search.

For example:

`.../people/JohnD/manager?scope=toTop`

Use scope if a search is between two entities that have a direct hierarchical relationship, for example a *manager* relationship between one user and another user, or a *memberOf* relationship between a user and a group.

The scope query parameter can be used with the following User Profile Services standard entities: manager, reports, groupMemberOf, groupMembers, groupOwner, and groupOwnerOf.

Note: Configure the `toTop` scope attribute value by editing the *User Profile* Service Provider in the Oracle Access Management system administration console. In the **Relationship Configuration** section of the page, edit the values in the **Scope for Requesting Recursion** column. See "Editing or Creating a User Profile Service Provider" in the *Administrator's Guide for Oracle Access Management* for more information.

This section includes one example:

- Search a Relationship With scope

Search a Relationship With scope

This example shows how to do a Manager relationship Search with `scope` set `toTop`.

cURL Commands

Create User "JohnD"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{
  "uid":"JohnD",
  "title":"Director, Development ",
  "state":"CA",
  "lastname":"Doe",
  "commonname":"John Doe ",
  "firstname":"John",
  "password":"secret12345",
  "country":"USA"}'
```

Create User "SusanS"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{
  "uid":"SusanS",
  "title":"Sr. Director, Development ",
  "state":"CA",
  "lastname":"Smith",
```

```
  "commonname":"Susan Smith",
  "firstname":"Susan",
  "password":"12345secret",
  "country":"USA"}'
```

Create User "AlanC"

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/ -d
'{
  "uid":"AlanC",
  "title":"VP, Identity Management Development ",
  "state":"CA",
  "lastname":"Cooper",
  "commonname":"Alan Cooper",
  "firstname":"Alan",
  "password":"welcome321",
  "country":"USA"}'
```

Create a "manger" relationship between JohnD and SusanS

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/JohnD/manager -d
'{
  "report-uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD",
  "manager-uri":"\/idaas_rest\/rest\/userprofile\/people\/SusanS"}'
```

Create a "manager" relationship between SusanS and AlanC

```
curl -H "Content-Type: application/json" --request POST
http://localhost:14100/idaas_rest/rest/userprofile/people/SusanS/manager -d
'{
  "report-uri":"\/idaas_rest\/rest\/userprofile\/people\/SusanS",
  "manager-uri":"\/idaas_rest\/rest\/userprofile\/people\/AlanC"}'
```

Perform a "manager" relationship Search with scope = toTop

```
curl -i --request GET "http://localhost:14100/idaas_rest/rest/userprofile/people/
JohnD/manager/?scope=toTop&pagePos=0&pageSize=2"
```

Expected Output With scope = toTop

```
{"next":
"\/idaas_rest\/rest\/userprofile\/people\/JohnD\/manager
?pageSize=2&scope=toTop&pagePos=1",
"elements":
[{
  "report-uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD",
  "uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD\/manager\/SusanS",
  "manager-uri":
    {
     "uid":"SusanS",
     "manager":"\/idaas_rest\/rest\/userprofile\/people\/SusanS\/manager",
     "state":"CA",
     "lastname":"Smith",
     "firstname":"Susan",
     "loginid":"SusanS",
     "uniquename":"5B543C30790511E1AF41BD17BAB1A1C1",
     "uri":"\/idaas_rest\/rest\/userprofile\/people\/SusanS",
     "country":"USA",
     "guid":"5B543C30790511E1AF41BD17BAB1A1C1",
     "title":"Sr. Director, Development ",
     "name":"SusanS",
```

```
     "commonname":"Susan Smith"
    }
 },
 {
  "report-uri":"\/idaas_rest\/rest\/userprofile\/people\/SusanS",
  "uri":"\/idaas_rest\/rest\/userprofile\/people\/SusanS\/manager\/AlanC",
  "manager-uri":
    {
     "uid":"AlanC",
     "guid":"31486BE0790611E1AF41BD17BAB1A1C1",
     "title":"VP, Identity Management Development ",
     "name":"AlanC",
     "state":"CA",
     "lastname":"Cooper",
     "commonname":"Alan Cooper",
     "loginid":"AlanC",
     "firstname":"Alan",
     "uniquename":"31486BE0790611E1AF41BD17BAB1A1C1",
     "uri":"\/idaas_rest\/rest\/userprofile\/people\/AlanC",
     "country":"USA"
    }
}],
"uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD\/manager
?pageSize=2&scope=toTop&pagePos=0"}
```


Expected Output Without scope = toTop


```
{"elements":
 [{
  "report-uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD",
  "uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD\/manager\/SusanS",
  "manager-uri":
    {
     "uid":"SusanS",
     "manager":"\/idaas_rest\/rest\/userprofile\/people\/SusanS\/manager",
     "state":"CA",
     "lastname":"Smith",
     "firstname":"Susan",
     "loginid":"SusanS",
     "uniquename":"5B543C30790511E1AF41BD17BAB1A1C1",
     "uri":"\/idaas_rest\/rest\/userprofile\/people\/SusanS",
     "country":"USA",
     "guid":"5B543C30790511E1AF41BD17BAB1A1C1",
     "title":"Sr. Director, Development ",
     "name":"SusanS",
     "commonname":"Susan Smith"
    }
 }],
"uri":"\/idaas_rest\/rest\/userprofile\/people\/JohnD\/manager
?pageSize=2&pagePos=0"}
```

Practical Examples

The examples in this section present a progression of REST calls. First a device registration handle is acquired and then used in subsequent calls to the Mobile and Social server in order to authenticate a user, obtain access to a protected resource, and interact with User Profile Services. The basic sequence is (1) obtain a device registration handle, (2) obtain a user token, and (3) obtain an access token.

> **Note:** The REST examples presented in this section include line breaks and indented code blocks to help make them easy to read.

- Mobile SSO Agent Requests Client Registration Handle (Client Token)
- Mobile SSO Agent Requests Client Registration Handle on Behalf of Business App
- A User Token Request
- An Access Token Request
- Access Manager Master Token Authentication
- Device Registration Request with KBA Response

Mobile SSO Agent Requests Client Registration Handle (Client Token)

This example shows the client registration request call that the mobile SSO agent app on an iOS device sends to the Mobile and Social Server.

The Request

```
curl -H "Content-Type: application/json" --request POST
http://hostname.example.com:18001/idaas_rest/rest/mobilejwtauthentication/register
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-d
'{
  "X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
  "X-Idaas-Rest-Subject-Username":"jdoe",
  "X-Idaas-Rest-Subject-Password":"password123",
  "X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTREGHANDLE",
  "deviceProfile":
    {
     "oracle:idm:claims:client:sdkversion":"11.1.2.0.0",
     "hardwareIds":
       {
        "oracle:idm:claims:client:udid":"0e83ff56a12a9cf0c7",
        "oracle:idm:claims:client:phonenumber":"1-650-555-1234",
        "oracle:idm:claims:client:macaddress":"00-16-41-34-2C-A6",
        "oracle:idm:claims:client:imei":"010113006310121"
       },
     "oracle:idm:claims:client:jailbroken":false,
     "oracle:idm:claims:client:geolocation":"+40.689060,-74.044636",
     "oracle:idm:claims:client:networktype":"PHONE_CARRIER",
     "oracle:idm:claims:client:vpnenabled":false,
     "oracle:idm:claims:client:ostype":"iPhone OS",
     "oracle:idm:claims:client:phonecarriername":"AT&T",
     "oracle:idm:claims:client:locale":"EN-US",
     "oracle:idm:claims:client:osversion":"4.0"
    }
  "clientId":"OICSecurityApp"
}'
```

The Response

```
{"X-Idaas-Rest-Token-Value":"eyJ0b2tlblR...l9M=",
 "X-Idaas-Rest-Token-Type":"CLIENTREGHANDLE",
 "handles":
    {"oaam.device":
      {
       "expirationTSInSec":1334423076,
       "value":"20_7fe4bde3d448598c4cb8211d214b5eaded0620428c06061b1261644603717cd3"
      },
     "oaam.session":
      {
       "expirationTSInSec":1332955447,
       "value":"18_2743f64c111cb6691ea18689317958192d748b191a4955851e43f40910079e9a"
      }
    }
}
```

Mobile SSO Agent Requests Client Registration Handle on Behalf of Business App

The Request

```
curl -H "Content-Type: application/json" --request POST
http://hostname.example.com:18001/idaas_rest/rest/mobilejwtauthentication/register
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred="T0lDU2VjdXJ...Gw5TT0="'
-d
'{
  "X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
  "X-Idaas-Rest-Subject-Username":"jdoe",
  "X-Idaas-Rest-Subject-Password":"password123",
  "X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTREGHANDLE",
  "deviceProfile":
    {
     "oracle:idm:claims:client:sdkversion":"11.1.2.0.0",
     "hardwareIds":
       {
        "oracle:idm:claims:client:udid":"0e83ff56a12a9cf0c7",
        "oracle:idm:claims:client:phonenumber":"1-650-555-1234",
        "oracle:idm:claims:client:macaddress":"00-16-41-34-2C-A6",
        "oracle:idm:claims:client:imei":"010113006310121"
       },
     "oracle:idm:claims:client:jailbroken":false,
     "oracle:idm:claims:client:geolocation":"+40.689060,-74.044636",
     "oracle:idm:claims:client:networktype":"PHONE_CARRIER",
     "oracle:idm:claims:client:vpnenabled":false,
     "oracle:idm:claims:client:ostype":"iPhone OS",
     "oracle:idm:claims:client:phonecarriername":"AT&T",
     "oracle:idm:claims:client:locale":"EN-US",
     "oracle:idm:claims:client:osversion":"4.0"
    }
  "handles":
     {"oaam.session":"18_2743f64c111cb6691ea18689317958192d748b191a4955851e43f40910079e9a",
      "oaam.device":"20_7fe4bde3d448598c4cb8211d214b5eaded0620428c06061b1261644603717cd3"
     },
  "clientId":"WhitePageApp"
}'
```

The Response

```
{"X-Idaas-Rest-Token-Value":"eyJ0b2tlblR...Lyhko=",
 "X-Idaas-Rest-Token-Type":"CLIENTREGHANDLE",
 "handles":
    {"oaam.device":
      {
       "expirationTSInSec":1334423298,
       "value":"20_7fe4bde3d448598c4cb8211d214b5eaded0620428c06061b1261644603717cd3"
      },
     "oaam.session":
      {
       "expirationTSInSec":1332955669,
       "value":"18_2743f64c111cb6691ea18689317958192d748b191a4955851e43f40910079e9a"
      }
    }
}
```

A User Token Request

The Request

```
curl -H "Content-Type: application/json" --request POST
http://hostname.example.com:18001/idaas_rest/rest/mobilejwtauthentication/authenticate
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred="T0lDU2VjdXJpdHlBc...Fa00vOD0="'
-d
'{
  "X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
  "X-Idaas-Rest-Subject-Username":"jdoe",
  "X-Idaas-Rest-Subject-Password":"password123",
  "X-Idaas-Rest-New-Token-Type-To-Create":"USERTOKEN",
  "deviceProfile":
    {
     "oracle:idm:claims:client:sdkversion":"11.1.2.0.0",
     "hardwareIds":
       {
        "oracle:idm:claims:client:udid":"0e83ff56a12a9cf0c7",
        "oracle:idm:claims:client:phonenumber":"1-650-555-1234",
        "oracle:idm:claims:client:macaddress":"00-16-41-34-2C-A6",
        "oracle:idm:claims:client:imei":"010113006310121"
       },
     "oracle:idm:claims:client:jailbroken":false,
     "oracle:idm:claims:client:geolocation":"+40.689060,-74.044636",
     "oracle:idm:claims:client:networktype":"PHONE_CARRIER",
     "oracle:idm:claims:client:vpnenabled":false,
     "oracle:idm:claims:client:ostype":"iPhone OS",
     "oracle:idm:claims:client:phonecarriername":"AT&T",
     "oracle:idm:claims:client:locale":"EN-US",
     "oracle:idm:claims:client:osversion":"4.0"
    }
  "handles":
    {"oaam.session":"21_9e2e728b3180a7a3c9b80cef542c58339c2c7ed0e1a3ba66db4807ef1cf1523d",
     "oaam.device":"23_3a958d144b04f91c53b4236ed9f880357122df946f14ba21d957be5b49ef529b"
    }
}'
```

The Response

```
{"X-Idaas-Rest-Token-Value":"eyJhbGciOiJSUzUx...1OC6qw",
 "X-Idaas-Rest-Token-Type":"USERTOKEN",
 "handles":
    {"oaam.device":
      {
       "expirationTSInSec":1334424634,
       "value":"23_3a958d144b04f91c53b4236ed9f880357122df946f14ba21d957be5b49ef529b"
      },
     "oaam.session":
      {
       "expirationTSInSec":1332957005,
       "value":"21_9e2e728b3180a7a3c9b80cef542c58339c2c7ed0e1a3ba66db4807ef1cf1523d"
      }
    }
}
```

An Access Token Request

The Request

```
curl -H "Content-Type: application/json" --request POST
http://hostname.example.com:18001/idaas_rest/rest/mobilejwtauthentication/access
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred="T0lDU2VjdXJpdHlBc...TFPQzZxdw==''
-d
'{
  "X-Idaas-Rest-Subject-Type":"TOKEN",
  "X-Idaas-Rest-Subject-Value":"eyJhbGciOiJSUzUxM...411OC6qw",
  "X-Idaas-Rest-Application-Context":"<webgate context>",
  "X-Idaas-Rest-Application-Resource":"http:\/\/am-v40z-04.us.example.com:7777\/index.html",
  "X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN",
  "deviceProfile":
    {
     "oracle:idm:claims:client:sdkversion":"11.1.2.0.0",
     "hardwareIds":
       {
        "oracle:idm:claims:client:udid":"0e83ff56a12a9cf0c7",
        "oracle:idm:claims:client:phonenumber":"1-650-555-1234",
        "oracle:idm:claims:client:macaddress":"00-16-41-34-2C-A6",
        "oracle:idm:claims:client:imei":"010113006310121"
       },
     "oracle:idm:claims:client:jailbroken":false,
     "oracle:idm:claims:client:geolocation":"+40.689060,-74.044636",
     "oracle:idm:claims:client:networktype":"PHONE_CARRIER",
     "oracle:idm:claims:client:vpnenabled":false,
     "oracle:idm:claims:client:ostype":"iPhone OS",
     "oracle:idm:claims:client:phonecarriername":"AT&T",
     "oracle:idm:claims:client:locale":"EN-US",
     "oracle:idm:claims:client:osversion":"4.0"
    }
  "handles":
   {"oaam.session":"21_9e2e728b3180a7a3c9b80cef542c58339c2c7ed0e1a3ba66db4807ef1cf1523d",
    "oaam.device":"23_3a958d144b04f91c53b4236ed9f880357122df946f14ba21d957be5b49ef529b"
   }
}'
```

Access Manager Master Token Authentication

The Request

```
curl -H "Content-Type: application/json" --request POST
http://hostname.example.com:18001/idaas_rest/rest/mobilejwtauthentication/authenticate
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-IDAAS-REST-AUTHORIZATION: UIDPASSWORD cred="T0lDU2VjdXJpdHlBc...TFPQzZxdw==" '
-d
'{
  "X-Idaas-Rest-Subject-Type":"USERCREDENTIAL"
  "X-Idaas-Rest-Subject-Username":"jdoe",
  "X-Idaas-Rest-Subject-Password":"password123",
  "X-Idaas-Rest-New-Token-Type-To-Create":"USERTOKEN",
  "OAM-Token-Type-To-Create":"USERTOKEN::OAMMT",
  "deviceProfile":
    {
     "oracle:idm:claims:client:sdkversion":"11.1.2.0.0",
     "hardwareIds":
       {
        "oracle:idm:claims:client:udid":"0e83ff56a12a9cf0c7",
        "oracle:idm:claims:client:phonenumber":"1-650-555-1234",
        "oracle:idm:claims:client:macaddress":"00-16-41-34-2C-A6",
        "oracle:idm:claims:client:imei":"010113006310121"
       },
     "oracle:idm:claims:client:jailbroken":false,
     "oracle:idm:claims:client:geolocation":"+40.689060,-74.044636",
     "oracle:idm:claims:client:networktype":"PHONE_CARRIER",
     "oracle:idm:claims:client:vpnenabled":false,
     "oracle:idm:claims:client:ostype":"iPhone OS",
     "oracle:idm:claims:client:phonecarriername":"AT&T",
     "oracle:idm:claims:client:locale":"EN-US",
     "oracle:idm:claims:client:osversion":"4.0"
    }
  "handles":
   {"oaam.session":"21_9e2e728b3180a7a3c9b80cef542c58339c2c7ed0e1a3ba66db4807ef1cf1523d",
    "oaam.device":"23_3a958d144b04f91c53b4236ed9f880357122df946f14ba21d957be5b49ef529b"
   }
}'
```

Device Registration Request with KBA Response

Knowledge-based authentication (KBA) is an authentication scheme in which the user is asked to answer at least one question.

The Request to Register a Device

```
curl -H "Content-Type: application/json" --request POST
http://server1.domain.com:14100/
oic_rest/rest/mobileoamauthentication/register  -H
'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain' -d
'{"X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTREGHANDLE",
"X-Idaas-Rest-Subject-Password":"welcome1",
"deviceProfile":{"oracle:idm:claims:client:sdkversion":"11.1.2.0.0","hardwareIds":
{"oracle:idm:claims:client:udid":"0e83ff56a12a9cf0c7",
"oracle:idm:claims:client:phonenumber":"1-650-555-1234",
"oracle:idm:claims:client:macaddress":"00-16-41-34-2C-A6",
"oracle:idm:claims:client:imei":"010113006310121"},
"oracle:idm:claims:client:jailbroken":false,
"oracle:idm:claims:client:geolocation":"+40.689060,-74.044636",
"oracle:idm:claims:client:networktype":"PHONE_CARRIER",
"oracle:idm:claims:client:vpnenabled":false,
"oracle:idm:claims:client:ostype":"iPhone OS",
"oracle:idm:claims:client:phonecarriername":"AT&T",
"oracle:idm:claims:client:locale":"EN-US",
"oracle:idm:claims:client:osversion":"4.0"},
"X-Idaas-Rest-Subject-Username":"JohnS","clientId":"OICSSOApp",
"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL"}'
```

The Response Containing the KBA Question

```
{"handles":{"oaam.device":{"expirationTSInSec":1352076952,"value":"563_
23552f26e974030dc16018cc6b76237432c363d47a019cec8c73aa318caf2f97"},
"oaam.session":{"expirationTSInSec":1350609323,"value":"561_
419dc5ee6b325535dd026c882ac67cabc271dd7e0297ab73c74573a49dec233a"},
"oic.multiStepAuthnSessionHandle":{"expirationTSInSec":1350606623,"value":
"eyJvcmlnU2VjdXJpdHlFdmVudHMiOlsiUkVHX1NFQ1VSSVRZX0NMSUVOVF9BUFAiXSwib3JpZ1JlcU1hc
CI6eyJjbGllbnRJUEFkZHJlc3MiOiIxMC4xMzMuMTM5LjE0MyIsIlgtSWRhYXMtUmVzdC1TdWJqZWN0LVB
hc3N3b3JkIjoid2VsY29tZTEiLCJYLUlkYWFzLVJlc3QtTmV3LVRva2VuLVR5cGUtVG8tQ3JlYXRlIjoiQ
0xJRU5UUkVHSEFOREExFIiwiWC1JZGFhcy1SZXN0LVN1YmplY3QtVXNlcm5hbWUiOiJKb2huUyIsImNsaWV
udElkIjoiT0lDU1NPQXBwIiwiWC1JZGFhcy1SZXN0LVN1YmplY3QtVHlwZSI6IlVTRVJDUkVERU5USUFMI
n0sImNvbnRyYWN0TmFtZSI6Ik1vYmlsZVNlcnZpY2VEb21haW4iLCJzZXJ2aWNlSWRFUCI6IlwvbW9iaWx
lb2FtYXV0aGVudGljYXRpb24ifQ=="}},"message":
"The Challenge Action is triggered",
"multi-step-challenge-question":{"challengeType":"KBA","locale":"en",
"questionRefId":"112","questionStr":
"What was the year of your favorite sports moment?"},
"oicErrorCode":"IDAAS-61010","status":"REQUIRE_MULTI_STEP_AUTHN"}
```

The Request to Register the Device Containing the KBA Answer

```
curl -H "Content-Type: application/json" --request POST
http://server1.domain.com:14100/oic_rest/rest/mobileoamauthentication/register
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain' -d
'{"X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTREGHANDLE",
"X-Idaas-Rest-Subject-Password":"welcome1","deviceProfile":
{"oracle:idm:claims:client:sdkversion":"11.1.2.0.0","hardwareIds":
```

{"oracle:idm:claims:client:udid":"0e83ff56a12a9cf0c7",
"oracle:idm:claims:client:phonenumber":"1-650-555-1234",
"oracle:idm:claims:client:macaddress":"00-16-41-34-2C-A6",
"oracle:idm:claims:client:imei":"010113006310121"},
"oracle:idm:claims:client:jailbroken":false,
"oracle:idm:claims:client:geolocation":"+40.689060,-74.044636",
"oracle:idm:claims:client:networktype":"PHONE_CARRIER",
"oracle:idm:claims:client:vpnenabled":false,
"oracle:idm:claims:client:ostype":"iPhone OS",
"oracle:idm:claims:client:phonecarriername":"AT&T",
"oracle:idm:claims:client:locale":"EN-US",
"oracle:idm:claims:client:osversion":"4.0"},
"X-Idaas-Rest-Subject-Username":"JohnS","multi-step-challenge-answer":
{"challengeType":"KBA","locale":"EN-US","answerStr":
"moment","questionRefId":"112"},
"handles":{"oaam.session":"561_
419dc5ee6b325535dd026c882ac67cabc271dd7e0297ab73c74573a49dec233a",
"oaam.device":"563_
23552f26e974030dc16018cc6b76237432c363d47a019cec8c73aa318caf2f97",
"oic.multiStepAuthnSessionHandle":
"eyJvcmlnU2VjdXJpdHlFdmVudHMiOlsiUkVHX1NFQ1VSSVRZX0NMSUVOVF9BUFAiXSwib3JpZ1JlcU1hc
CI6eyJjbGllbnRJUEFkZHJlc3MiOiIxMC4xMzMuMTM5LjE0MyIsIlgtSWRhYXMtUmVzdC1TdWJqZWN0LVB
hc3N3b3JkIjoid2VsY29tZTEiLCJYLUlkYWFzLVJlc3QtTmV3LVRva2VuLVR5cGUtVG8tQ3JlYXRlIjoiQ
0xJRU5UUkVHSEFOREExFIiwiWC1JZGFhcy1SZXN0LVN1YmplY3QtVXNlcm5hbWUiOiJKb2huUyIsImNsaWV
udElkIjoiT0lDU1NPQXBwIiwiWC1JZGFhcy1SZXN0LVN1YmplY3QtVHlwZSI6IlVTRVJDUkVERU5USUFMI
n0sImNvbnRyYWN0TmFtZSI6Ik1vYmlsZVNlcnZpY2VEb21haW4iLCJzZXJ2aWNlSWRFUCI6IlwvbW9iaWx
lb2FtYXV0aGVudGljYXRpb24ifQ=="},"X-Idaas-Rest-Subject-Type":"USERCREDENTIAL"}'

The Response with a Client Registration Handle

{"X-Idaas-Rest-Token-Value":"eyJvcmFjbGU6aWRtOmNsYWltczpjbGllbnQ6c2RrdmVyc2lvbiI6I
jExLjEuMi4wLjAiLCJ0b2tlblR5cGUiOiJDTElFTlRSRUdIQU5ETEUiLCJvcmFjbGU6aWRtOmNsYWltczp
jbGllbnQ6bWFjYWRkcmVzcyI6IjAwLTE2LTQxLTM0LTJDLUE2IiwicmVnVXNlciI6IkpvaG5TIiwiaXNzI
joiTW9iaWxlT0FNQXV0aGVudGljYXRpb24iLCJvcmFjbGU6aWRtOmNsYWltczpjbGllbnQ6b3N0eXBlIjo
iaVBob25lIE9TIiwib3JhY2xlOmlkbTpjbGFpbXM6Y2xpZW50OmltZWkiOiIwMTAxMTMwMDYzMTAxMjEiL
CJyZWdUUyI6MTM1MDYwNTc4MCwianRpIjoiYTNlMWM1MjYtYjBjMS00ZDg0LThjYzAtZjYyMDNmYjM4NWV
lIiwib3JhY2xlOmlkbTpjbGFpbXM6Y2xpZW50Om9zdmVyc2lvbiI6IjQuMCIsImNsaWVudElkIjoiT0lDU
1NPQXBwIn0=.qA6Ez+gXNdLbk/hD5LRVDaBRK3t6b6IOOk7Z8iwW03s=",
"X-Idaas-Rest-Token-Type":"CLIENTREGHANDLE",
"handles":{"oaam.device":{"expirationTSInSec":1352077009,"value":"563_
23552f26e974030dc16018cc6b76237432c363d47a019cec8c73aa318caf2f97"},
"oaam.session":{"expirationTSInSec":1350609380,"value":"561_
419dc5ee6b325535dd026c882ac67cabc271dd7e0297ab73c74573a49dec233a"}}}

Part IV

Developing with Identity Federation

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

Part IV contains the following chapters:

13

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, which is discussed here. This chapter provides the following sections:

- Section 13.1, "Introduction to User Provisioning Plug-ins"
- Section 13.2, "Introduction to Plug-in Interfaces"
- Section 13.3, "Sample Code: Custom User Provisioning Plug-in"
- Section 13.4, "Developing a User Provisioning Plug-in"

> **See Also:** For more information about using the default user provisioning plug-in, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

13.1 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.

13.2 Introduction to Plug-in Interfaces

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*.

13.3 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 Section 3.4, "Sample Code: Custom Database User Authentication Plug-in".

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

- Example 13–1, "Sample UserProvisioning.java"
- Example 13–2, "Sample UserPlugin.xml"
- Example 13–3, "Sample MANIFEST.MF"

Example 13–1 Sample UserProvisioning.java

```
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 proisioning 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.oracle.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);
    }

    }
}
```

Example 13–2 Sample UserPlugin.xml

```
<Plugin type="User Provisioning">
<author>uid=User1</author>
<email>User1@mycompany</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>
```

Example 13–3 Sample 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
```

13.4 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:

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

13.4.1 Process Overview: Developing a Custom Plug-in

As Identity Federation leverages the Access Manager plug-in framework, the process is similar for both. For more information, see Section 3.1.2, "About Planning, the Authentication Model, and Plug-ins".

1. Extend `OIFUserProvisioningPlugin` class and implement the following methods. For more information, see Section 3.5.1, "About Writing a Custom Authentication Plug-in".
   - Implement `initialize` method
   - Implement `process` method
2. Develop plug-in code using appropriate Access Manager 11*g* interfaces and packages. For more information, see:
   - Section 3.3, "Introduction to Plug-in Interfaces"
   - Section 3.4, "Sample Code: Custom Database User Authentication Plug-in"
3. Prepare metadata for the custom plug-in. For more information, see Section 3.4.2, "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:
   - Section 3.4.3, "Sample Manifest File for the Plug-in"
   - Section 3.4.4, "Plug-in JAR File Structure"
5. Proceed to Section 13.4.2, "Files Required for Compiling a Plug-in".

For information about deploying and managing custom authentication plug-ins, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

13.4.2 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.

Part V

Developing with Security Token Service

This part discusses developing applications using the Oracle Access Management Security Token Service APIs.

Part V contains the following chapters:

- Chapter 14, "Developing a Custom Token Module"

14

Developing a Custom Token Module

When Oracle Security Token Service does not support the token that you want to validate or issue out-of-the-box, you can write your own validation and issuance module classes. This chapter contains information on Oracle Security Token Service custom token options. It includes the following sections:

- Section 14.1, "Introduction to Oracle Security Token Service Custom Token Module Classes"
- Section 14.2, "Writing a TokenValidatorModule Class"
- Section 14.3, "Writing a TokenIssuanceModule Class"

14.1 Introduction to Oracle Security Token Service Custom Token Module Classes

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

- The custom validation class, which is used to validate a custom token.
- The custom issuance class, which is used to issue a custom token.

The following overview outlines the tasks you must perform.

Task overview: Deploying custom token module classes

1. Writing a TokenValidatorModule Class to validate a custom token with Oracle Security Token Service, if needed.
2. Writing a TokenIssuanceModule Class to issue a custom token with Oracle Security Token Service, if needed.
3. Create a Custom Token module that will allow the user to create Validation Templates and Issuance Templates for their custom token. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.
4. Create Validation and Issuance Templates for the custom token, and use the custom templates in Endpoints and Partner Profiles as you would use the templates of standard tokens. For more information, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

14.2 Writing a TokenValidatorModule Class

This section provides the following topics:

- About Writing a TokenValidatorModule Class
- Writing a TokenValidatorModule Class

14.2.1 About Writing a TokenValidatorModule Class

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

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

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

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

Example 14–1 EmailTokenValidatorModuleImpl.java

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

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

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

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

public class EmailTokenValidatorModuleImpl implements TokenValidatorModule{
```

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

    private static final String TEST_SETTING_IN_TEMPLATE = "testsetting";

    public void init(Map options1) throws TokenProcessingException{

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

    }

    public TokenResult validate(TokenContext context) throws
TokenProcessingException{

        byte[] tokenBytes = (byte[])context.getOtherProperties().get("XML_TOKEN");
        Document inputDocument = (Document)context.getOtherProperties().get("XML_
DOM");


        Element tokenElement = (Element)context.getOtherProperties().get("TOKEN_
ELEMENT");
        String encodedBytes = (String) context.getOtherProperties().get("BST_
CONTENT");
        byte[] decodedBytes = null;
        BASE64Decoder decoder = new BASE64Decoder();
       try{
if(encodedBytes != null){
        decodedBytes = decoder.decodeBuffer(encodedBytes);
}
        }catch(java.io.IOException exp){
         exp.printStackTrace();
        }


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

            //add any other attributes - necessary only if you need for mapping or
issuance
            result.setTokenProperty("testattr2", "attr2");

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

            //add any other attributes - necessary only if you need for mapping or
issuance
            result.setTokenProperty("testattr2", "attr2");
```

```
            return result;

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

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


    }
}
```

The following overview outlines development highlights for this module class.

Development highlights: Writing a TokenValidatorModule class

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

      On Success, return:

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

      On Failure, return:

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

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

      ```
      result.setTokenProperty("STS_SUBJECT_ID", emailAddress);

      //add any other attributes - necessary only if you need for mapping or
      issuance
      result.setTokenProperty("testattr2", "attr2")
      ```

14.2.2 Writing a TokenValidatorModule Class

Perform the following tasks to write a custom TokenValidatorModule class.

Task overview: Writing a TokenValidatorModule class

1. Develop your own module class while referring to:

- Section 14.2.1, "About Writing a TokenValidatorModule Class"
- *Oracle Fusion Middleware Java API Reference for Oracle Access Management Security Token Service*

2. Proceed as needed:
   - Section 14.3, "Writing a TokenIssuanceModule Class"
   - For information abut managing a custom Security Token Service configuration, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*

14.3 Writing a TokenIssuanceModule Class

This section provides the following topics:

- About Writing a TokenIssuanceModule Class
- Writing a TokenIssuanceModule Class

14.3.1 About Writing a TokenIssuanceModule Class

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

Example 14–2 provides and example of `EmailTokenIssuanceModuleImpl` class. The overview that follows outlines development highlights for this module class.

Example 14–2 EmailTokenIssuanceModule.java

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

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

import javax.xml.namespace.QName;

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

public class EmailTokenIssuerModuleImpl implements TokenIssuerModule{

    Map config;

    private static final String TEST_SETTING_IN_TEMPLATE = "testsetting";


    public void init(Map options) throws TokenProcessingException
    {
       config = options;
```

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


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

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

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

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


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


              result.setTokenProperties(resultMap);
        return result;
```

```
    }

}
```

Development highlights: Writing a TokenIssuanceModule class

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

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

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

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

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

```
HashMap attributes = (HashMap)context.getOtherProperties().get("STS_TOKEN_
ATTRIBUTES");
System.out.println("attributes : " + attributes.toString());
String emailAddress = null;
Iterator attrIter = null;
if (attributes != null) {
attrIter = attributes.keySet().iterator();
}

if (attrIter != null) {
while (attrIter.hasNext()) {
String attributeName = (String)attrIter.next();
if ("mail".equals(attributeName)) {
Object valuesObj = attributes.get(attributeName);
if (valuesObj instanceof List){
Iterator iter = ((List)valuesObj).iterator();

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

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

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

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

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

Note: The attributes set as token properties are available in the context. The attributes can be used for token mapping or can be specified in the relying party profile attributes section for inclusion in the outgoing token in the usual way.

14.3.2 Writing a TokenIssuanceModule Class

Task overview: Writing an Issuance Module class

1. Write the issuance module class as you refer to Section 14.3.1, "About Writing a TokenIssuanceModule Class" and *Oracle Fusion Middleware Java API Reference for Oracle Access Management Security Token Service*.
2. For information about managing a custom Security Token Service configuration, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

Part VI

Appendices

This part contains reference appendices.

Part VI contains the following appendices:

- Appendix A, "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–1 on page A-2.
3. The user submits the login page, passing the parameters contained in Table A–2 on page 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.

A.1.1.1 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.

A.1.1.2 Password Is About to Expire

A warning page is displayed where the user can either change their password, or continue without changing before continuing.

A.1.1.3 Grace Login Is in Force

Same behavior as when password is about to expire.

A.1.1.4 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–1 on page A-2.

Table A–1 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.

Table A–1 (Cont.) Login Page Parameters Submitted to the Page by the Single Sign-On

Parameter	Description
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–2 to the authentication URL:

`http://sso_host:sso_port/sso/auth`

Table A–2 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

The URL for the change password page must accept the parameters listed in Table A–3.

> **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–3 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–3 (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–5 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–4 to the change password URL:

http://*sso_host*:*sso_port*/sso/ChangePwdServlet

Table A–4 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–5 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.

A.3.1 OSSO 10g Login Page Error Codes

When OAM Server is set to OSSO10*g*, the login page must process the error codes listed in Table A–5.

Table A–5 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."

Table A–5 (Cont.) Login Page Error Codes

Value of p_error_code	Corresponding message and description
internal_server_try_later_err	Description: Internal server error report with "try later" prompt. Message: "Internal Server Error. Please try the operation later."
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.

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.

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–2) in the login form. A number of sites contain a full list of ISO-639 two-letter language codes. Here is one of them:

```
http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt
```

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
```

> **Note:** In the event that the `locale` parameter is passed to the single sign-on server (Table A–1), the parameter value is sent to mod_osso. mod_osso prefixes this value to the HTTP Accept-Language header before passing the header to applications.

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 inOracle 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 Examples of 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.6.1 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 Adding an External Application

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. This page contains the following headings and fields:

Table A–6 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–7 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–8 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.
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.

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.