Oracle® Fusion Middleware

Developer's Guide for Oracle Access Management

11*g* Release 2 (11.1.2.3.0) for All Platforms

E54425-06

October 2016

ORACLE®

Oracle Fusion Middleware Developer's Guide for Oracle Access Management, 11*g* Release 2 (11.1.2.3.0) for All Platforms

E54425-06

Primary Author: Annapoorna A J

Contributing Author: Vinayak Lokhande, Binitha Monnappa, Kavitha Ramasamy.

Contributor: Toby Close, Vadim Lander, Jeremy Banford, Sreehari Narasimhaiah, Vamsi Motukuru, Paresh Raote, Prakash Manwani.

Contents

3 Developing Custom Authentication Plug-ins

4 Developing Custom Pages

5 Managing Policy Objects

6 Developing an Application to Manage Impersonation

Part III Developing with Mobile and Social

7 Developing Applications Using the Mobile and Social Client SDKs

8 Developing Mobile and Social Services Applications with the Java Client SDK

10 Developing Mobile and Social Services Applications with the Android Client SDK

11 Developing Applications Using the Social Identity Client SDK

12 Extending the Capabilities of the Mobile and Social Server

13 Customizing Oracle Mobile Authenticator

14 Using the Mobile and Social REST API

16 Customizing the OAuth Services

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

Related Documents

For more information, see the following documents in the Oracle Fusion Middleware 11*g* Release 2 (11.1.2.3.0) 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 for Identity and Access Management*

Conventions

The following text conventions are used in this document:

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

What's New in Oracle Access Management?

This section describes new features and guide changes in Oracle Access Management 11*g*.

The following sections describe the new features and changes.

- Guide Changes: 11g Release 2 Patch Set 3 (11.1.2.3.0)
- New Features in 11g Release 2 (11.1.2)
- Product and Component Name Changes

Guide Changes: 11*g* Release 2 Patch Set 3 (11.1.2.3.0)

The following developer content has been added or modified for this 11.1.2.3.0 release.

- Updated Chapter 18, "Using the REST API for Identity Federation" to include enhanced support in this guide (E54425-06) for the October 2016 Oracle Access Management documentation set refresh.
- Chapter 18, "Using the REST API for Identity Federation" has been added to this guide (E54425-05) for the October 2016 Oracle Access Management documentation set refresh.
- Mobile Services has been renamed to Mobile and Social Services.
- Chapter 19, "Developing a Message Processing Plug-in" has been added to this guide for the April 2016 Oracle Access Management documentation set refresh.
- Guide change sections in this chapter for releases before this current patch set release (11g Release 2 Patch Set 3) have been removed for the September 2015 Oracle Access Management documentation set refresh. This includes details regarding 11g Release 2 Patch Set 2 (11.1.2.2.0) and earlier.
- Some changes have been made to the organization of this guide for the September 2015 Oracle Access Management documentation set refresh.

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

Oracle Access Management 11*g* Release 2 (11.1.2.3.0) includes the following components. The new features discussed in this guide are described in the following sections.

- Oracle Access Management Access Manager
- Oracle Access Management Mobile and Social
- Oracle Access Management Identity Federation

- Oracle Access Management Security Token Service

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, Android devices, and Java. If you are developing an application on a platform or device that cannot use the iOS, Android, 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 17.

Oracle Access Management Security Token Service

There are no changes to Security Token Service APIs in this 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 general information about developing with the SDKs and APIs.

Part I contains the following chapter.

- Chapter 1, "Developing with Oracle Access Management Components"

1

Developing with Oracle Access Management Components

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

This chapter introduces the Oracle Access Management components.

1.1 About Access Manager

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

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

For 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".

1.2 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"

1.3 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. You can also develop a message processing plug-in. For more information about the Identity Federation APIs, see Part V, "Developing with Identity Federation".

1.4 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 VI, "Developing with Security Token Service".

1.5 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 2 contains the following chapters.

2

Developing Access Clients

Oracle Access Management Access Manager (Access Manager) provides a pure Java software developer kit (SDK) and application programming interfaces (APIs) for creating custom Access Clients.

This chapter discusses how to develop a custom Access Client and provides the following sections.

2.1 About Developing Access Clients

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 is intended for use by Java application developers in the development of tightly coupled, performant integrations. 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. In addition to this guide, for more information see *Oracle Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*.

> **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.

The following Access API 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.

  > **Note:** 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.

- **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 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*. The following sections have more information.

- When to Create a Custom Access Client
- Access Client Architecture

2.1.2.1 When to Create a Custom Access Client

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

- Protection for non-HTTP resources.
- Protection for a custom web server developed to implement a special feature (for example, a reverse proxy).

- Implementation of single sign-on (SSO) to protect a combination of HTTP and non-HTTP resources.

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

2.1.2.2 Access Client Architecture

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

1. Custom Access Client code.

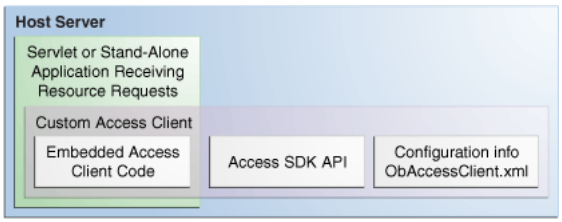
   Built into a servlet or standalone application. For the 11*g* release, you write Access Client code using the Java language platform.

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 protect 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.

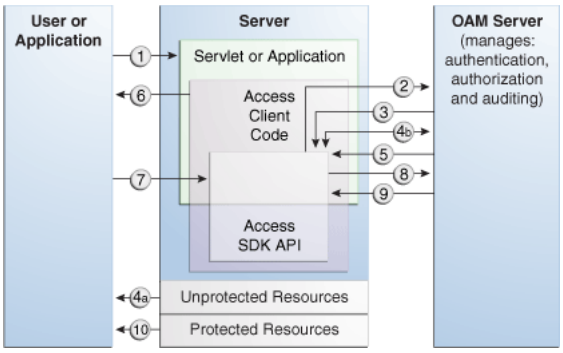
Figure 2–1 Architectural Detail of an Access Client



2.1.3 About Access Client Request Processing

Regardless of the variability introduced by the types of resources discussed in Section 2.1.2.2, "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. Details of the process overview are explained below the figure.

Figure 2–2 Process Overview: Handling a Resource Request



1. The application or servlet containing the Access Client code receives a user request for a resource.
2. The Access Client constructs a `ResourceRequest` structure, which the Access Client code uses when it asks the OAM Server whether the requested resource is protected.
3. The OAM Server responds.
4. Depending upon the situation, one of the following occurs:
   - If the resource is not protected, the Access Client grants or denies access to the resource depending on the value of the `DenyOnNotProtected` flag. Default value is true.

     For Access Manager 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 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.zip` file from Oracle Technology Network.
2. Extract the contents of the file `oam-java-asdk.zip` to a local directory.
3. Add `oamasdk-api.jar` to your CLASSPATH.

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

2.3 Developing Access Clients

The following topics are discussed in this section:

2.3.1 Structure of an Access Client

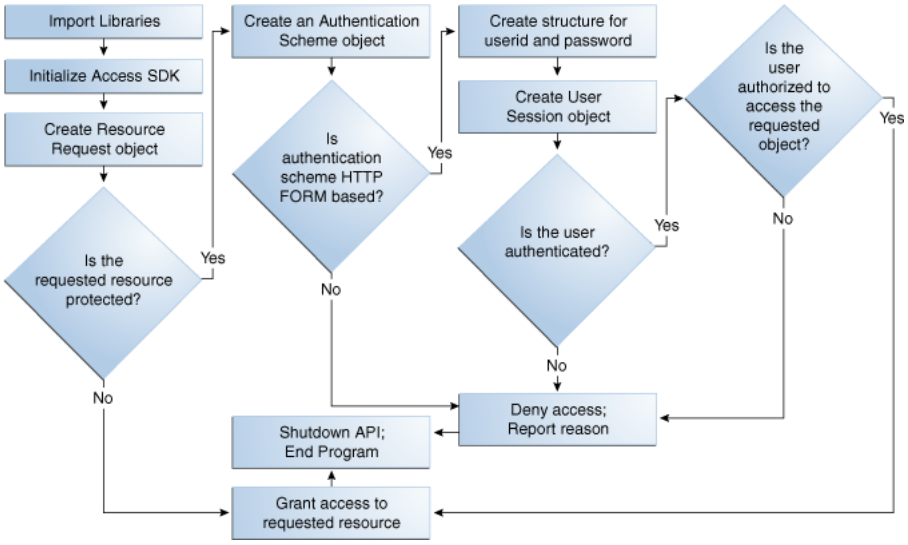
The structure of a typical Access Client application roughly mirrors the sequence of events required to set up an Access Client session.

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.

2.3.2 Typical Access Client Execution Flow

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

Figure 2–3 Process Flow for Form-based Applications



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

   Resource Not Protected: If the resource is not protected, the Access Client grants or denies access to the resource depending on the value of the `DenyOnNotProtected` flag. Default value is `true`. For Access Manager 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*.

2.3.3 Sample Code: Simple Access Client

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

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

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

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

See Section 2.3.4, "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();
  }
}
```

2.3.4 Annotated Sample Code: Simple Access Client

Import standard Java library class Hashtable to hold credentials.

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

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

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

This application is named `JAccessClient`.

```
public class JAccessClient {
```

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

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

```
public static final String 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 the `ObAccessClient.xml` configuration file. There are multiple ways to provide configuration location to initialize the Access SDK. For more information refer to *Oracle Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*.

The instance of `AccessClient` initializes the Access SDK API. When the AccessClient instance is created in OAM_10G mode, you must use a 10g agent profile. Similarly, when the AccessClient instance is created in OAM_11G mode, you must use an 11g agent profile. `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.

2.3.5 Sample Code: Java Login Servlet

This example follows the basic pattern of API calls that define an Access Client, as described in Section 2.3.3, "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*.

> **Note:** 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 2.3.6, "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>");
   }
}
```

2.3.6 Annotated Sample Code: Java Login Servlet

Import standard Java packages to support input, output, and basic functionality.

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

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

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

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

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

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

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

The `init` method is called once by the servlet engine to initialize the Access Client. In init method, Access SDK can be initialized by instantiating `AccessClient` by passing the location of the configuration file ObAccessClient.xml file. For more information for creating Access Client, refer to *Oracle Fusion Middleware Access SDK Java API Reference for Oracle Access Management Access Manager*. 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, the 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.

> **Note:** When the AccessClient instance is created in OAM_10G mode, you must use a 10g agent profile. Similarly, when the AccessClient instance is created in OAM_11G mode, you must use an 11g agent profile.

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

Redirect the user's browser to the target page.

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

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

```
 } else {
    out.println("<p>User " + sUserName + " not authorized
```

```
      for " + requestedPage + "\n");
 }
```

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

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

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

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

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

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

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

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

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

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

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

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

If an error occurs, report the backtrace.

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

Complete the output stream to the user's browser.

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

2.3.7 Sample Code: Additional Methods

Building on the basic pattern established in the sample application `JAccessClient.java`, discussed in Section 2.3.3, "Sample Code: Simple Access Client", the following sample invokes several additional OAM Server methods. For instance, it inspects the session object to determine which actions and named responses are

currently configured in the policy rules associated with the current authentication scheme.

For this demonstration to take place, you must configure some actions through the OAM Server prior to running the application. For details about authentication action and configuring user authentication, see *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 2.3.8, "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();
```

```
   }
  }
```

2.3.8 Annotated Sample Code: Additional Methods

Import standard Java libraries to provide basic utilities, enumeration, and token processing capabilities.

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

Import the Access SDK API libraries.

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

This class is named `access_test_java`.

```
public class access_test_java {
```

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

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

Set the current ResourceRequest to `res`.

```
ResourceRequest res;
```

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

```
Hashtable parameters = null;
```

Create a new hashtable named `cred`.

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

Initialize `AccessClient` reference to null.

```
AccessClient ac = null;
```

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

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

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

```
 return;
} else {
```

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

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

```
}
```

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

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

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

```
else
  configDir = null;
```

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

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

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

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

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

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

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

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

Assign the first token to the variable `name`.

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

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

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

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

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

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

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

Create `AccessClient` instance using `configDir`, in case if its null provide configuration file location using other options. For more information for creating Access Client, see

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

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

2.3.9 Sample Code: Certificate-Based Authentication in Java

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

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

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

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

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

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

// Create hashtable to hold credentials
Hashtable<String, String> creds = new Hashtable<String, String>();

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

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

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

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

The following import statements are associated with the snippet:

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

2.3.10 Sample Code: OAM_ID Cookie Creation Using ASDK

The OAM_ID cookie is used in a user session to implement complete SSO using ASDK. This cookie can be created using the Oracle Access Management Console or by editing the `oam-config.xml` file.

To create OAM_ID cookie using the Oracle Access Management Console, set the mandatory parameters, `AllowTokenScopeOperations` and `AllowMasterTokenRetrieval` for the WebGate agent where the SDK application is deployed. Use the sample code to retrieve master token in ASDK which is your java-based code. Retrieving this master token sets the OAM_ID cookie that is used for SSO between agents.

If the mandatory parameters are not displayed in the Oracle Access Management Console, you can edit the `oam-config.xml` file to create the OAM_ID cookie. Add the following settings to your WebGate 11g agent configuration:

```
<Setting Name="AllowTokenScopeOperations"
Type="xsd:boolean">true</Setting>
<Setting Name="AllowMasterTokenRetrieval"
Type="xsd:boolean">true</Setting>
```

Example 2–4 Sample ASDK Code for creating OAM_ID Cookie

```
//TODO: Use Set-Cookie header for httpOnly=true
//TODO: Typically this cookie's host is OAM host, set it as domain level cookie
retrieving the domain since we are setting this manually.
Cookie oamCookie2 = new Cookie("OAM_ID", user.getScopedSessionToken(null));
oamCookie2.setPath("/");
oamCookie2.setDomain("oracle.com");
response.addCookie(oamCookie2);
System.out.println("Cookie2: " + oamCookie2.getValue());
```

2.4 Generating Access SDK Logs

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. The log generated by the Access SDK provides information about operations performed. For example, operation status, any errors or exceptions that occur, and any general information that is helpful for troubleshooting can be logged. This section describes the messages and exceptions used by the Access SDK to indicate status or errors in the execution log.

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

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

- **OperationNotPermittedException**

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

  If the Access Client is not allowed to perform these operations, the 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.

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

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

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

You must provide the log file path against the `java.util.logging.FileHandler.pattern` property in the `logging.properties` file. If

you provide only the file name, the file will be created under the current directory. The following is an example `logging.properties` file.

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

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

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

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


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

The following is a sample of the log output:

```
Apr 19, 2011 5:20:39 AM AccessClient createClient
FINER: ENTRY
Apr 19, 2011 5:20:39 AM ObAAAServiceClient setHostPort
FINER: ENTRY
Apr 19, 2011 5:20:39 AM ObAAAServiceClient setHostPort
FINER: RETURN
Apr 19, 2011 5:20:39 AM ObAAAServiceClient setHostPort
FINER: ENTRY
Apr 19, 2011 5:20:39 AM ObAAAServiceClient setHostPort
FINER: RETURN
Apr 19, 2011 5:20:39 AM AccessClient createClient
FINER: RETURN
Apr 19, 2011 5:20:39 AM AccessClient initialize
FINER: read config from server, re-init if needed
Apr 19, 2011 5:20:39 AM AccessClient updateConfig
```

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

2.5 Building an Access Client Program

The following topics are discussed in this section:

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

2.5.1 Setting the Development Environment

The development environment has the following requirements:

- 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 2.2, "Installing Access SDK".

2.5.2 Compiling a New Access Client Program

After configuring the development environment as documented in Section 2.5.1, "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 2.2, "Installing Access SDK".

2.6 Configuring and Deploying Access Clients

After development, the Access Client must be deployed in a live Access Manager 11*g* environment in order to test and use it. 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 2.2, "Installing Access SDK".
2. Copy the Access Client to the computer hosting the application to be protected.
3. Configure the Access Client.
4. Verify you have the required Java environment available.

   If your Access Client is in a standalone environment, you can use Java Development Kit (JDK) or Java Runtime Environment (JRE). If your Access Client is a servlet application, you can use Java EE or the Java environment available with your Java EE container.
5. Verify that the Access SDK jar file is in the CLASSPATH. If in a non-JRF environment, verify that the necessary JPS jar files are in the CLASSPATH. For more information, see Section 2.2, "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*.

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

- Configuration Requirements
- Generating the Required Configuration Files
- SSL Certificate and Key File Requirements

2.6.1 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 the OAM Server when encrypting a token issued for the agent.

  The cwallet.sso file can be stored in the same location as other files or elsewhere. The path must be declared in jps-config.xml and is relative to the jps-config.xml location. cwallet.sso applies to 11g agents only.

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

  1. Prepare a merge-cred.xml, specifying the directory for the source cwallet.sso (agent registration artifacts) and the destination cwallet.sso (system artifacts). The file contents are like those defined in Example 2–5.

Example 2–5 merge-cred.xml Sample

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

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

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

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

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

```
    </jpsContext>

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

2. Run the following WLST command to merge the wallets.

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

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

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

- **jps-config.xml**

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

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

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

  A sample jps-config.xml file is included in the client install package zip file. This applies to 11g agents only.

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

- Java Security Grants

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

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

```
<grant>
  <grantee>
   <codesource>
    <url>... ...</url>
   </codesource>
  </grantee>
  <permissions>
   <permission>
     <class>oracle.security.jps.service.credstore.
```

```
            CredentialAccessPermission</class>
        <name>context=SYSTEM,mapName=OAMAgent,keyName=*</name>
        <actions>read</actions>
       </permission>
      <permissions>
    <grant>
```

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

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

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

- **JKS Keystores for SSL**

  This file is required only if the transport security mode is Simple or Cert. Both the 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 2.1.2.2, "Access Client Architecture" and Section 2.6.2, "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 2.6.2, "Generating the Required Configuration Files".

- **Log Configuration**

  Is required in order to generate a log file. For more information, see Section 2.4, "Generating Access SDK Logs".

2.6.2 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 Oracle Access Management Administration Console will also create a password.xml file.

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

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

2.6.3 SSL Certificate and Key File Requirements

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. The following sections have more information.

- Simple Transport Security Mode
- Cert Transport Security Mode

2.6.3.1 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/.

2.6.3.2 Cert Transport Security Mode

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

1. Import a CA certificate of the certifying authority using the certificate and key pair issued for Access Client and OAM Server. Follow the steps in Section 2.6.3.2.1, "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 2.6.3.2.2, "Setting Up The Keystore".

2.6.3.2.1 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.

2.6.3.2.2 Setting Up The Keystore The Access Client's SSL certificate and private key file must be added to the keystore. The SSL certificate and private key file must be generated 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.

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

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	

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

2.7.1 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.

2.7.2 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.

2.7.3 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.

2.8 Migrating or Converting 10g Applications

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.

Before migrating an application, ensure that your development environment and the 11*g* Access SDK is configured correctly. For more information, see Section 2.6, "Configuring and Deploying Access Clients". This section contains the following topics:

- Migrating Your 10g ASDK Component To Work with 11g
- Converting Your 10g Code

2.8.1 Migrating Your 10g ASDK Component To Work with 11g

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 2.7.2, "Compatibility of 10g JNI ASDK and 11g Access SDK".

Support for the `com.oblix.access` 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.

Note: 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.

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

- The configuration items listed in Section 2.6.1, "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.

To set your environment, follow the instructions in Section 2.5.1, "Setting the Development Environment". The 10*g* JNI ASDK is named jobaccess.jar. If jobaccess.jar is in your CLASSPATH, it must be removed.

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. The following sections have details depending on your component's mode.

- Migrating the 10g ASDK Component in Simple Mode
- Migrating the 10g ASDK Component in Cert Mode

2.8.1.1 Migrating the 10g ASDK Component in 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 2.6.3, "SSL Certificate and Key File Requirements".
5. Copy the JKS stores to ASDK_INSTALL_DIR/access/oblix/config/simple.

2.8.1.2 Migrating the 10g ASDK Component in 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 2.6.3, "SSL Certificate and Key File Requirements".

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

2.8.2 Converting Your 10g 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. Table 2–4 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. You can migrate application code that was implemented using 10*g* JNI ASDK to achieve the same functionality in 11*g* Access SDK. The following sections explains how to modify existing application code to use the new API in 11*g* Access SDK.

- Initializing and Uninitializing Access SDK
- Performing Access Operations

2.8.2.1 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
```

2.8.2.2 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
```

2.9 Best Practices

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

- Avoiding Problems with Custom Access Clients
- Identifying and Resolving Access Client Problems
- Resolving Environment Problems
- Tuning for High Load Environment

2.9.1 Avoiding Problems with Custom Access Clients

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

- Make sure that your Access Client attempts to connect to the correct OAM Server.
- Make sure the configuration information on your OAM Server matches the configuration information on your Access Client. You can check the Access Client configuration information on your OAM Server, using the Oracle Access Management Administration Console. For details, see "Registering 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 OBACCESS_INSTALL_DIR environment variable *must* be set on your Windows or UNIX-type host computer so that you can compile and link your Access Client. In general, you also want the variable to be set whenever your Access Client is running.
- Use the exception handling features (try, throw, and catch) of the language used to write your custom Access Client code to trap and report problems during development.
- Your Access Client represents just one thread in your entire, multi threaded application. To ensure safe operation within such an environment, Oracle recommends that developers observe the following practices for developing thread-safe code:
  - Use a thread safe function instead of its single thread counterpart. For instance, use localtime_r instead of localtime.
  - Specify the appropriate build environment and compiler flags to support multithreading. For instance, use -D_REENTRANT. Also, use -mt for UNIX-like platforms and /MD for Windows platforms.
  - Take care to use in thread-safe fashion shared local variables such as FILE pointers.

2.9.2 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*.

2.9.3 Resolving Environment Problems

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

- Java EE Containers
- Oracle WebLogic Server
- Other Application Servers

2.9.3.1 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.

2.9.3.2 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.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.

2.9.3.3 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.

2.9.4 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 to test the credentials provided by a user when he or she tries to access a resource. The plug-ins can be taken from a standard set provided with OAM Server installation, or the custom plug-ins created by your own Java developers.

This chapter provides the following sections regarding authentication plug-ins.

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

> **See Also:** 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:

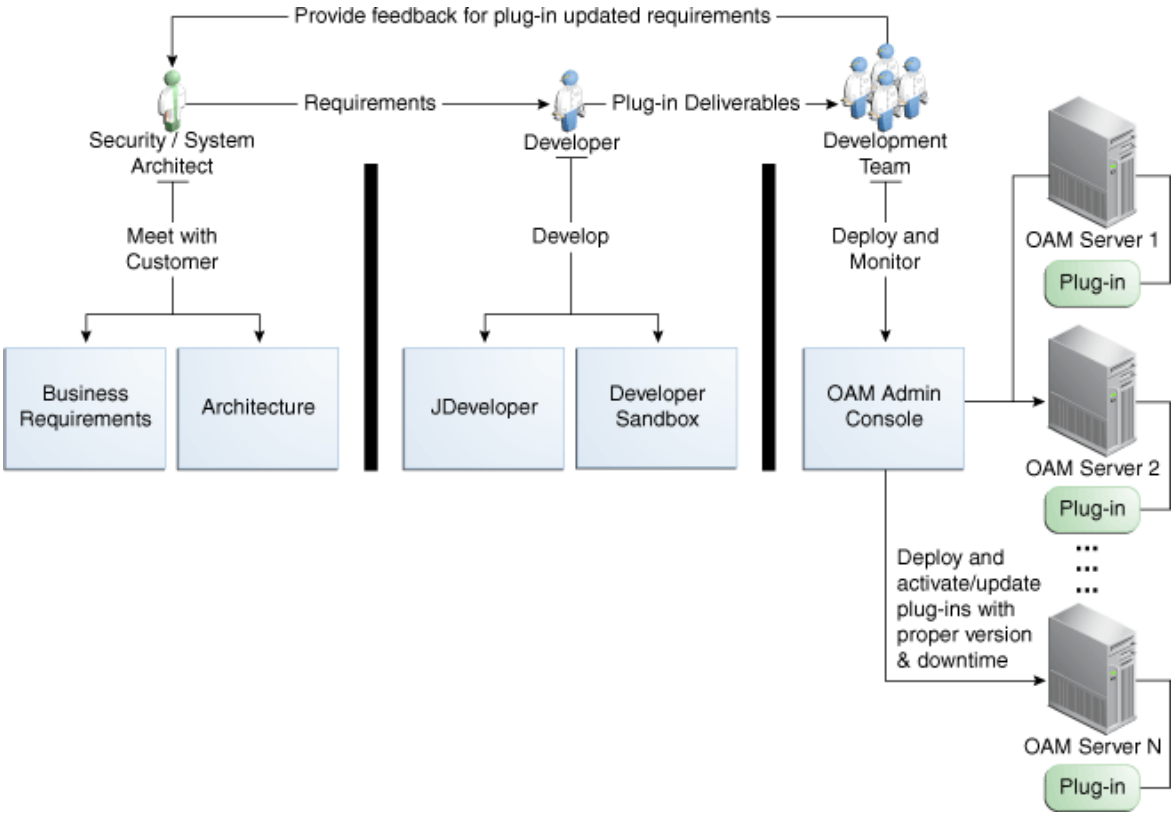
- Authentication plug-in interfaces and SDK tooling to build customized authentication modules (plug-ins) to bridge the out-of-the-box features with individual requirements. The new interfaces and SDK tooling:
  - Provide backward compatibility to support custom Oracle Access Manager 10*g* plug-ins.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

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 to add the following functionality. This is not a complete listing; other types of plug-ins are supported.

- User Identity Mapping

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

- User Authentication

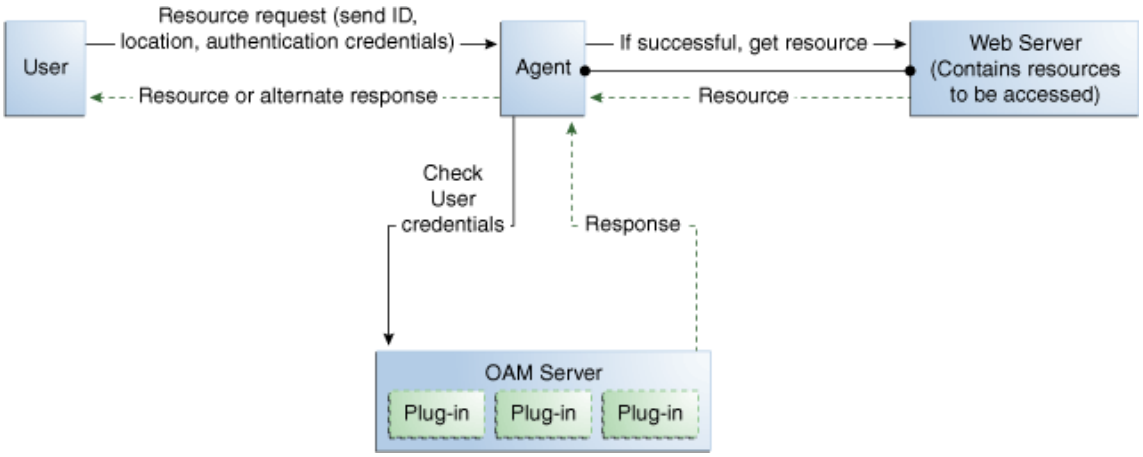
  Responses (not provided out-of-the-box) might be needed when authenticating the user. Custom plug-ins can fulfill this need.

- Custom Responses

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

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

Figure 3–2 Authentication Model and Plug-ins



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

When a certain request comes in, there are two possible ways to handle it. One is to have specific schemes run depending on the attributes of the request, using a decision engine to run one or multiple schemes to properly authenticate the user. This requires less code within each scheme and allows for more modularity. The second option is to have every scheme be hard-coded to handle the various attributes of requests for specific purposes, not using a decision engine to piece together which schemes need to be run (only one scheme is run). Each request approach has its own advantages and disadvantages as documented in Table 3–2.

Table 3–2 Request Approach Comparison

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

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

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

3.1.2.1 About the Decision Engine Approach Process

1. The request comes from the user with a certain IP address at midnight.

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

3.1.2.2 About the Hard-Coded Approach Process

1. The request comes from the user with a certain IP address at midnight.
2. The online bank account access scheme is chosen from among other authentication schemes (credit card access scheme, new account creation and verification, and so on).
3. The scheme first checks the IP address to determine if the user has previously made attempts to connect from the computer. It determines the user has.
4. The scheme checks the time. It requires a security question to be answered, which is answered successfully.
5. The scheme requires the user to enter his login credentials, and he authenticates successfully.

3.2 Introduction to Multi-Step Authentication Framework

This section provides the following topics:

- About the Multi-Step Framework
- Process Overview: Multi-Step Authentication
- About the PAUSE State
- 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

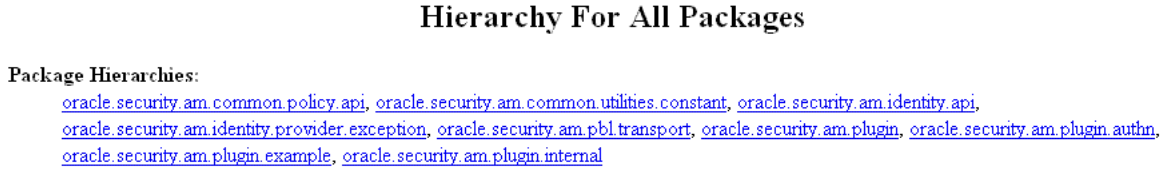


Figure 3–4 Plug-in Class Hierarchy

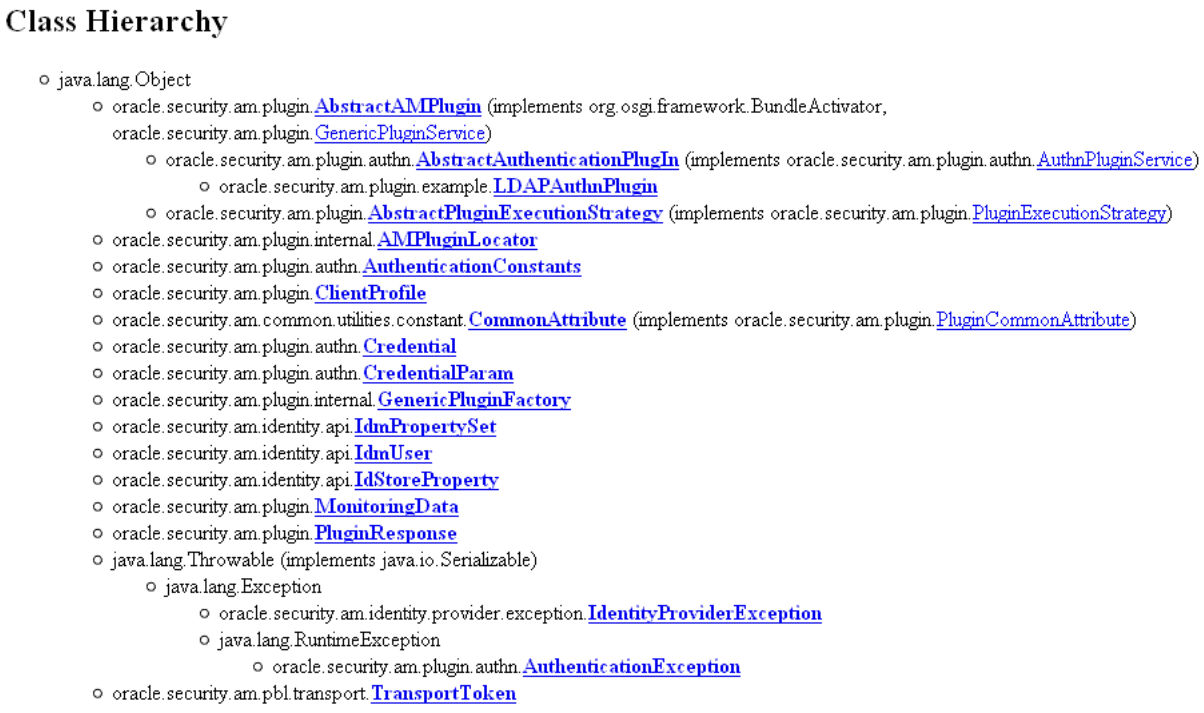


Figure 3–5 Plug-in Interface Hierarchy

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@example.com</email>
```

```
<creationDate>09:32:20, 2010-12-02</creationDate>
<description>Custom User Authentication Plugin Validation Against Domain
Name</description>
<configuration>
<AttributeValuePair>
<Attribute type="string" length="20">DataSource</Attribute>
<mandatory>true</mandatory>
<instanceOverride>false</instanceOverride>
<globalUIOverride>true</globalUIOverride>
<value>jdbc/CISCO</value>
</AttributeValuePair>
</configuration>
</Plugin>
```

3.4.3 Sample Manifest File for the Plug-in

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

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

Example 3–2 Sample Manifest File

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

```
<globalUIOverride>true</globalUIOverride>
<value>FORWARD</value>
</AttributeValuePair>
.
</configuration>
.
</Plugin>
```

3.4.4 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.

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

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

```
,jndi-1.2.1.jar,ldap.jar,providerutil.jar,oaam_soap_client.jar,oaam_
core.jar,oaam_uio.jar
```

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

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

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

3.5.2 Writing a Custom Authentication Plug-in

This section provides steps to write a custom authentication plug-in. The following overview describes the actions a developer must take after the system architect identifies the business requirements for this plug-in and considers the authentication flow when a user requests a resource. For more information, see Section 3.1.2, "About

Planning, the Authentication Model, and Plug-ins".

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

Oracle Access Manager provides default login, logout, error and password pages and an extensible framework for creating a custom page tailored to the look and feel of your company's brand.

This chapter explains how to develop custom pages and how to deploy them in your environment. It provides the following sections.

4.1 Introducing the Custom Pages Framework

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

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

Login, logout, error and password pages are packaged as part of the custom pages framework. An out-of-the-box Web application archive (WAR) file is provided that can be used as a starting point to develop customized pages. This WAR is named oamcustompages.war and located in the *MW_HOME*`/oam/server/tools/custompages/` directory. After creating a custom HTML page, add it to the WAR. For more information, see Section 4.3.1, "Creating a Form-Based Login Page."

Note: A custom page can be used in combination with existing Access Manager authentication modules or a custom authentication plug-in. For information about developing a plug-in, see Chapter 3, "Developing Custom Authentication Plug-ins."

Any user facing custom pages must return the OAM_REQ token and the authentication endpoint. The following sections have more information.

- Returning the OAM_REQ Token
- Returning the End Point

4.1.1 Returning the OAM_REQ Token

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

Note: For more information, see "Introduction to SSO Cookies" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

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

```
String reqToken = request.getParameter(GenericConstants.AM_REQUEST_TOKEN_
IDENTIFIER);

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

4.1.2 Returning the End Point

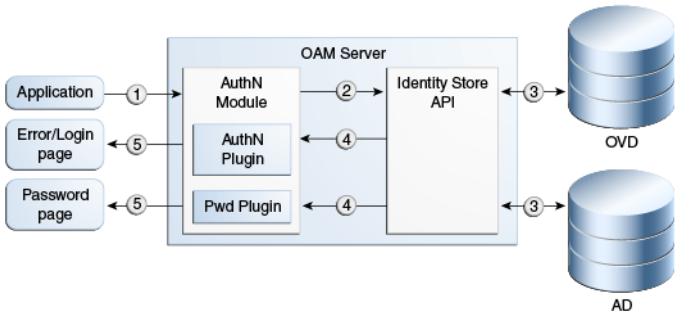
The end point, `/oam/server/auth_cred_submit`, must be returned to the OAM Server. For example:

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

4.2 Authenticating with Custom Pages

The authentication process involves determining what credentials a user must supply when requesting access to a resource, gathering the credentials over HTTP, and returning an HTTP response that reflects the results of credential validation. Figure 4–1 shows the end-to-end request flow for authenticating a user accessing an Access Manager protected resource that results in an error page.

Figure 4–1 Authentication Request Flow



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

1. When a user requests a protected resource, the authentication flow is triggered and a login page is displayed. The user enters the required credentials which are then submitted to the authentication engine.
2. The Identity Store API establishes a connection to the backend store to complete authentication. AD and OVD are shown as example identity stores.
3. The Identity Store layer returns the results to the plug-in and the authentication engine layer. The authentication layer maps the error codes from the backend to the corresponding Access Manager error codes. For information about the standard error codes, see Table 4–2.

   The results include any authentication error codes and native error codes acquired from the identity stores. Native error codes are returned to the login page as unmapped secondary error codes (`p_sec_error_msg`). For information about secondary error codes, see Section 4.4.4, "Secondary Error Message Propagation".

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

The following sections contain more information.

- Using mod_osso Agent
- Using Unsolicited Post
- Using Unsolicited Login With DCC WebGates
- Setting Custom OSSO Cookies After Authentication

4.2.1 Using mod_osso Agent

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

4.2.1.1 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*.

4.2.1.2 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>
```

4.2.1.3 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.

4.2.2 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.example.com:7778`*`/sample-web/headers.jsp`.
3. Once the credentials are validated, OAM Server redirects to the success URL after setting `OAM_ID` cookie as part of HTTP redirect (HTTP response code 302).

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

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

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

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

4.2.3 Using Unsolicited Login With DCC WebGates

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

1. Configure the DCC WebGate.

   Refer to the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

2. Enable Direct Authentication.

   Ensure that ServiceStatus under DirectAuthenticationServiceDescriptor in the `oam-config.xml` file is set to true. The DirectAuthenticationServiceDescriptor parameter is under OAMServicesDescriptor.

3. Add `TunneledUrls=/oam/server/authentication` to the User-defined parameters of the DCC WebGate profile.

   `/oam/server/authentication` is a Direct Authentication URL. For information on configuring Tunneled URLs, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

4. Restart the DCC WebGate.

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

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

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

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

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

   Note: See Section 4.2.2, "Using Unsolicited Post" to configure the Authentication Scheme for the resources protected using the Direct Authentication URL.

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

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

4.2.4 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=".example.com")

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

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

deleteOSSOResponseCookieConfig(cookieName = "ORASSO_UCM_COOKIE2")
```

4.3 Understanding Custom Login Pages

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

- **Context Type** = CustomWar
- **Challenge URL** = Relative path for the URL of the login page inside the WAR file
- **Context Value** = Custom WAR's root path. If a customized error page is included as part of the Custom WAR file, you must specify the absolute URL in the authentication policy-failure redirect URI. For example: `http://host:port/SampleLoginWar/pages/MFAError.jsp`.

Note: 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*.

More information is in the following sections.

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

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

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

A custom login form page has the following requirements:

- The page must be built to support the desired authentication module.
- The page must support retrieval of the OAM_REQ token. See Section 4.1.1, "Returning the OAM_REQ Token".
- The page must retrieve the end point. See Section 4.1.2, "Returning the End Point".

4.3.2 Page Redirection Process

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

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

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

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

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

4.4 Understanding Custom Error Pages

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

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

For information on how error code query parameters `p_error_code` and `p_sec_error_msg` will map to the custom error codes in your environment, see Section 4.4.5, "Sample Code: Retrieving Error Codes".

> **See Also:** Section 3.5.3, "Error Codes in an Authentication Plug-In" and Section 4.5.5, "Sample Code: Retrieving Password Policy Error Codes"

4.4.1 Enabling Error Page Customization

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

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

Information on how to use `updateCustomPages` and other WLST commands can be found in the *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference for Identity and Access Management* and in Specifying the Custom Error and Logout Page Deployment Paths in this chapter.

4.4.2 Standard Error Codes

Access Manager provides standard error codes that indicate the reasons for failure. Common reasons include an invalid username and password combination, a locked or disabled user account, or an internal processing error. The reason for the authentication error is received from the backed identity store and mapped to a specific error code maintained in the Access Manager Server. This error code is then propagated to the login or error page. (The custom error page also displays any internal server errors as well.)

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

Table 4–1 Types of Error Information

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

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

Table 4–2 Standard Error Codes and Message

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

4.4.3 Security Level Configuration

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

- SECURE: Most secure level. Provides a generic primary error message that gives little information about the internal reason for the error.
- EXTERNAL: The recommended level and is the default.

- INTERNAL: The least secure level. Enables propagation of error code to login or error page.

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

Table 4–3 Error Condition Mapping by Security Level

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

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

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

4.4.4 Secondary Error Message Propagation

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

> **Note:** The primary error codes are propagated to the error or login page in all modes. The secondary error message is propagated only when the security level is configured to be INTERNAL. For more information, see Section 4.4.3, "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 string of code 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

4.4.5 Sample Code: Retrieving Error Codes

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

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

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

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

Example 4–1 Resource Bundle Code

```
package mytest.error;
     Import java.util.ListResourceBundle;
  public class ExampleErrorMsg extends ListResourceBundle {
/* (non-Javadoc)
 * @see java.util.ListResourceBundle#getContents()
 */
public Object[][] getContents()
    {
        return m_contents;
    }
    /** The Constant m_contents. */
    private static final Object[][] m_contents =
     {
        {"OAM-1", " An incorrect Username or Password was
 specified "},
       {"OAM-2", " An incorrect Username or Password was
```

```
 specified "},
        {"OAM-3", "Unexpected Error occurred while processing
 credentials. Please retry your action again!"},
        {"", ……};
             }
    }
```

Example 4–2 Error Code Page

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

4.4.6 Error Data Sources Summary

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

Table 4–4 Authentication Plug-In Error Data Sources

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

4.5 Understanding Custom Password Pages

Access Manager processes user-entered passwords and password changes with form-based Java Server Pages (JSP). These pages can be customized in several languages or to display a company logo. When writing a custom password page for

authenticating users, a common method to follow is to redirect the user to a password page hosted outside Access Manager; custom pages are written to process a redirect from the user and to render the HTML. They can be kept as a JSP, or written using ASP.net, Perl, PHP, and other similar technologies. The pages enable pre-processing of the user's submission (POST) before their credentials are sent to Access Manager, if desired. The following sections contain more details.

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

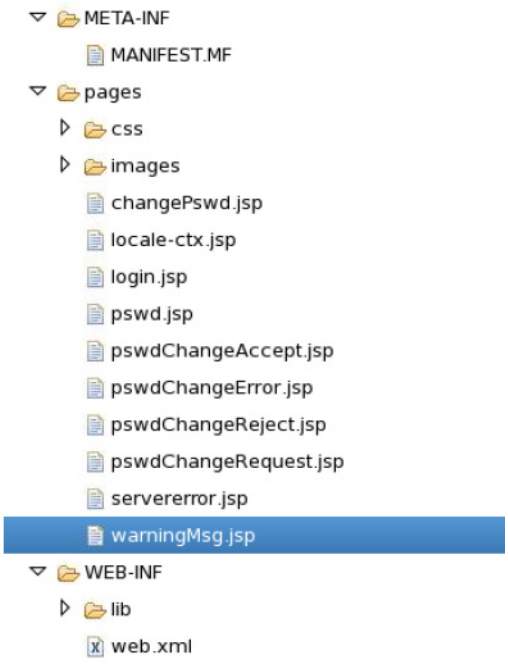
4.5.1 Customizing the Password Page WAR

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

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

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

Figure 4–2 Unarchived WAR



4.5.2 Using the Request Cache

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

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

4.5.3 Specifying the Password Service URL

The starting point for password pages is `pswd.jsp`. The location of this page is configured under Password Policy using the Oracle Access Management Administration Console. For details, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

1. Log in to the Access Manager Administration Console as administrator.
2. Click the Password Policy link under Access Manager.
3. Define or modify the value of the Password Service URL attribute.
4. Click Apply.

4.5.4 Sample Code: Retrieving Warning Messages

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

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

4.5.5 Sample Code: Retrieving Password Policy Error Codes

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

```
String simpleMessage = "";
  String result = request.getParameter("rejectedRuleDesc");
  if(result.indexOf('~') != -1) {
   String[] results = result.split("~");
   for(String eachResult : results) {
      if(eachResult.indexOf(":") != -1) {
        String messageKey = eachResult.substring(0, eachResult.indexOf(":"));
        String resourceBundleKey =
UrlSubstitutionMessages.ERRORCODEMAP.get(messageKey);
        String placeHolderValue = eachResult.substring(eachResult.indexOf(":") +
1, eachResult.length());
```

```
        String displayValue = Localizer.localize(resourceBundleKey,
placeHolderValue, myLocale);
        simpleMessage += displayValue + "<br>";
      }
      else {
        String resourceBundleKey =
UrlSubstitutionMessages.ERRORCODEMAP.get(eachResult);
        String displayValue = Localizer.localize(resourceBundleKey, null,
myLocale);
        simpleMessage += displayValue + "<br>";

      }
   }
  }
```

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

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

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

Table 4–5 Password Validation Error Codes

Message Key in URL	Message Key for Resource Bundle	Message Text
PSWD-1	passwordPolicy.message.minLength	Password must be at least {0} characters long
PSWD-2	passwordPolicy.message.maxLength	Password must not be longer than {0} characters
PSWD-3	passwordPolicy.message.minAlpha	Password must contain at least {0} alphabetic characters
PSWD-4	passwordPolicy.message.minNumber	Password must contain at least {0} numeric characters
PSWD-5	passwordPolicy.message.minAlphaNumeric	Password must contain at least {0} alphanumeric characters
PSWD-6	passwordPolicy.message.minSpecialChars	Password must contain at least {0} special characters
PSWD-7	passwordPolicy.message.maxSpecialChars	Password must not contain more than {0} special characters
PSWD-8	passwordPolicy.message.maxRepeated	Any particular character in the password must not be repeated more than {0} times
PSWD-9	passwordPolicy.message.minUnique	Password must contain at least {0} unique characters
PSWD-10	passwordPolicy.message.minUpperCase	Password must contain at least {0} uppercase letters

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

Message Key in URL	Message Key for Resource Bundle	Message Text
PSWD-11	passwordPolicy.message.minLowerCase	Password must contain at least {0} lowercase letters
PSWD-12	passwordPolicy.message.maxAge	Password will expire {0} days after the last password change
PSWD-13	passwordPolicy.message.warnAfter	Password change reminder will be sent {0} days after the last password change
PSWD-14	passwordPolicy.message.reqdChars	Password must contain the following characters: {0}
PSWD-15	passwordPolicy.message.invalidChars	Password must not contain the following characters: {0}
PSWD-16	passwordPolicy.message.validChars	Password can contain the following characters: {0}
PSWD-17	passwordPolicy.message.invalidStrings	Password must not contain the following strings: {0}
PSWD-18	passwordPolicy.message.startsWithChar	Password must start with an alphabetic character
PSWD-19	passwordPolicy.message.disAllowUserId	Password must not match or contain user ID
PSWD-20	passwordPolicy.message.disAllowFirstName	Password must not match or contain first name
PSWD-21	passwordPolicy.message.disAllowLastName	Password must not match or contain last name
PSWD-22	passwordPolicy.message.dictMessage	Password must not be a dictionary word
PSWD-23	passwordPolicy.message.enforceHistory	Password must not be one of {0} previous passwords
PSWD-24	passwordPolicy.message.minAge	Password cannot be changed for {0} days after the last password change
PSWD-25	passwordPolicy.message.minUnicode	Password must contain at least {0} Unicode characters
PSWD-26	passwordPolicy.message.maxUnicode	Password must not contain more than {0} Unicode characters

4.5.6 Sample Code: Obtaining Password Policy Rules

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

```
String simpleMessage = "";
String result = request.getParameter("ruleDesc");
if(result.indexOf('~') != -1) {
String[] results = result.split("~");
for(String eachResult : results) {
if(eachResult.indexOf(":") != -1) {
String messageKey = eachResult.substring(0, eachResult.indexOf(":"));
String resourceBundleKey = UrlSubstitutionMessages.ERRORCODEMAP.get(messageKey);
String placeHolderValue = eachResult.substring(eachResult.indexOf(":") + 1,
```

```
eachResult.length());
String displayValue = Localizer.localize(resourceBundleKey, placeHolderValue,
myLocale);
simpleMessage += displayValue + "<br>";
}
else {
String resourceBundleKey = UrlSubstitutionMessages.ERRORCODEMAP.get(eachResult);
String displayValue = Localizer.localize(resourceBundleKey, null, myLocale);
simpleMessage += displayValue + "<br>";

}
}
}
```

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

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

4.6 Using the Credential Collectors with Custom Pages

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

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

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

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

This section contains details regarding the DCC.

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

4.6.1 About the Detached Credential Collector with Custom Pages

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

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

4.6.2 Creating a Form-Based Login Page Using DCC

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

   For more information, see Section 4.3.1, "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.

4.6.3 About Custom Login and Error Pages for DCC Tunneling

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

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

A procedure (similar to Creating a Form-Based Login Page Using DCC) should be followed to create and use these custom pages. For more details on credential collection and DCC tunneling, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

4.7 Specifying the Custom Error and Logout Page Deployment Paths

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

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

The logout page path is:

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

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

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

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

5

Managing Policy Objects

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

The following sections contain details regarding the Policy Administration API.

5.1 About the Policy Administration API

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

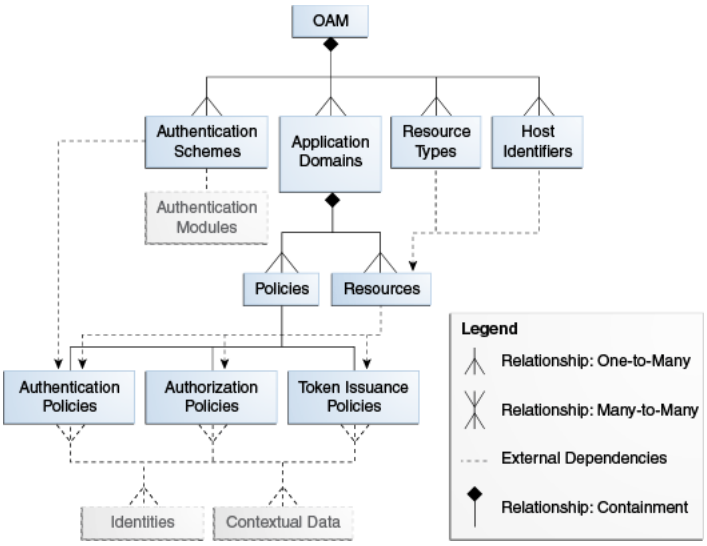


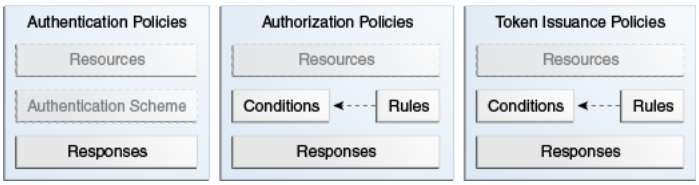
Table 5–1 provides details about the policy objects that can be managed using the RESTful interfaces. Each policy object is represented as an HTTP resource that is accessible through an HTTP uniform resource locator (URL).

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

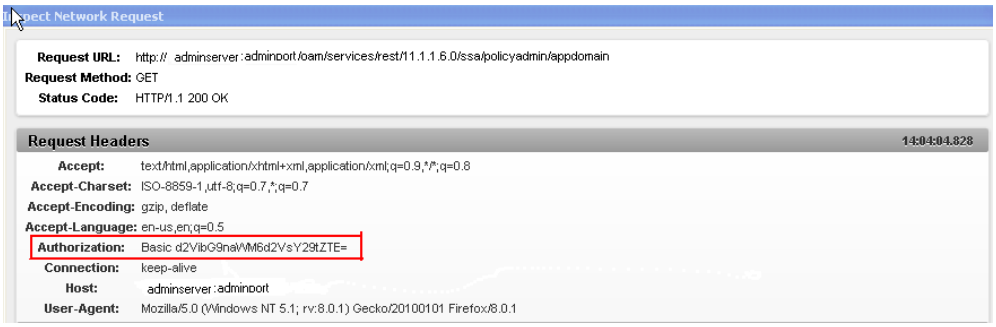


You can access the OAM Server RESTful interfaces through client applications such as:

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

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

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

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

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

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

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

Table 5–2 Resource URLs

Policy Object	URL	Artifact Mandatory Parameter
Application Domain	`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/appdomain`	IDENTIFIER
Host Identifier	`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/hostidentifier`	IDENTIFIER
Resource Type	`/oam/services/rest/11.1.2.0.0/ssa/policyadmin/resourcetype`	IDENTIFIER

Table 5–2 (Cont.) Resource URLs

Policy Object	URL	Artifact Mandatory Parameter
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
Client does not accept produced content type	406	Not acceptable
Request parameters semantics incorrect	422	Unprocessable entity <additional information on nature of error>

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

Error Condition	HTTP Return Code	Content
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 Client Tooling

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

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

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

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

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

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

5.5 cURL Command Examples

The following examples are provided as reference.

- Retrieve Application Domains cURL Command
- Create a New Application Domain cURL Command
- Retrieve All Authentication Schemes cURL Command
- Create an Authentication Scheme cURL Command
- Retrieve a Specific Authentication Scheme cURL Command
- Retrieve All Resources in an Application Domain cURL Command
- Create a Resource in an Application Domain cURL Command
- Retrieve All Policies in an Application Domain cURL Command

Retrieve Application Domains cURL Command

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

The following is sample output from this cURL command.

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

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

Create a New Application Domain cURL Command

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

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

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

The following is sample output from this cURL command.

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

Retrieve All Authentication Schemes cURL Command

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

The following is sample output from this cURL command.

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

Create an Authentication Scheme cURL Command

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

The following is a sample input file.

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

The following is sample output from this cURL command.

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

Retrieve a Specific Authentication Scheme cURL Command

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

The following is sample output from this cURL command.

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

Retrieve All Resources in an Application Domain cURL Command

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

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"
```

Retrieve All Policies in an Application Domain cURL Command

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

6

Developing an Application to Manage Impersonation

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

- About Impersonation
- Configuring Impersonation Support
- 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 also supported when using impersonation feature.

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

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

In the following example, assume:

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

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

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

For example, LDAP search for `orclguid`:

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

where:

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

For example, LDAP search for `orclImpersonationGrantee`:

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

where:

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

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

```
A14BEB42E822D605E040E50AB29327E7|20100324163000Z|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.example.com:7777/appl2.html");
   m.put(SsoService.FAILURE_URL, "https://login01.example.com:7777/fail.html");
   m.put(SsoService.IMP_USER_ID, "mcooper");

    sso.beginImpersonation(req, res, m);


    [....]

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

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

Example 6–3 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.example.com/oracleas/schema/11/jps-config-11_1.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.example.com/oracleas/schema/11/jps-config-11_1
1.xsd">
    <property value="off" name="oracle.security.jps.jaas.mode"/>
    <propertySets>
        <propertySet name="saml.trusted.issuers.1">
            <property value="www.example.com" name="name"/>
        </propertySet>

[...]

        <propertySet name="props.auth.uri.0">
            <property value="/oamsso/logout.html" name="logout.url"/>
            <property
value="https://login01.example.com:7777/oam/server/impersonate/end
              name="imp.end.url"/>
            <property
value="https://login01.example.com:7777/oam/server/impersonate/start
              name="imp.begin.url"/>
            <property value="/${app.context}/adfAuthentication" name="login.url
             .BASIC"/>
            <property value="/${app.context}/adfAuthentication" name="login.
             url.ANONYMOUS"/>
            <property value="/${app.context}/adfAuthentication" name="login.
             url.FORM"/>
        </propertySet>
        <propertySet name="props.auth.level.0">
            <property value="0" name="type-level:ANONYMOUS"/>
            <property value="1" name="type-level:BASIC"/>
            <property value="2" name="type-level:FORM"/>
        </propertySet>
    </propertySets>

[...]
```

6.1.4 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.example.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.example.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. The following sections contain details.

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

6.2.1 Configuring Impersonation Using oam-config.xml

The impersonation feature for the OAM Server is enabled by configuring the `oam-config.xml` file. Example 6–6 shows the relevant section of the file and the

parameters that can be set. EnableImpersonation must be set to 'true' to enable impersonation. The default setting is 'false'.

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

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

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

6.2.2 Configuring Impersonation Using idmConfigTool

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

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

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

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

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

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

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

   ```
   83295E092B2F9FD4E040E50AEBB91998|20100604224517Z|20110604224517Z;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. To set the authentication scheme to LDAPScheme:

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

For more information about the LDAPScheme, see "Managing Authentication Schemes" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

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 advice is provided as an example of the steps to take. Adjust the steps as needed for your environment.

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

   For more information, see "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.

It contains the following chapters:

7

Developing Applications Using the Mobile and Social Client SDKs

Mobile and Social Client SDKs are provided with Oracle Access Management. They are used for building Identity security features into your applications and enabling you to use your existing Identity infrastructure for authentication, authorization, and directory-access services.

This chapter briefly introduces the Mobile and Social client SDKs and 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 and Social Services Applications

Three Client SDKs—Android, 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 and Social 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 Android, iOS, or Java SDKs, you can write code to directly send Mobile and Social REST calls to the Mobile and Social server.

See Chapter 14, "Using the Mobile and Social REST API", which documents the Mobile and Social REST API.

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

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

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

Feature	Android	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 10, "Developing Mobile and Social Services Applications with the Android Client SDK" See Chapter 9, "Developing Mobile and Social 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 and Social Services Applications with the Java Client SDK"
Interact with a Directory server and implement User Profile Services	✔	✔	✔	See Chapter 10, "Developing Mobile and Social Services Applications with the Android Client SDK" See Chapter 9, "Developing Mobile and Social Services Applications with the iOS Client SDK" See Chapter 8, "Developing Mobile and Social Services Applications with the Java Client SDK"

Table 7–1 (Cont.) Features and Capabilities of the Java and iOS Mobile and Social Services Client SDKs

Feature	Android	iOS	Java	Notes
Create a mobile single sign-on (SSO) app	✔	✔		See Chapter 10, "Developing Mobile and Social Services Applications with the Android Client SDK" See Chapter 9, "Developing Mobile and Social Services Applications with the iOS Client SDK"

7.3 Introduction to 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.

- In general, LDAP attribute names are not case sensitive. However, when communicating with the Oracle Identity Governance Framework (IGF) APIs, LDAP attribute names *are* case sensitive.
- 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:
>
> `< >, #, |, &, ?, ( ), { }`

7.4 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 11, "Developing Applications Using the Social Identity 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 12.2, "Create a new Identity Service Provider for Internet Identity Services".

8

Developing Mobile and Social 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 Before You Begin

The Mobile and Social Java Client SDK for Mobile and Social Services is included in the Oracle Access Management distribution package. It 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 Java 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 and Social Services" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management* for information about configuring a service provider. The code samples are organized into the following sections.

- Initialize Objects and Define Endpoints
- 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 Import the Java Client SDK Classes

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;
```

8.2.2 Initialize Objects and Define Endpoints

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);
```

8.2.3 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);
```

8.2.4 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);
```

8.2.5 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);
```

8.2.6 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()));
}
```

8.2.7 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()));
}
```

8.2.8 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);
```

8.2.9 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 "Introduction to 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:

- Importing Java Classes and Declaring People
- 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 Importing Java Classes and Declaring People

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:

- Importing Java Classes and Declaring Groups
- 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
- Removing a Group (or a User) from a Group Using deleteOwnerOf

8.3.2.1 Importing Java Classes and Declaring Groups

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:

- Importing Java Classes and Declaring Groups
- Creating User Data with a Helper Utility
- Establishing Manager and Reports Relationships with a Helper Utility
- Creating Users at Different Hierarchies with a Data Preparation Utility
- 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 Importing Java Classes and Declaring Groups

The following code samples import the Java classes and 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 User Data with a Helper Utility

```
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());

}
```

8.3.3.3 Establishing Manager and Reports Relationships with a Helper Utility

```
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());
}
```

8.3.3.4 Creating Users at Different Hierarchies with a Data Preparation Utility

```
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);
```

8.3.3.5 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
}
```

8.3.3.6 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.
}
```

8.3.3.7 Retrieve All Reports Using Scope=All Feature

The following code samples verify 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;
  }
}
```

8.3.3.8 Retrieve the Manager Chain Using Scope=toTop Feature

The following code samples use 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();
```

8.3.3.9 Retrieve Report Details Using Pre-Fetch Feature

The following code samples retrieve 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);
```

8.3.3.10 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);
```

8.3.3.11 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 and Social Services Applications with the iOS Client SDK

This chapter describes how to develop Mobile and Social 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.

The iOS SDK supports devices running iOS 6.0 and above. 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 the following modules:

- **Authentication Module** - Processes authentication requests on behalf of users, devices, and applications.
- **Secure Storage Module** - Provides APIs to store and retrieve sensitive data using the iOS Keychain feature.
- **User Role Module** - Provides User Profile Services that allow users and applications to get User and Group details from a configured Identity store.
- **Cryptography Module** - Provides intuitive Objective C APIs for common cryptography tasks.
- **REST Web Service Handler Module** - Provides access to REST Web services protected by Access Manager.

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. Download `libIDMMobileSDK.a` to your development environment and add it to Xcode.
2. Download the `PublicHeaders` and `PublicResources` folders located in `oamms_sdk_for_ios.zip`.

   The `PublicHeaders` directory contains the IDM Mobile SDK header files.

   The `PublicResources` directory contains the IDM Mobile SDK resources.
3. Add the contents of `PublicHeaders` and `PublicResources` to your project.
4. Add the following frameworks to your project:
   - `SystemConfiguration.framework`
   - `Security.framework`
   - `CoreLocation.framework`

Important: Before linking your project, in Build Settings add as a single line both the `-ObjC` and `-all_load` flags to "Other linker flags." Without these flags your application will crash with a "selector not recognized" runtime exception.

Because `libIDMMobileSDK.a` 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/`

You can now start coding using the IDM Mobile iOS Client SDK.

Important: The iOS SDK supports devices running iOS 6.0 and above.

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. It contains sample code for the following tasks.

- Initializing the Required Objects
- Setting Up the Service
- Completing the Authentication Process
- Logging a User Out

Refer to "Configuring Mobile and Social Services" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management* for information about configuring a service provider.

9.2.1 Initializing the Required Objects

Create an `NSMutableDictionary` object and populate parameters:

```
NSMutableDictionary *sdkProps = [[NSMutableDictionary alloc] init];

[sdkProps setObject:OM_PROP_AUTHSERVER_OAMMS forKey:OM_PROP_AUTHSERVER_TYPE];
[sdkProps setObject:@"http://oammsurl" forKey:OM_PROP_OAMMS_URL];
[sdkProps setObject:@"myApp" forKey:OM_PROP_APPNAME];
[sdkProps setObject:@"MobileServiceDomain" forKey:OM_PROP_OAMMS_SERVICE_DOMAIN];
```

Next, create the `OMMobileSecurityService` object and pass the `Dictionary` object as properties.

```
OMMobileSecurityService *mss = [[OMMobileSecurityService alloc]
                                        initWithProperties:sdkProps
                                                  delegate:self];
```

- The `OM_PROP_OAMMS_URL` property key 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 `OM_PROP_APPNAME` property key 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 `OM_PROP_OAMMS_SERVICE_DOMAIN` property key is the name of the Service Domain in Mobile and Social server that contains the Application Profile for this application.

9.2.2 Setting Up the Service

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 on completion the iOS client SDK calls the following delegate method:

```
mobileSecurityService:(OMMobileSecurityService *)mobileSecurityService
didReceiveApplicationProfile:(NSDictionary *)applicationProfile
error:(NSError *)error;
```

Note: The name of the delegate to be implemented by the application developer is OMMobileServiceDelegate.

This method returns an `NSDictionary` object that contains the application profile details.

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

---

9.2.3 Completing the Authentication Process

Upon receiving the application profile, start the authentication process. Supply your own customized login and KBA (knowledge-based authentication) views in the authentication request as in the following sample. If a customized view is not required, pass a value of zero for `authnReq` in `startAuthenticationProcess`. If a customized login views are not passed, the SDK will throw a default login and KBA view.

```
OMAuthenticationRequest *authnReq = [OMAuthenticationRequest alloc] init];
```

```
authnReq.kbaView = myKBAView;
authnReq.authView = myLoginView;
[[mss startAuthenticationProcess:authnReq presenterViewController:myViewController];
```

This 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. This method takes a Boolean: `true` means check with the server to see if the authentication context is still valid; `false` means try to get the authentication context locally without checking whether it is still valid. For details about `OMAuthenticationContext`, see the API documentation.

```
- (void)mobileSecurityService: (OMMobileSecurityService *)mobileSecurityService
      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;
}
```

At this point your application can use the token obtained from the Mobile and Social server to make additional web service calls.

Note: The `OMMobileServiceDelegate` now contains selectors that include `mobileSecurityService`. The old selectors without `mobileSecurityService` are also available. When both are implemented, the new selector alone is called.

9.2.4 Logging a User Out

Call the following method to log out the user:

```
[mss logout:false];
```

The Mobile and Social logout operation involves invoking a REST Web service to delete tokens maintained in the server. When the Boolean parameter `clearRegistrationHandle` is true, the service clears device registration handles stored in the keychain. If false, only the session tokens are deleted.

Because a web service call is involved, the following delegate is called in `OMMobileServiceDelegate` when the operation completes:

```
- (void)mobileSecurityService:(OMMobileSecurityService *)mobileSecurityService
             didFinishLogout:(NSError *)error;
```

sample implementation of the delegate is as follows:

```
- (void) mobileSecurityService:(OMMobileSecurityService *)mobileSecurityService
                  didFinishLogout:(NSError *)error
    {
        NSString *msg = nil;
        if (error)
            msg = [NSString stringWithFormat:@"%@-%05ld: %@", [error domain],
                   (long)[error code], [error localizedDescription]];
        else
            msg = [NSString stringWithFormat:@"You are logged out successfully"];

        UIAlertView *alertView = [[UIAlertView alloc]
                                  initWithTitle:@"Logout"
                                  message:msg
                                  delegate:self
                                  cancelButtonTitle:@"OK"
                                  otherButtonTitles:nil];
        [alertView show];
        [alertView release];
    }
```

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 Setting Up URL-Based Configuration

URL-based configuration can easily configure an app utilizing the SDK. It works like this: Create an URL containing SDK configuration properties. When a user opens this URL using a mobile browser, the mobile app receives the URL in the `application:openURL:sourceApplication:annotation` delegate method. The mobile app needs to pass this URL to the IDM Mobile SDK. The SDK extracts and persists the configuration parameters and, once persisted, the app can reuse the configuration for subsequent authentication. This provides a user friendly way to configure the app.

The configuration URL should adhere to the following pattern:

```
<urlscheme>://[host]?<parameter1>::=<value1>&<parameter2>::=<value2>&...&<parameterN>::=<valueN>
```

For example:

```
wp://settings?AuthServerType::=OAMMSAuthentication&OAMMSURL::=http://host123.us.example.com:14100
&OAMMSServiceDomain::=MobileServiceDomain&ApplicationName::=WhitePages
```

The url scheme identifies the app to be opened. This can be added to an iOS project in Xcode by setting "URL Schemes" in the project Info section. Note that the URL scheme needs to match the setting in the OAM console. An URL scheme defined as `wp` in the console needs to be defined as `wp:// in Xcode`. URL schemes must be unique.

In the app, the application delegate is the right place to handle the URL and complete the configuration steps. A code sample is given below.

```
- (BOOL) application:(UIApplication *)application
            openURL:(NSURL *)url
   sourceApplication:(NSString *)sourceApplication
          annotation:(id)annotation
{
/* This snippet assumes "settings" is passed as host in the config URL. Replace it
with the actual host passed in the config URL*/
   if([host url] isEqualToString:@"settings"])
   {
       NSSet *sdkPropsFilter = [NSSet setWithObjects:OM_PROP_AUTHSERVER_TYPE,
                                    OM_PROP_OAMMS_URL,
                                    OM_PROP_OAMMS_SERVICE_DOMAIN,
                                    OM_PROP_APPNAME,nil];
       [OMMobileSecurityService parseConfigurationURL:url
                                  persistInUserDefaults:true
                                                withKey:nil
                                           andFilters:sdkPropsFilter];
   }
   return YES;
}
```

The mobile app can also apply some filters on the configuration received from the URL. Refer to the `andFilters` parameter in the code above. Using this parameter the app can specify a list of SDK parameters that should be allowed. The remaining parameters in the URL extracted from the configuration URL are ignored by the IDM Mobile SDK. After you have received and persisted the configuration, the `OMMobileSecurityService` object can be initialized. The `withKey` parameter can be used to identify a set of configuration parameters. When this is set to zero, the SDK treats it as the default configuration.

```
OMMobileSecurityService *mss = [[OMMobileSecurityService alloc] initWithDelegate:self];
```

This will create an `OMMobileSecurityService` object with the configuration parameters stored in `NSUserDefaults`. Use the following selector to pick the configuration associated with `key`:

```
- (id)initWithPropertiesAvailableInNSUserDefaultsWithKey:(NSString *)key
```

This helps maintain multiple configuration parameter sets in a mobile app.

9.4 About Initialization Properties for M&S Authentication

You can set the following properties when you initialize the SDK.

Table 9–1 iOS Client SDK Initialization Properties

Property Name	Property Value(s)	Type
OM_PROP_AUTHSERVER_TYPE (AuthServerType)	■ OM_PROP_AUTHSERVER_OAMMS (OAMMSAuthentication)	NSString **Note:** This is a required property.
OM_PROP_OAMMS_URL (OAMMSURL)	An NSURL Object or a valid NSString object that can be used to create NSURL. If you do not use this property, specify the following properties: ■ OM_PROP_OAMMS_HOST ■ OM_PROP_OAMMS_PORT ■ OM_PROP_OAMMS_PORT_IS_SSL	NSString or NSURL
OM_PROP_OAMMS_SERVICE_DOMAIN (OAMMSServiceDomain)	Required property. Any valid Service Domain name passed as an NSString object.	NSString
OM_PROP_APPNAME (ApplicationName)	Required property. A valid application name passed as an NSString object.	NSString
OM_PROP_OFFLINE_AUTH_ALLOWED (OfflineAuthAllowed)	Either true or false. This is an optional property, however, if it is not specified, offline authentication is not allowed. The Mobile and Social server can also specify this property in the application profile. The server setting overrides the application setting.	NSString
OM_PROP_MAX_LOGIN_ATTEMPTS (MaxLoginAttempts)	The maximum number of login attempts allowed by the user. This is an optional property. If this property is not specified, only one login attempt is allowed. The Mobile and Social server can also specify this property in the application profile. The server setting overrides the application setting.	NSNumber

Table 9–1 (Cont.) iOS Client SDK Initialization Properties

Property Name	Property Value(s)	Type
OM_PROP_KEYCHAIN_DATA_PROTECTION (KeychainDataProtection)	Use this property to specify the KeyChain Item protection level. Must be one of the following values: ■ OM_KEYCHAIN_DATA_ACCESSIBLE_WHEN_UNLOCKED (KeychainDataAccessibleWhenUnlocked) ■ OM_KEYCHAIN_DATA_ACCESSIBLE_AFTER_FIRST_UNLOCK (KeychainDataAccessibleAfterFirstUnlock) ■ OM_KEYCHAIN_DATA_ACCESSIBLE_ALWAYS (KeychainDataAccessibleAlways) ■ OM_KEYCHAIN_DATA_ACCESSIBLE_WHEN_UNLOCKED_THIS_DEVICE_ONLY (KeychainDataAccessibleWhenUnlockedThisDeviceOnly) ■ OM_KEYCHAIN_DATA_ACCESSIBLE_AFTER_FIRST_UNLOCK_THIS_DEVICE_ONLY (KeychainDataAccessibleAfterFirstUnlockThisDeviceOnly) ■ OM_KEYCHAIN_DATA_ACCESSIBLE_ALWAYS_THIS_DEVICE_ONLY (KeychainDataAccessibleAlwaysThisDeviceOnly) This is an optional property. If it is not specified, the property defaults to the highest level: OM_KEYCHAIN_DATA_ACCESSIBLE_WHEN_UNLOCKED_THIS_DEVICE_ONLY	NSString
OM_PROP_LOCATION_UPDATE_ENABLED (LocationUpdateEnabled)	Either true or false. This is an optional property and the default value is false. Location update consumes battery on the device and should be enabled only when required.	NSString
OM_PROP_LOCATION_UPDATE_DISTANCE_FILTER (LocationUpdateDistanceFilter)	Location update (when enabled) is sent only when the device moves beyond the specified distance in meters. Use this property to control the frequency of updates generated. The Mobile and Social server can also specify this property in the Mobile Custom Attributes section with the key LocationUpdateEnabled and a numeric value. The server setting overrides the application setting. When location update is enabled and the distance filter is not set or is invalid, the default distance filter is taken as 1000m.	NSNumber
OM_PROP_AUTO_LOGIN_ALLOWED (AutoLoginAllowed)	Either true or false. This is an optional property, however, if it is not specified, the Auto Login feature is disabled.	NSString
OM_PROP_REMEMBER_CREDENTIALS_ALLOWED (RememberCredentialsAllowed)	Either true or false. This is an optional property, however, if it is not specified, the Remember Credentials feature is disabled.	NSString
OM_PROP_REMEMBER_USERNAME_ALLOWED (RememberUsernameAllowed)	Either true or false. This is an optional property, however, if it is not specified, the Remember User Name feature is disabled.	NSString

Table 9–1 (Cont.) iOS Client SDK Initialization Properties

Property Name	Property Value(s)	Type
OM_AUTO_LOGIN_DEFAULT (AutoLoginDefault)	Either true or false. This is an optional property, however, if it is not specified, the default value for the Auto Login user preference is false.	NSString
OM_REMEMBER_CREDENTIALS_DEFAULT (RememberCredentialDefault)	Either true or false. This is an optional property, however, if it is not specified, the default value for the Remember Credentials user preference is false.	NSString
OM_REMEMBER_USERNAME_DEFAULT (RememberUsernameDefault)	Either true or false. This is an optional property, however, if it is not specified, the default value for the Remember User Name user preference is false.	NSString

The following OM_PROP_CRYPTO_SCHEME cryptography property is an optional property. If the application requires offline authentication and this property is not specified, the default crypto scheme is OM_PROP_CRYPTO_SSHA512.

This property can also be set using the Mobile and Social server console. Choose **Custom Settings** and click **Mobile Custom Attributes** and use the Attribute Name and Attribute Value listed in the last two columns in the table. The server setting overrides the application setting.

For information about the cryptography module included in the Mobile and Social iOS Client SDK, see Section 9.14, "Using the Cryptography Module."

Table 9–2 Cryptography Scheme Property Attributes for the iOS Client SDK

Property	Value	Attribute Name	Attribute Value
OM_PROP_CRYPTO_SCHEME	OM_PROP_CRYPTO_PLAINTEXT	CryptoScheme	PlainText
	OM_PROP_CRYPTO_AES		AES
	OM_PROP_CRYPTO_SHA1		SHA-1
	OM_PROP_CRYPTO_SHA224		SHA-224
	OM_PROP_CRYPTO_SHA256		SHA-256
	OM_PROP_CRYPTO_SHA384		SHA-384
	OM_PROP_CRYPTO_SHA512		SHA-512
	OM_PROP_CRYPTO_SSHA1		SaltedSHA-1
	OM_PROP_CRYPTO_SSHA224		SaltedSHA-224
	OM_PROP_CRYPTO_SSHA256		SaltedSHA-256
	OM_PROP_CRYPTO_SSHA384		SaltedSHA-384
	OM_PROP_CRYPTO_SSHA512		SaltedSHA-512

9.5 About Offline Authentication

Offline authentication is supported for the Mobile and Social authentication flow.

The OMAuthenticationRequest object that is passed to the startAuthenticationProcess:presenterViewController: method has a property called connectivityMode. This property accepts values from the enum OMConnectivityMode that is defined in the OMAuthenticationRequest.h file.

The values are described as follows.

- **`OMConnectivityOnline`** - Always authenticates with the server. Fails if the device cannot reach the Internet.
- **`OMConnectivityOffline`** - Authenticates locally with cached credentials. Offline authentication happens even if the device is online and can reach the server.
- **`OMConnectivityAuto`** - Authentication happens with the server if the server is reachable. Otherwise, authentication happens offline if the device is not connected to the Internet.

Because offline authentication is part of `OMAuthenticationRequest`, the setting is valid for the current request alone. During offline authentication if the number of failure attempts exceeds the value of the `OM_PROP_MAX_LOGIN_ATTEMPTS` setting (discussed in the "About Initialization Properties for M&S Authentication" section), the locally stored credentials are wiped off and online authentication is attempted.

> **Note:** Offline authentication only works if the Offline Authentication" setting in the server is set to "Allowed." See "Editing or Deleting an Application Profile" in the *Administrator's Guide for Oracle Access Management* for more information.

9.6 Invoking Social Identity Authentication

Social Identity authentication is a feature of the Oracle Mobile and Social server that enables authentication against third party OpenId and OAuth providers like Google, Twitter, Facebook, and so on. Refer to the "Configuring Social Identity" chapter in the *Administrator's Guide for Oracle Access Management* for more information.

For Social Identity authentication (also known as *Relying Party* authentication), the SDK can be invoked using the same APIs that you use to authenticate against the Mobile and Social server. When you configure the Service Domain for authentication against the Social Identity Service, the SDK automatically follows the Relying Party authentication flow. After successful authentication, control is returned back to the app using the URL scheme configured in the Mobile and Social server.

Add this code snippet to your application in the `application:openURL:sourceapplication:annotation` method of the `UIApplicationDelegate` object to complete the flow. This helps pass the information received from the server to the SDK to signify that authentication has completed.

```
// Called when application in invoked via some URL
- (BOOL)application:(UIApplication *)application
            openURL:(NSURL *)url
  sourceApplication:(NSString *)sourceApplication
         annotation:(id)annotation
{

    ...
    ...

    NSMutableDictionary *userInfo = [[NSMutableDictionary alloc]
                                     initWithCapacity:1];
    [userInfo setObject:url forKey:OM_RESPONSE_URL];

    //Post the notification to IDM Mobile SDK
    [[NSNotificationCenter defaultCenter]
     postNotificationName:OM_PROCESS_URL_RESPONSE
```

```
                                object:self
                              userInfo:userInfo];

    ...
    ...

    return TRUE;
}
```

After authentication succeeds, you can access the token from the provider (if any) with the key `OM_ACCESS_TOKEN` from the dictionary `OMAuthenticationContext.accessTokens`. If `emailid` is configured, it is available as `OMauthenticationContext.userName`.

9.7 Invoking User Profile Services With the iOS Client SDK

Before working with the code samples in this section, see "Introduction to 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
- Using the Asynchronous API

9.7.1 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.

```
- (id)getAttribute: (NSString *)attrName returningError: (NSError **)error;

- (NSArray *)searchUsersWithFilter: (NSDictionary *)filter
                                       isSimpleSearch: (BOOL)simpleSearch
                                attributesToBeFetched: (NSArray *)attributesToFetch
                                            pageSize: (NSInteger)pageSize
                                        pagePosition: (NSInteger)pagePosition
                                               error: (NSError **)error;

- (OMAsyncOpHandle *)getAttributeAsynchronously: (NSString *)attrName;

- (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;
```

9.7.2 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;
```

9.7.3 Working With Organizations

Use the following APIs to request information about managers and their reports. They are available in `OMUser`.

- Use the following APIs to get a user's manager.

```
- (OMUser *)getManager: (NSError **)error;
- (OMAsyncOpHandle *)getManagerAsynchronously;
```

- Use the following APIs to retrieve a given user's report.

> **Note:** You need to implement a delegate to get results from the asynchronous call. For details see Chapter 9.7.4, "Using the Asynchronous API."

```
- (NSArray *)getReporteesWithAttributes: (NSArray *)attributes returningError:
(NSError **)error;
- (OMAsyncOpHandle *)getReporteesAsynchronouslyWithAttributes:(NSArray
*)attributes;
```

9.7.4 Using the Asynchronous API

Follow these steps to use the User Profile Service's asynchronous API:

1. The calling class has to implement `OMEntityDelegate`.
2. Implement the delegate method `-didReceiveEntities:from:withAsynchronousHandle:`

The following code snippet demonstrates the user search operation:

```
    OMUserManager *userManager = [[mss userRoleProfileService] getUserManager];
    userManager.delegate = self; //class that implements OMEntityDelegate
    [userManager searchUsersAsynchronouslyWithFilter:filter isSimpleSearch:YES
attributesToBeFetched:attributes pageSize:pageSize pagePosition:pagePosition];

// This method receives the asynchronous operation result
-(void)didReceiveEntities:(id)entities error:(NSError *)error from:(id)omObject
       withAsynchronousHandle:(OMAsyncOpHandle *)asyncHandle
```

9.8 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*.

9.8.1 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.

   The Launch Pad opens.

2. Under **Access Manager** click **Authentication Schemes**, then click the **Create Authentication Scheme** 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.

9.9 Authenticating Using Client Certificate

IDM Mobile SDK supports 2-way SSL mutual authentication. In addition to the client validating the identity of the server, if the authentication server is configured to perform 2-way SSL, the server will require the client certificate to validate the identity of the client. In other words, this is an additional authentication scheme where the server identifies the client, to avoid anonymous access.

Authentication based on client certificates (mutual authentication) is more secure than authentication based on the username and password method. This is because passwords do not have high entropy and it makes them vulnerable to brute force attacks. In addition, complex passwords are difficult to remember. In such a situation, client certificates are preferred as they are complex and use asymmetric cryptography to make them less vulnerable to attacks.

IDM Mobile SDK provides APIs that can be used to import PKCS12-encoded client certificates and to use these certificates to provide authentication, while making a network connection through SDK.

The following sections contain more information.

- Importing a Client Certificate
- Performing Standalone Authentication
- Performing Mixed Mode Authentication

9.9.1 Importing a Client Certificate

The client certificate should be delivered at runtime to the mobile application but it can also be packaged while developing the application.

Note: For information about how to deliver a file at runtime, see the "Registering the File Types Your App Supports" page at the following link:

`https://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/DocumentInteraction_TopicsForIOS/Articles/RegisteringtheFileTypesYourAppSupports.html`

The file is placed in "Documents" directory which is only available to the application. You must call an IDM Mobile SDK API that imports the certificate to the keychain and also deletes the file.

The two following APIs for importing client certificates are available in the `OMCertService` class:

- `+(void)importClientCertificateFromFile:(NSURL *)fileURL presenter:(UIViewController *)presenter delegate:(id)delegate`

  Refer to Section 9.9.1.1, "importClientCertificateFromFile:presenter:delegate:" for more information.

- `+(NSArray *) importClientCertificateFromFile:(NSURL *)fileURL password:(NSString *)password error:(NSError **)error;`

  Refer to Section 9.9.1.2, "importClientCertificateFromFile:password:error:" for more information.

9.9.1.1 importClientCertificateFromFile:presenter:delegate:

This is an asynchronous API that imports the certificate and the result is delivered through a `OMCertServiceDelegate` protocol method.

- **fileURL** is the local URL of the certificate file.
- **presenter** is a UI View Controller in which IDM Mobile SDK will present its UI to collect the password.
- **delegate** is the `OMCertServiceDelegate` object whose `-(void)didImportClientCertificate:(NSArray *)certInfo error:(NSError *)error` method will be called.

The `certInfo` object is an NSArray of NSDictionary objects, which contain information about each of the imported client identities.

9.9.1.2 importClientCertificateFromFile:password:error:

This is synchronous method that imports the certificate and returns an NSArray of NSDictionary objects. These objects will contain information about the imported client identities if the execution of the method is successful. If an error is encountered, the passed error reference is populated.

- **fileURL** is the local URL of the certificate file.
- **password** is the password used to decrypt the file.
- **error** is the reference to a NSError object that will be populated if an error occurs while importing the certificate.
- **Return Value** is the an array of certificate information returned if the import is successful, else the values is nil. No information is returned.

9.9.2 Performing Standalone Authentication

IDM Mobile SDK provides client certificate-based authentication as a separate authentication service. The following procedure describes the standalone authentication process:

1. The following is a sample code for standalone client certificate authentication:

```
NSMutableDictionary *props = [[NSMutableDictionary alloc] init];
[props setObject:OM_PROP_AUTHSERVER_CLIENT_CERT forKey:OM_PROP_AUTHSERVER_
TYPE];
[props setObject:@"https://host:port/resource" forKey:OM_PROP_LOGIN_URL];
[props setObject:@"ClientCertApp" forKey:OM_PROP_APPNAME];
self.mss = [[OMMobileSecurityService alloc] initWithProperties:props
delegate:self];
[self.mss setup];
```

2. This preceding code will initialize the SDK with client certificate authentication. There is no application profile that needs to be downloaded. However, the `mobileSecurityService: didReceiveApplicationProfile:error:` method of the `OMMobileServiceDelegate` protocol will be called. Then, the authentication process can be started using the following sample code:

```
[self.mss startAuthenticationProcess:nil presenterViewController:self];
```

3. If a client certificate challenge is received by the IDM Mobile SDK and client certificates are installed, the user will be presented with a screen where an appropriate client identity can be selected. Then, if the server accepts the client certificate, the authentication will succeed and the `mobileSecurityService:didFinishAuthentication:error:` method of the `OMMobileServiceDelegate` protocol will be called.

9.9.3 Performing Mixed Mode Authentication

Certificate-based authentication can be integrated with other authentication schemes as well. During an authentication scheme, if a client certificate challenge arises, then IDM Mobile SDK will present the list of installed certificates to the user. The user must choose a certificate, after which the authentication will proceed according to the authentication scheme. To ensure that the client certificate authentication option is present with other authentication schemes, the following additional property must be added while initializing the SDK:

```
[props setObject:@"TRUE" forKey:OM_PROP_PRESENT_CLIENT_IDENTITY_ON_DEMAND];
```

The default value of this property is `false`.

9.10 Understanding and Using OAuth2.0 for iOS SDK

The IDM Mobile SDK provides for authorization with the OAM (M&S) OAuth Server to access protected resources. The SDK will also work with any OAuth2.0 generic server, if it supports the grant types mentioned in Section 9.10.1.1.2, "Access Token Retrieval" for mobile clients.

After initializing the IDM Mobile SDK with the correct properties and after authentication, the application must be able to get the access tokens.

The following is a list of new properties that have been added for OAuth2.0:

Configuration Property	Valid Value	Mandatory
OM_PROP_ AUTHSERVER_ TYPE (AuthServerType)	String Constant OM_PROP_ OAUTH_OAUTH20_ SERVER	Yes
OM_PROP_ OAUTH_ AUTHORIZATION_ GRANT (OAuthAuthZGrantType)	String Constant OM_OAUTH_ AUTHORIZATION_ CODE OM_OAUTH_ IMPLICIT OM_OAUTH_ RESOURCE_ OWNER OM_OAUTH_ CLIENT_ CREDENTIALS OM_OAUTH_ ASSERTION OM_OAUTH_OAM_ CREDENTIAL	Yes
OM_PROP_ OAUTH_ AUTHORIZATION_ ENDPOINT (OAuthAuthZEndpoint)	NSString	Yes
OM_PROP_ OAUTH_TOKEN_ ENDPOINT (OAuthTokenEndpoint)	NSString	Mandatory if the OM_PROP_ OAUTH_ AUTHORIZATION_ GRANT is AUTHORIZATION_ CODE

Configuration Property	Valid Value	Mandatory
OM_PROP_ OAUTH_ REDIRECT_ ENDPOINT (OAuthRedirectEndpoint)	NSString	Yes
OM_PROP_ OAUTH_CLIENT_ID (OAuthClientID)	NSString	Yes
OM_PROP_ OAUTH_SCOPE (OAuthScope(s))	NSSet	Yes
OM_PROP_ BROWSER_MODE (BrowserMode)	OM_PROP_ BROWSERMODE_ EMBEDDED or OM_PROP_ BROWSERMODE_ EXTERNAL	No However, the default will be set to `EXTERNAL`, if a value is not set during initialization.
OM_PROP_ OAUTH_CLIENT_ SECRET (OAuthClientSecret)	NSString	No
OM_PROP_OAM_ OAUTH_SERVICE_ ENDPOINT (OAMOAuthService Endpoint)	NSString	No **Note:** This is only required for OAM OAuth mobile clients
OM_PROP_ OAUTH_ ASSERTION_JWT (OAuthUserJWTAssertionValue)	NSString	No
OM_PROP_ OAUTH_ ASSERTION_SAML2 (OAuthUserSAML2UserAssertionValue)	NSString	No
OM_PROP_ OAUTH_CLIENT_ ASSERTION_SAML2 (OAuthClientSAML2 AssertionValue)	NSString	No
OM_PROP_ OAUTH_CLIENT_ ASSERTION_JWT (OAuthClientJWTAssertionValue)	NSString	No

9.10.1 OAM Mobile and Social (M&S) OAuth

The OAM M&S server supports the following OAuth clients, which can be used with mobiles.

- **Web Clients:** These clients can use the IDM Mobile SDK with the OAuth Generic flows as discussed below. The web-based clients are considered as trusted clients and possess a client secret. Therefore, the authorization server knows the identity of the client that is making the request. When using this client type, the application should pass the client secret during SDK initialization.
- **Mobile Clients:** Mobile clients are not considered to be confidential clients and hence, these clients do not posses any client secret. The OAM OAuth server provides a way by which the mobile clients can register, by sending the device claims along with username and password of the end user. This flow is proprietary to the OAM OAuth server. After dynamic client registration is performed, the IDM Mobile SDK receives a client assertion and a user assertion (if the "SERVER SIDE SSO" is disabled). The client assertion is required to be sent in all the subsequent requests for access token acquisition or for other token operations. IDM Mobile SDK manages the life cycle of the client assertion and does not return this to the client application.

New Token Type	Significance
Client Assertion	This token is generated after the dynamic client registration is done. This token must be sent in every request made to the authorization server. The server identifies the request or client from the client assertion and it validates the same. Client assertions are usually JWT tokens.
User Assertion	Since dynamic client registration involves user authentication, M&S along with client assertions generates a user assertion. This user assertion marks the user session at the server side. The user assertion is returned by the server only when the SERVER SIDE SSO is disabled. The client or SDK can thus use the same assertion for acquiring the access token for resource access every time, till the user session is valid at the server. Also when the SERVER SIDE SSO is enabled, the SDK or client can still use the user assertion by adding the following parameter in every request: `use_server_side_device_store = true`

9.10.1.1 Authentication

After the setup is done, application can now perform authentication which in this case is to get an access token for accessing the OAuth protected resources. The authentication flow is not different from the other flows.

In iOS, the OMMobileSecurityService's startAuthenticationProcess:nil presenterViewController:presenter is used to initiate the authentication process. After completing, the SDK calls back (void)mobileSecurityService: (OMMobileSecurityService *)mobileSecurityServicedidFinishAuthentication: (OMAuthenticationContext *)contexterror: (NSError *)error of OMMobileServiceDelegate.

If the authentication is successful the context will contain the access token along with the other axillary tokens like user_assertion if available. Also, if the authentication was not successful the authentication context will be `null`.

This flow Authentication involves the following steps:

- Dynamic Client Registration
- Access Token Retrieval

9.10.1.1.1 Dynamic Client Registration

Mobile clients are usually not considered to be confidential hence server does not create any secrets for these clients. However, OAM M&S server provides a way by which the client can register itself dynamically to get a client token (client_assertion) which is tied to the user and the device from which the registration is taking place. The device info is useful for the server in order to track the user and the device so that it can trigger it OAAM plugin if applicable. For more information refer to the server side documentation in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

OAM M&S supports two ways of performing client registration: 2-legged client registration and 3-legged client registration. The client registration flow is decided by the SDK based on the grant type provided by the application during initialization. The property used to define the grant type is OM_PROP_OAUTH_AUTHORIZATION_GRANT_TYPE.

- **2-Legged Client Registration:** This is done when the grant type is either RESOURCE_OWNER or ASSERTION or CLIENT_CREDENTIALS or OAM_CREDENTIALS. After end of this flow SDK gets the client_assertion along with user assertion token.

  Note: The availability of user_assertion on the client side is dependant on the value that is set for the SERVER_SIDE_SSO property in the server.

- **3-Legged Client Registration:** This is done when the grant type is AUTHORIZATION_CODE. This flow will invoke the external or embedded browser based on the initialization property. Usually after end of this flow IDM Mobile SDK gets only the client_assertion token.

9.10.1.1.2 Access Token Retrieval

After the dynamic client registration is done and the SDK has a valid client_assertion token it can proceed with the access token acquisition flow. The access token can be acquired using the grant type specified using the initialization property OM_PROP_OAUTH_AUTHORIZATION_GRANT. Table 9–3 contains information on the grant types and the appropriate property value for each.

Table 9–3 Grant Types for Getting the Access Token

Grant Type	Initialization Property Value	Notes
OAM Credential Grant Type	The value of the initialization property should be OM_OAUTH_OAM_CREDENTIAL.	In 2-legged client registration flow, the SDK will obtain a client_assertion and a user_assertion token based on the value set for the SERVER_SIDE_SSO property. To complete the authentication flows, SDK will first exchange the client_assertion to get the access token for OAuth protected resources, and then use the client_assertion and user_assertion (if available) to get the OAM_ID(USERTOKEN_MT) and OBSSOCookie(USERTOKEN) tokens. This grant type is provided by the server for the clients who use the OAM_ID cookie to access the webgate protected resources.
Resource Owner Grant Type	The value of the initialization property should be OM_OAUTH_RESOURCE_OWNER.	2-legged client registration is performed. The SDK collects the user name and password in order to complete the 2-legged client registration. Hence the SDK will reuse these user credentials to get the access token. The SDK does not honor the SERVER_SIDE_SSO property sent by the server in this flow and will always send the user credentials to get an access token. If the client assertion is expired or invalidated, the SDK will again ask for the end user credentials to first renew the client assertion and then get the access token.

Table 9–3 (Cont.) Grant Types for Getting the Access Token

Grant Type	Initialization Property Value	Notes
Client Credentials Grant Type	The value of the initialization property should be OM_OAUTH_CLIENT_CREDENTIALS.	2-legged client registration is performed. To complete the authentication flow, the SDK uses the client_assertion obtained after client registration to get the access token. If the client assertion is expired or invalidated, the SDK will ask for the end user credentials to first renew the client assertion and then get the access token.
Assertion Grant Type	The value of the initialization property should be OM_OAUTH_ASSERTION.	In 2-legged client registration flow, the SDK gets a client_assertion token and a user_assertion token. The user_assertion token is made available by server only if the SERVER_SIDE_SSO is false. To complete the authentication flow the SDK uses the user assertion obtained after client registration to get the access token. This grant type honors the SERVER_SIDE_SSO configuration property from the server by appropriately forming the request based on the property value. If the SERVER_SIDE_SSO property is set to `true`, the SDK adds the `oracle_use_server_device_store=true` parameter in the access token request. In doing so, it requires the server to reuse the user session that was created at the server side. Also if oracle_use_server_device_store=true, then the client does not need to send the user assertion every time. Applications can provide custom assertions for this flow using the initialization property OM_PROP_OAUTH_ASSERTION_JWT for JWT assertion and OM_PROP_OAUTH_ASSERTION_SAML2 for SAML2 assertions. The SDK will use this assertion for access token acquisition to complete the authentication flow.
Authorization Code Grant Type	The value of the initialization property should be OM_OAUTH_AUTHORIZATION_CODE.	Involves the 3-legged client registration flow in which the SDK receives only the client_assertion. During this flow, the SDK invokes either the external or embedded browser based on the initialization property OM_PROP_BROWSER_MODE. The SDK must invoke the browser twice to complete the authentication flow. The first time is for client registration and the second time is for getting the access token. NOTE: If the application provides the value as BrowserMode.EXTERNAL for the OM_PROP_BROWSER_MODE property, the user may not have to enter the credentials twice in the external browser. This is because the server will set the OAM_ID cookie in the external browser during client registration flow itself, and so, during the access token acquisition the browser will not prompt for the login credentials.

9.10.2 Standard Flows (Generic Implementation)

Note: Client secret has to be supplied for confidential clients.

IDM Mobile SDK supports authorization against any OAuth2.0 compliant server. The Current implementation supports the following grant types:

- Implicit grant type. (The "client id" and "authorization endpoint" are required)
- Authorization code grant type. (The "client_id," "authorization end point," and "token endpoint" are required)
- Resource Owner grant type. (The "client_id" and "token end point" are required)

- Client Credentials. (The "client_id" and "client_secret" are required)
- Assertion. (The application must provide an assertion value using either the `OM_PROP_OAUTH_ASSERTION_JWT` or `OM_PROP_OAUTH_ASSERTION_SAML2` property, based on the type of assertion)

9.10.2.1 Authentication Scopes

The SDK accepts a list of scopes to authenticate the user (get a valid access token) with the OAuth2.0 server.

After the authentication of the user and user consent with the Authorization server, the access token returned by the Authorization server is retrieved by the IDM Mobile SDK and is associated initially with the given scopes.

So, before the start of any authentication process, the SDK checks for the availability of any valid access token for the scopes asked in request. If matching access tokens exist or if they are wider than the requested scopes, then the SDK will not authenticate again. It will instead return the same authentication context or access token.

The IDMMobile SDK provides the ability to refresh the access token if it is supported by the authorization server. The life cycle of an access token is controlled by the SDK. The SDK provides the API to get the access token for passed scopes, and the ability to refresh it if required.

The following is the sample code for initialization:

```
                    NSMutableDictionary  *sdkProps = [[NSMutableDictionary alloc]
init];
        [sdkProps setObject:OM_PROP_OAUTH_OAUTH20_SERVER

                     forKey:OM_PROP_AUTHSERVER_TYPE];
        [sdkProps setObject:@"<Token Endpoint>"

                     forKey:OM_PROP_OAUTH_TOKEN_ENDPOINT];
        [sdkProps setObject:@"<Authorization Endpoint>"

                     forKey:OM_PROP_OAUTH_AUTHORIZATION_ENDPOINT];
        [sdkProps setObject:@"<Client ID>"

                     forKey:OM_PROP_OAUTH_CLIENT_ID];
        [sdkProps setObject:OM_OAUTH_AUTHORIZATION_CODE

                     forKey:OM_PROP_OAUTH_AUTHORIZATION_GRANT];/*The value could
be any of the grant type supported by SDK*/
        [sdkProps setObject:@"<Registered Redirect URL>"

                     forKey:OM_PROP_OAUTH_REDIRECT_ENDPOINT];
        [sdkProps setObject:OM_PROP_BROWSERMODE_EXTERNAL

                     forKey:OM_PROP_BROWSERMODE];
         NSMutableArray *scope = [[NSMutableArray alloc] init];
[scope addObject:@"<OAuth Scope>"];
[sdkProps setObject:[NSSet setWithArray:scope] forKey:OM_PROP_OAUTH_SCOPE];

        OMMobileSecurityService *mss = nil;
        mss = [[OMMobileSecurityService alloc] initWithProperties:sdkProps

                                                          delegate:self];
        self.mss = mss; /*Please note that initialized SDK object need to be
retained atleast till the authentication completes.*/
```

If the initialization process is successful, an Object for OMMobileSecurityService will be returned. A nil return value signifies that there are some invalid configuration parameters. You must reconfigure the parameters and retry the initialization process.

After initialization, you must set up the SDK. To do so, use the following API:

```
[self.mss setup];
```

The SDK will download the M&S mobile client in the setup call. This is an asynchronous API.

After the setup completes, you must call the `startAuthenticationProcess:presenterViewController` API on the mss object to start the OAuth flow.

A list of scopes can be passed to this API as part of OMAuthenticationRequest object. If passed, these scopes will override the ones passed during the following operation:

```
        OMAuthenticationRequest *req = [[OMAuthenticationRequest alloc] init];

        NSMutableArray *scope = [[NSMutableArray alloc] init];

        [scope addObject:@"<OAuth Scope>"];

        req.oauthScope = [NSSet setWithArray:scope];

        [scope release];

        NSError *error = [self.mss startAuthenticationProcess:req
presenterViewController:self];
```

Note: The application must implement the `didFinishAuthentication: delegate` method of `OMMobileSecurityServiceDelegate,` to retrieve control after authentication.

The following is the sample code for M&S mobile client initialization:

```
 NSMutableDictionary *sdkProps = [[NSMutableDictionary alloc] init];

[sdkProps setObject:OM_PROP_OAUTH_OAUTH20_SERVER

forKey:OM_PROP_AUTHSERVER_TYPE];

[sdkProps setObject:@"<M&S servers service endpoint for the desired OAuth service
profile>"

forKey:OM_PROP_OAUTH_OAM_SERVICE_ENDPOINT];

[sdkProps setObject:@"<Mobile Client ID>"

forKey:OM_PROP_OAUTH_CLIENT_ID];

[sdkProps setObject:OM_OAUTH_RESOURCE_OWNER

forKey:OM_PROP_OAUTH_AUTHORIZATION_GRANT_TYPE];

NSMutableArray *scope = [[NSMutableArray alloc] init];
```

```
[scope addObject:@"<OAuth Scope>"];

[sdkProps setObject:[NSSet setWithArray:scope] forKey:OM_PROP_OAUTH_SCOPE];
```

9.10.3 New APIs

The three following API's have been added to `OMAuthenticationContext` class:

- **-(BOOL)isValidForScopes:(NSSet *)scopes refresh:(BOOL)refresh:**

  This API takes a set of scopes and a bool to refresh, if a refresh token exists for a matching token.

- **-(NSArray *)accessTokensForScopes:(NSSet *)scopes:**

  This API takes a set of scopes and return all valid tokens which are tied to that set of scopes. If `nil` is passed as scope, then all valid access tokens are returned.

  Note: The OAuth tokens are present in an NSArray (named tokens), in the `OMAuthenticationContext` object.

- **-(void)checkValidityAsynchronouslyForScopes:(NSSet*)scopes andRefreshExpiredTokens:(BOOL)refresh:**

  This API takes a set of scopes and a bool to refresh, if a refresh token exists for a matching token. It is an "async call" which will tell if a valid token exists for passed scopes. This API will try to refresh an expired token if the value for `refreshExpiredToken` is set to `true`. An application developer must implement the `OMAuthenticationContextDelegate` method to get the delegate callback.

Sample Code:

```
- (void)checkAuthContextValidity

{

      self.authContext.delegate = self; //This delegate needs to be set before
calling the async API
      NSMutableArray *scope = [[NSMutableArray alloc] init];
     [scope addObject:@"UserProfile.users"];
     [self.authContext checkValidityAsynchronouslyForScopes:[NSSet
setWithArray:scope] andRefreshExpiredTokens:YES];
}
//This is the OMAuthenticationDelegate method which needs to be implemented for
getting the async API callback
- (void)authenticationContext:(OMAuthenticationContext *)authenticationContext
          didFinishValidation:(BOOL)valid
     forMobileSecurityService:(OMMobileSecurityService *)mobileSecurityService
                     andError:(NSError *)error
{
    if(valid)
    {
          NSMutableArray *scope = [[NSMutableArray alloc] init];
         [scope addObject:@"UserProfile.users"];
         NSArray *token = [authenticationContext tokensForScopes:[NSSet
setWithArray:scope]]; //This token can be used for subsequent //requests
    }
    else
    {
         [mobileSecurityService startAuthenticationProcess:nil
```

```
presenterViewController:self];//Ifnovalidtokenexistsforpassedscopes
//authentication can be started
    }
}
```

9.10.4 Using the External Browser

If the delegate is not implemented, you must implement it using the following snippet of sample code:

```
application:openURL:sourceApplication:annotation method of UIApplicationDelgate :
        NSMutableDictionary *userInfo = [[NSMutableDictionary alloc]
initWithCapacity:1];
        [userInfo setObject:url forKey:OM_RESPONSE_URL];
        //Post the notification to IDM Mobile SDK
        [[NSNotificationCenter defaultCenter] postNotificationName:OM_PROCESS_URL_
RESPONSE
                                                            object:self
                                                          userInfo:userInfo];

        [userInfo release];
```

9.10.5 Accessing Protected Resources

Use the following API in the `OMRESTRequest` request to insert Access Tokens in a header before accessing an OAuth Protected resource:

```
- (OMAsyncOpHandle *)executeRESTRequestAsynchronously: (NSMutableURLRequest
*)request

                                          forScopes: (NSSet *)scopes

                                  convertDataToJSON: (BOOL)isJsonRepresentation
```

This API executes the given REST request asynchronously, after adding access tokens to header for given scopes, and returns a handle to the asynchronous operation. The caller can abort the asynchronous request using this handle at any time before the request is completed. The REST request is executed in a separate thread and the result is returned through the `OMRESTRequestDelegate` delegate. The following flows are supported in OAuth SIM v3.1:

- Client_credentials flow with id & secret
- Client_credentials flow with JWT Bearer Token signed with Client's private key
- Resource owner password flow with client id & secret and resource owner password
- Resource owner password flow with JWT Bearer Token signed with Client's private key and resource owner password
- Jwt-bearer(OM_OAUTH_ASSERTION) flow with Client JWT Token and User JWT Token signed using the Client's private Key
- Jwt-bearer(OM_OAUTH_ASSERTION) flow with Client JWT Token and User JWT Token Issued by SIM Server

The following sections have some details.

- Initializing the SDK for Identity Domain Header Injection
- Initializing the SDK for Client Token

- Initializing the SDK for User Token

9.10.5.1 Initializing the SDK for Identity Domain Header Injection

The following property needs to be set while initializing the SDK for Identity domain header Injection:

- OM_PROP_IDENTITY_DOMAIN_NAME

  The value of this parameter should contain Identity Domain name

- OM_PROP_IDENTITY_DOMAIN_NAME_IN_HEADER

  This property is introduced to maintain backward compatibility as till now SDK used to prefix identity domain with username. Pass true for this property if identity domain needs to be passed in header.

9.10.5.2 Initializing the SDK for Client Token

The following property needs to be set while initializing the SDK for Client token:

- OM_PROP_OAUTH_CLIENT_ASSERTION_JWT

  It should contain value for Client JWT token

- OM_PROP_OAUTH_CLIENT_ASSERTION_SAML2

  It should contain value for Client SAML2 token. However, the SIM server only supports JWT Token currently, so this property can be ignored.

9.10.5.3 Initializing the SDK for User Token

The following property needs to be set while initializing the SDK for User token:

- OM_PROP_OAUTH_ASSERTION_JWT It should contain value for User JWT token.

- OM_PROP_OAUTH_ASSERTION_SAML2

  It should contain value for User SAML2 token. However, the SIM server only supports JWT Token currently, so this property can be ignored.

The following is the sample code for Android for SIMv3.1:

> **Note:** Read the comments corresponding to each property in the sample code and only the property names which are required for the application usage must be passed during SDK initialization. Apart from the initialization properties, every thing else remains same i.e. authenticate , isValid, logout.

```
private Map<String, Object> getConfigMapForSIMv31() {
    Map<String, Object> configMap = new HashMap<String, Object>();
    configMap
            .put(OMMobileSecurityService.OM_PROP_APPNAME, "DemoSIMv3.1App");
    configMap.put(OMMobileSecurityService.OM_PROP_AUTHSERVER_TYPE,
            AuthServerType.OAuth20);
    configMap.put(OMMobileSecurityService.OM_PROP_OAUTH_CLIENT_ID,
            "client_id");
    configMap.put(OMMobileSecurityService.OM_PROP_OAUTH_CLIENT_SECRET,
            "client_secret");
    configMap.put(OMMobileSecurityService.OM_PROP_OAUTH_TOKEN_ENDPOINT,
            "http://www.example.com:1234/token_endpoint");\\
```

```
        configMap.put(
                OMMobileSecurityService.OM_PROP_OAUTH_AUTHORIZATION_ENDPOINT,
                "http://www.example.com:1234/authz_endpoint");\\
        configMap
                .put(OMMobileSecurityService.OM_PROP_REMEMBER_USERNAME_ALLOWED,
                        true);
        configMap.put(
                OMMobileSecurityService.OM_PROP_OAUTH_AUTHORIZATION_GRANT_TYPE,
                OAuthAuthorizationGrantType.RESOURCE_OWNER);

        configMap.put(OMMobileSecurityService.OM_PROP_IDENTITY_DOMAIN_NAME,
                "DemoTenant");// If this is supplied then the SDK will not
                                // expect the idenity domain to be collected
                                // from user or from the authentcationRequest
                                // object. Instead use the same identity domain
passed during init.
        configMap.put(
                OMMobileSecurityService.OM_PROP_IDENTITY_DOMAIN_NAME_IN_HEADER,
                true);// This property is required if the application wants the
                        // SDK to send the identity domain as the header.


        configMap.put(OMMobileSecurityService.OM_PROP_OAUTH_ASSERTION_JWT,
                "user assertion");// required when the application wants to use
                                    // assertion as a grant type.
        // configMap
        configMap.put(OMMobileSecurityService.OM_PROP_LOGOUT_URL,
                TesterAppConstants.OAUTH_LOGOUT_URL);
        configMap
                .put(OMMobileSecurityService.OM_PROP_SESSION_ACTIVE_ON_RESTART,
                        true);// required if the application wants the SDK to
                                // persist the authentication context, The
                                // default behavior is not to persist the
                                // authentication context thus forcing re
                                // authentication on application restart.
        configMap.put(OMMobileSecurityService.OM_PROP_OAUTH_REDIRECT_ENDPOINT,
                "myredirecturl://");
        String scope = "scope1,scope2,scope3";
        Set<String> scopeset = new HashSet<String>();
        if (!TextUtils.isEmpty(scope)) {
            String[] scopes = scope.split(",");
            for (String s : scopes) {
                scopeset.add(s);
            }
        }
        if (!scopeset.isEmpty())
            configMap
                    .put(OMMobileSecurityService.OM_PROP_OAUTH_SCOPE, scopeset);
        configMap
                .put(OMMobileSecurityService.OM_PROP_REMEMBER_USERNAME_ALLOWED,
                        true);
        configMap.put(OMMobileSecurityService.OM_PROP_OFFLINE_AUTH_ALLOWED,
                true);
        configMap.put(OMMobileSecurityService.OM_PROP_BROWSER_MODE,
                TesterAppConstants.OAUTH_BROWSER_MODE);

        return configMap;
    }
```

9.10.6 Credential Collection

For the `IMPLICIT` and `AUTHORIZATION_CODE` grant types, the SDK invokes an external or embedded browser. The authorization server will load the login page in the browser and so, the end user submits the login credentials directly to the authorization server.

For the `RESOURCE_OWNER`, `CLIENT_CREDENTIALS`, `ASSERTION`, and `OAM_CREDENTIAL` grant types, the SDK will provide a native UI to collect the login credentials of the end user by default. However, the application can direct the SDK to provide a custom view or UI for credential collection. Refer to Section 9.13, "Login and KBA View Customization" for more information.

Note: Only the "remember user name flag" can be used with OAuth2.0 because the resource owner grant type SDK collects the credentials. Refer to Section 9.16, "Using the Credential Store Service (KeyChain)" for more information.

9.11 Invoking REST Web Services

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 values configured in the OAM 11g R2
    // WebGate user defined parameters.
    // We need to configure OAM 11g R2 WebGate with these parameters:
    //  i) OAMAuthUserAgentPrefix=<Prefix for User-Agent HTTP header> (Example: OIC)
    // ii) OAMAuthAuthenticationServiceLocation=<OIC Server URL> (Example:
    //     http://host123.us.example.com:14100/oic_rest/rest/mobileoamauthentication)
    [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 `OAMAuthUserAgentPrefix` and `OAMAuthAuthenticationServiceLocation` properties. The same property values 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.

9.11.1 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.

9.12 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 or converting an application to 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*).

Follow these steps to initialize and configure the SDK.

1. Introduce the following instance variable to the class implementing `OMMobileServiceDelegate`:

```
BOOL _setupDone; //Indicates if the application profile was downloaded
successfully
NSString *_ssoAppBundleID; //Store the bundle ID of the SSO client app
NSURL *_ssoRequestURL; // Stores the request URL sent by the client app
BOOL _profileError; //Indicates if an error occurred in the profile download
```

```
NSDictionary *_queryParams; // Stores query parameters from the SSO request URL
```

2. Add the following methods to the same class:

```
/* This method is the entry point to handle SSO requests. If the app Profile is
not downloaded, it starts the download or it starts by processing the request.
The App profile download can be triggered from this function if it is not being
checked every time the app becomes active. */

- (void) handleRequestWithURL:(NSURL *)url bundleID:(NSString *)appBundleID
{
    if (!_setupDone)
    {
        self.ssoRequestURL = url;
        self.ssoAppBundleID = appBundleID;
        return;
    }
    [self ssoRequestWithURL:url fromApp:appBundleID];
}

/* This function invokes the SDK to process the SSO request. */

- (void)ssoRequestWithURL:(NSURL *)url fromApp:(NSString *)appBundleID
{
    if(_profileError)
    {
        _profileError = FALSE;
        self.ssoAppBundleID = nil;
        self.ssoRequestURL = nil;
        return;
    }
    NSDictionary *params = [self.mobileServices parseURL:url
fromApp:appBundleID];
    self.ssoRequestURL = nil;
    self.ssoAppBundleID = nil;
    self.queryParams = params;

    /* If this is an SSO request coming from either a native app or a browser
     * then do the SSO flow. Else do application specific logic. */

    if ([self.mobileServices isSSORequest:self.queryParams])
    {
        [self.mobileServices processSSORequest:self.queryParams
presenter:self]; //It is assumed that the class also extends UIViewController.
    }
}
```

3. Add the following code to the `didReceiveApplicationProfile:error` method:

```
    if (error)
    {
        _profileError = TRUE;
    }
    else
    {
        _setupDone = TRUE;
    }
 if(self.ssoRequestURL != nil && self.ssoAppBundleID != nil)
    {
        [self ssoRequestWithURL:self.ssoRequestURL
fromApp:self.ssoAppBundleID];
```

```
            return;
        }
```

4. Add the following code to the `didFinishAuthentication:error` method:

```
if([self.mobileServices isSSORequest:self.queryParams])
    {
        [self dismissModalViewControllerAnimated:NO];
        [error retain];
        [self performSelector:@selector(completeSSOAuthentication:)
                  withObject:(id)error afterDelay:0]; /* This is required
because the SSO Agent app  starts the client app registration after
authentication. The registration should occur in the next run loop cycle.*/
        return;
    }
- (void)completeSSOAuthentication:(id)object
{
    NSError *error = (NSError *)object;
    [self.mobileServices completeSSORequest:self.queryParams presenter:self
error:error];
}
```

5. Add the following code to the `didFinishRegistration:error` method:

```
    [registrationHandle retain];
    [self dismissModalViewControllerAnimated:true];
    [self.mobileServices sendSSOResponseWithHandles:registrationHandle
error:loginError params:self.queryParams];
    [registrationHandle release];
```

6. Add the following to the `application:openURL:sourceApplication:annotation` method of `UIApplicationDelegate` :

```
/* This starts the SSO request handle process. You can check the request URL to
make sure it is an SSO request. */

    [<object of class implementing handleRequestWithURL:bundleID method>
handleRequestWithURL:url bundleID:sourceApplication];
```

The following is a sample implementation of `application:openURL:sourceApplication:annotation`:

```
- (BOOL) application:(UIApplication *)application
            openURL:(NSURL *)url
   sourceApplication:(NSString *)sourceApplication
          annotation:(id)annotation
{
    self.viewController.isConfigUpdate = false;
    NSString *queryString = [url query];
    NSString *host = [url host];
    if ([queryString hasPrefix:@"oamconfig=true&"])
    {
        //do something
    }
    else if([host isEqualToString:@"settings"])
    {
        //do Something
    }
    else
    {
        [self.viewController handleRequestWithURL:url
```

```
                                                    bundleID:sourceApplication];
    }
    return YES;
}
```

Note that the app delegate of the iOS mobile SSO app should implement the `openURL` method to handle SSO requests coming from other applications. 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.

9.13 Login and KBA View Customization

You can create custom views to get login and knowledge-based authentication (KBA) information from the user. The SDK will present these views to the user as needed. Otherwise, the SDK presents the default native view. The following sections have details.

- Implementing Native View Customization
- Implementing Progress View Customization

9.13.1 Implementing Native View Customization

Custom native views should subclass `OMAuthView` and override some methods described later in this section. When creating custom native views:

- Subclass `OMAuthView`.
- Override the `viewLoaded` method. If any part of the Remember Credentials feature is enabled then the `authData` dictionary will contain credentials and user preferences for the feature.
- Override the `retrieveAuthData` method. This method should populate the `authdata` dictionary with user input.

Following is sample code to illustrate this.

```
MyLoginView.h file
@interface MyLoginView : OMAuthView
   //Declare properties required for getting user input
@end
MyLoginView.m file
@implementation MyLoginView
-(void) viewLoaded
{
   // Get credentials and user preferences from self.authData
}
-(NSDictionary *) retrieveAuthData
```

```
{
   // Populate self.authData with user input
}
@end
```

When passing the custom native views to the SDK:

- Create an `OMAuthenticationRequest` object.
- Set the `authView` and `kbaView` properties to custom native view objects.
- Pass the `OMAuthenticationRequest` object to the `mss.startAuthenticationProcess:presentViewController` method.

Following is sample code to illustrate this.

```
// Initialize mss object
OMMobileSecurityService *mss = [[OMMobileSecurityService alloc]
initWithProperties:sdkProps delegate:self];
[mss setup];
//Wait for setup to complete
//Create authentication request
OMAuthenticationRequest *authnReq = [OMAuthenticationRequest alloc] init];
MyLoginView *loginView = [[MyLoginView alloc] init];
MyKBAView *kbaView = [[MyKBAView alloc] init];
authReq.authView = loginView;
authReq.kbaView = kbaView;
[mss startAuthenticationProcess:authnReq presenterViewController:myPresenter];
```

9.13.2 Implementing Progress View Customization

The progress view shown after the login screen can be customized. The pattern is similar to the Login view customization. The app needs to extend the `OMAuthProgressView` class and register the instance of the app's implementation in `OMAuthenticationRequest`. A code snippet is given below:

```
OMAuthenticationRequest *authReq = [[OMAuthenticationRequest alloc] init];
MyProgressView *myProgressView = [[MyProgressView alloc]
initWithFrame:self.view.frame];
authReq.authProgressView = myProgressView;
[myProgressView release];
```

9.14 Using the Cryptography Module

The cryptography APIs provided by iOS are C APIs that are cumbersome to use. The Mobile and Social iOS Client SDK provides intuitive Objective C APIs for common cryptography tasks that are simple to use. The SDK uses the APIs listed here to protect and store credentials for offline authentication. For detailed information about API methods, please refer to the API documentation included in the SDK download.

The following sections contain high-level summaries of the functionality provided by the cryptography module.

- Hashing
- Symmetric Key Encryption/Decryption
- Asymmetric Key Cryptography

9.14.1 Hashing

Use the hashing capability to get the hash value of a string (typically a password) with an auto-generated random salt. The following API also adds the salt to the output and prefixes the algorithm name to the output. This is suitable for storage and transfer over the network. The generated salt, if requested, can be extracted using the `outSalt` parameter.

```
NSString *secret = "mypassword";
NSString *outSalt;
NSError *error;

NSString *hashValue = [OMCryptoService SHA256HashAndBase64EncodeData:[secret
                                                                      dataUsingEncoding:
                                                                      NSUTF8StringEncoding]
                                                          withSaltOfBitLength:24
                                                                      outSalt:&outSalt
                                                                      outError:&error];
```

The hashing capability provides the following functionality:

- Convenience APIs are provided to perform hashing using the following algorithms SHA1, SHA224, SHA256, SHA384, SHA512
- A similar set of APIs that operate without a salt are also provided
- A random salt generation method is available if the salt needs to be stored separately
- The resulting hash value can be optionally Base64 encoded
- The algorithm name can be optionally prefixed to the output
- In addition to the convenience API, a more sophisticated method that can customize all values is also provided:

```
hashData:withSalt:algorithm:appendSaltToOutput:base64Encode:prefixOutputWithAlgorithmName:outError:
```

9.14.2 Symmetric Key Encryption/Decryption

This API helps perform symmetric key encryption and decryption of data. The following example helps encrypt a given data using AES algorithm, PKCS7 padding using an auto-generated random symmetric key.

```
NSData *plainText = ...
NSString *outKey;
NSError *error;

NSString *cipherText = [OMCryptoService encryptData:plainText
                              withSymmetricKeyOfLength:24
                                       outSymmetricKey:&outKey
                                              outError:&error];
```

To decrypt the above cipher text, the following API can be used.

```
NSString *plainText = [OMCryptoService decryptData:cipherText
                                   withSymmetricKey:outKey
                                           outError:&error];
```

- The supported algorithms are AES128 (key sizes 16,24,32), DES (key size 8), 3DES (key size 24).
- A secure symmetric key generation API is available

- The resultant cipher text can be optionally Base64 encoded
- The algorithm name can be optionally prefixed to the output.
- A more sophisticated method that makes it possible to specify the algorithm, initialization vector, padding, and so on is available:

```
encryptData:withSymmetricKey:initializationVector:algorithm:padding:mode:base64EncodeOutput:prefixO
utputWithAlgorithmName:outError:

decryptData:withSymmetricKey:initializationVector:algorithm:padding:mode:isInputPrefixedWithAlgorit
hmName:isInputBase64Encoded:outError:
```

9.14.3 Asymmetric Key Cryptography

The asymmetric key cryptography APIs that are part of `OMCryptoService` help with the following operations:

- Key pair generation and storage in a key chain
- Sign and verify operations with keys in a key chain
- Wrap and unwrap operations with keys in a key chain
- Extraction of the public key from a key chain
- Deletion of a key pair in a key chain

9.15 Using the Auto Login and the Remember Credentials Features

The Mobile and Social iOS Client SDK provides APIs that can securely store user credentials and play them back to a login server with or without user interaction. Deploy this feature in apps where security is not critically important to make it easier for users to log in. This feature requires at least version 11.1.2.2.0 of the Mobile and Social Client SDK.

To use this feature, the user selects one of up to three option boxes on the login screen. (You can choose to enable one, two, or all three of these options in your app.) If more than one option is selected, the higher priority feature takes precedence. The priority of the options is as follows:

1. Auto Login
2. Remember Credentials
3. Remember User Name Only

So for example, if Auto Login and Remember Credentials are selected, then Auto Login takes priority.

The three options are described as follows:

- Auto Login - The Mobile and Social iOS Client SDK securely caches the user's credentials and automatically supplies them at the login screen. This option does not require any user interaction to log the user in. The user see a progress screen that provides feedback while authentication is underway.
- Remember Credentials - The Mobile and Social iOS Client SDK securely caches the user's credentials and auto-fills the user name and password fields on the login screen. The user has to click the Login button to proceed with the authentication process. The user can enter different credentials and make changes to the option boxes if necessary.

- Remember User Name Only - The Mobile and Social iOS Client SDK securely caches the user name and auto-fills the user name on the login screen. The user has to input the password and click the login button to proceed with the authentication process. The user can enter different user name and make changes to the option boxes if necessary.

The following sections have more details.

- Enabling the Auto Login and Remember Credentials Feature
- Handling User Preferences
- Clearing Credentials and Preferences from Mobile Devices
- Creating a Custom Login Screen

9.15.1 Enabling the Auto Login and Remember Credentials Feature

This is a client-side only feature that does not require server configuration to deploy. To enable the features described, the following SDK configuration parameters should be added to your code:

Table 9–4 Auto Login and Remember Credentials Configuration Parameters

Parameter	Description
`OM_PROP_AUTO_LOGIN_ALLOWED`	Boolean value that enables/disables the Auto Login feature.
`OM_PROP_REMEMBER_CREDENTIALS_ALLOWED`	Boolean value that enables/disables the Remember Credentials feature.
`OM_PROP_REMEMBER_USERNAME_ALLOWED`	Boolean value that enables/disables the Remember User Name Only feature.

9.15.2 Handling User Preferences

Pass the following properties while initializing the `mss` object to pre-populate the option boxes with a default value. Only one of the following options can be true. If all the option box states are false, all of the option boxes on the login screen will be empty.

Table 9–5 Configuration parameters used to set the option box default values

Parameter	Description
`OM_AUTO_LOGIN_DEFAULT`	Boolean value that specifies the default value of the "Auto Login" option box.
`OM_REMEMBER_CREDENTIAL_DEFAULT`	Boolean value that specifies the default value of the "Remember Credential" option box.
`OM_REMEMBER_USERNAME_DEFAULT`	Boolean value that specifies the default value of the "Remember User name Only" option box.

Persist option box states as key-value pairs in `NSUserDefaults`. Use the combination of the server URL and the application identifier as a key so that the user preferences for each login connection can be uniquely identified.

9.15.3 Clearing Credentials and Preferences from Mobile Devices

Authentication failure can result if the user credentials were changed since the last login, or if the mobile device is not able to reach the authentication service due to a

network or server issue. If authentication fails using the stored credentials, the SDK deletes the stored password (if present) from the keychain. The login page will then either revert to the Remember User Name Only feature, or control will revert to the mobile app using `OMMobileService` delegate. This decision is based on an SDK-level parameter named `OM_PROP_MAX_LOGIN_ATTEMPTS`, which is a numeric value that stores how many incorrect attempts for authentication are allowed before control is given back to the mobile app.

The SDK will clear a stored user password from the mobile device in the following scenarios:

- The user authentication fails (for example, if the server password is no longer valid, the user is blocked, or if the user authentication fails for another reason).
- The Mobile and Social iOS Client SDK `logout` method is called
- A session time-out is detected

The SDK will clear the stored user name, password, and option box states from the mobile device in the following scenario:

- The Mobile and Social iOS Client SDK logout method is called and the `clearRegistrationHandles` parameter is set to `TRUE`

9.15.4 Creating a Custom Login Screen

If you want to create a custom login screen instead of using the basic login view provided by the SDK, you need to subclass `OMAuthView` and add its object to the current authentication request object so that your custom view will be used for authentication instead of the default view. The Mobile and Social iOS Client SDK uses an authentication data dictionary to set user credentials and option box states. This authentication data dictionary is available as a property to every subclass of `OMAuthView`. Be sure to read these values and display them properly in the UI elements. Similarly, when submitting user credentials, be sure to read values from the UI elements and set them properly in the authentication data dictionary.

The following table lists the keys that you need to use to access credential properties in the authentication data dictionary.

Table 9–6 Keys used to access credential properties in the data dictionary

Key	Description
`OM_PROP_AUTO_LOGIN_ALLOWED`	Boolean value that enables/disables the Auto Login feature.
`OM_PROP_REMEMBER_CREDENTIALS_ALLOWED`	Boolean value that enables/disables the Remember Credentials feature
`OM_PROP_REMEMBER_USERNAME_ALLOWED`	Boolean value that enables/disables the Remember User Name Only feature
`OM_AUTO_LOGIN_PREF`	Boolean value that specifies the user's preference regarding the Auto Login feature.
`OM_REMEMBER_CREDENTIALS_PREF`	Boolean value that specifies the user's preference regarding the Remember Credentials feature.
`OM_REMEMBER_USERNAME_PREF`	Boolean value that specifies the user's preference regarding the Remember User Name Only feature.

Table 9–6 (Cont.) Keys used to access credential properties in the data dictionary

Key	Description
OM_USERNAME	String value that specifies the user's user name.
OM_PASSWORD	String value that specifies the user's password.

9.16 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 sections contain code snippets that illustrate how to use OMCredentialStore.

- Adding a User Name and Password
- Adding a User Name, Password and Tenant Name
- Deleting a Credential
- Updating a User Name and Password
- Updating a User Name, Password and Tenant Name
- Getting a User Name and Password
- Storing a Property in KeyChainItem
- Storing Multiple Properties in KeyChainItem
- Deleting a Property in KeyChainItem
- Getting a Property

9.16.1 Adding 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;
```

9.16.2 Adding 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;
```

9.16.3 Deleting 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;
```

9.16.4 Updating 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;
```

9.16.5 Updating 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;
```

9.16.6 Getting a User Name and Password

This example retrieves the user name, password, and tenant name for a given key.

```
- (OMCredential *)getCredential:(NSString*)key;
```

9.16.7 Storing a Property in KeyChainItem

This example stores a property in `KeyChainItem`.

```
- (void)storeProperty: (NSString *)property withKey: (NSString *)key;
```

9.16.8 Storing 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;
```

9.16.9 Deleting a Property in KeyChainItem

Deletes a given property store in `KeyChainItem`. All details stored with the property are deleted.

```
- (NSError *)deletePropertyWithKey:(NSString *)key
```

9.16.10 Getting a Property

Returns a property from `KeyChainItem`.

```
- (id)getPropertyForKey:(NSString *)key
```

10

Developing Mobile and Social Services Applications with the Android Client SDK

This chapter describes how to develop Mobile and Social Services applications with the Android Client SDK. This SDK serves as a security layer for developing secure mobile applications on Android devices. It essentially simplifies the development of the applications by taking control of authentication, authorization, user profile services and secure storage. The minimum Android version supported by the Mobile and Social Android Client SDK is Android 2.2. This chapter provides the following sections:

10.1 Getting Started With the Android Client SDK

This SDK (`oamms_sdk_for_android.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 Mobile Android Client SDK is provided as a library project. It contains five 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.
- **REST Handler Module** - Provides access to REST web services and automatic injection of tokens for Access Manager protected REST web services.
- **Cryptography Module** - Provides simplified APIs to perform cryptography tasks like hashing, encryption, and decryption.
- **Secure Storage Module** - Provides APIs to store and retrieve sensitive data using the preferences storage of Android.

Important: The minimum Android version supported by the Mobile and Social Android Client SDK is Android 2.2.

10.1.1 Developing and Packaging Android Applications

Review the following notes prior to developing and packaging Android applications.

Import the Library Project Into Your Workspace

The `oamms_sdk_for_android.zip` archive contains an `IDMMobileSDK` folder. The `IDMMobileSDK` folder is a library project.

To import the library project into your workspace in Eclipse, choose **File > Import > Existing projects into workspace**.

Point Your Application to the Library

In Eclipse, choose **Project > Properties > Android > Library** and add a reference to the library project.

Add Permission Statements Before Compiling

Before compiling your application, add the following `<uses-permission>` statements to the application consuming Mobile and Social services. Add the statements to the `AndroidManifest.xml` file. Your application must include these statements, otherwise it will crash.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

For more information about requesting permissions, see the Android Developer documentation:

http://developer.android.com/index.html

10.2 Invoking Authentication Services With the Android Client SDK

This section provides sample code that demonstrates how to authenticate with the Mobile and Social server. Refer to "Configuring Mobile and Social Services" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management* for information about configuring the Mobile and Social server.

Step 1: Create an Object of OMMobileSecurityService

Create an object of `OMMobileSecurityService` as follows by providing the required Mobile and Social server details.

```
OMMobileSecurityService mss = new OMMobileSecurityService(context, configProperties, callback);
```

Table 10–1 OMMobileSecurityService Parameters

Parameter	Description
`context`	The context of the `Application` object. Pass the context that is returned by `Context#getApplicationContext()`. See the following URL for more information about the `getApplicationContext()` method: http://developer.android.com/reference/android/content/Context.html#getApplicationContext() Do not pass an `Activity` reference because the `OMMobileSecurityService` instance exists throughout the application life cycle and if the `OMMobileSecurityService` instance has a reference to an `Activity`, it could lead to a memory leak.
`configProperties`	The Map that contains the configuration properties. See Table 10–5 in Section 10.4, "Initialization Properties" for the list of configuration properties.
`callback`	Specifies an instance of a class that has implemented the interface `OMMobileServiceCallback`. The SDK uses this to return control to the application after calling `OMMobileSecurityService#setup()`, `OMMobileSecurityService#authenticate()`, and `OMMobileSecurityService#logout()`.

Initialization Sample Code

```
        Map<String, Object> configProp = new HashMap<String, Object>();

          //The following strings should be available in configProperties.
          //Otherwise, IllegalArgumentException will be thrown
          configProp.put(OMMobileSecurityService.OM_PROP_AUTHSERVER_TYPE,
OMMobileSecurityService.AuthServerType.OAMMS);
          configProp.put(OMMobileSecurityService.OM_PROP_OAMMS_SERVICE_DOMAIN,
"MobileServiceDomain");
          configProp.put(OMMobileSecurityService.OM_PROP_APPNAME,  "MyApp");
          configProp.put(OMMobileSecurityService.OM_PROP_OAMMS_URL,
"http://www.example.com:14100");

        OMMobileSecurityService mss = new
```

```
OMMobileSecurityService(getApplicationContext(), configProp, callback);
        //getApplicationContext():  context of the calling application;
        //callback: Instance of a  class which has implemented the interface
        //OMMobileServiceCallback
```

- `OM_PROP_OAMMS_URL` - The URL (`<protocol>://<host>:<port>`) required to reach the Mobile and Social server. Only the HTTP and HTTPS protocols are supported.
- `OM_PROP_OAMMS_SERVICE_DOMAIN` - Specifies the name of the service domain that has been created on the Mobile and Social server. The application profile that `OM_PROP_APPNAME` refers to should be defined as an application profile in this service domain. For more information, see "Defining Service Domains" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.
- `OM_PROP_APPNAME` - 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 "Defining Application Profiles" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

Note: If your apps will authenticate against a single OAM server or a single ServiceDomain, there should be only one instance of the `OMMobileSecurityService` object throughout the application life cycle. This is required to maintain one valid session with the server at a time, and also to support the single sign-on feature.

To maintain one instance of `OMMobileSecurityService` throughout the application life cycle, either extend the Application class in the Android SDK, or create a custom static singleton class.

Refer to the Android documentation for information about extending the Application class:

http://developer.android.com/reference/android/app/Application.html

If your apps will authenticate against different OAM servers, or with the same OAM server but different authentication schemes, then one `OMMobileSecurityService` instance should be created for every server or authentication scheme. For example, if the same app authenticates against OAM server and against Facebook for different use cases, then create separate instances of `OMMobileSecurityService` to handle both use cases. On the OAM server, configure one ServiceDomain with a "Mobile Service Authentication" authentication scheme, and configure the other with a "Social Identity Authentication" authentication scheme.

Step 2: Register the Activity

Register the current activity before invoking methods in `OMMobileSecurityService`, such as `setup()`, `authenticate()`, and `processSSORequest(Intent intent)`. Deregister the activity when it is not in the foreground. Do this to ensure that there is not a memory leak. To give the reference to the current activity, register the current activity in `onResume()` and deregister the activity in `onPause()` for all activities that call the methods mentioned above. If any of these methods are called in `onCreate()` of the activity, then before calling these methods, register the activity using `OMMobileSecurityService#registerActivity(activity)`. When the activity moves to

the background, deregister the activity using the method `OMMobileSecurityService#deregisterActivity()`).

```
mss.registerActivity(this);
```

Step 3: Call the setup() Method

Call the `setup()` method, which initiates the download of your application profile from the Mobile and Social server.

```
mss.setup();
```

This is an asynchronous call. When the application profile download completes, or if a problem like network unavailability is encountered, the SDK returns control to the calling activity by using the following callback:

```
OMMobileServiceCallback#processSetupResponse(OMOMMobileSecurityService mss,
OMApplicationProfile profile, OMMobileSecurityException mse)
```

If the application profile could not be downloaded, the exception details can be found in `OMMobileSecurityException mse`. The SDK downloads and stores the application profile. The SDK will not download the application profile on subsequent calls to `OMMobileSecurityService#setup()` until the `ProfileCacheDuration` setting has expired. The `ProfileCacheDuration` setting is defined in the application profile.

Step 4: Call the authenticate() Method

Once the setup successfully completes, the business application can perform authentication by calling `OMMobileSecurityService#authenticate()`. This is an asynchronous call and the SDK will return control to the application once authentication is completed or if authentication has failed for some reason. The control will be returned to the application through `OMMobileServiceCallback#processAuthenticationResponse(OMMobileSecurityService mss, OMAuthenticationContext context, OMMobileSecurityException mse)`. All the details with respect to the authenticated session are available in the `OMAuthenticationContext` context. If authentication fails for some reason, the details are available in `OMMobileSecurityException mse`.

Set up the page as follows:

```
View view = mss.authenticate();

setContentView(view);
```

The `authenticate()` method will return a `ViewFlipper` object that holds the processing view. When the view is focused it will trigger the actual authentication flow.

If a valid authentication context is already available in the cache for the above request, then it will return control back to the calling business application without performing any request to the server.

The following sample code demonstrates the `processAuthenticationResponse()` method:

```
public void processAuthenticationResponse(OMMobileSecurityService mss,
OMAuthenticationContext context, OMMobileSecurityException mse)

{
    if (context != null)
    {
        Toast.makeText(getApplicationContext(), "Logged in successfully.",
```

```
Toast.LENGTH_LONG).show();
    }
    else
    {
        Toast.makeText(getApplicationContext(), "Failure in login due to " +
mse.getLocalizedMessage(), Toast.LENGTH_LONG).show();
    }
}
```

Note: To customize the login view or the knowledge-based authentication (KBA) view, see "Login View and KBA View Customization" later in this chapter.

Step 5: Call the logout() Method

Call the following method to logout the user:

```
mobileSecurityService.logout(boolen isForgetDevice)
```

Table 10–2 logout() Method Parameter

Parameter Name	Parameter Values	Type
isForgetDevice	**true** - Clears everything stored in the credential store, including hashed passwords for supporting offline authentication, client registration handles, and tokens. **false** - Clears only user session-related items from the store, such as tokens.	Boolean

The Mobile and Social server logout operation involves invoking a REST Web service to delete tokens maintained on the server. Consequently, the callback `OMMobileServiceCallback#processLogoutResponse(OMMobileSecurityService mss, OMMobileSecurityException mse)` is called after the server interaction is completed. Below is a sample callback implementation:

```
public void processLogoutResponse(OMMobileSecurityService mss,
OMMobileSecurityException mse)
{
            if(mse == null)
            {
                Toast.makeText(getApplicationContext(), "Logout is successful!",
Toast.LENGTH_SHORT).show();
            }
            else
            {
                Toast.makeText(getApplicationContext(), "Logout failure. " +
mse.getErrorMessage(), Toast.LENGTH_SHORT).show();
            }
}
```

10.3 URL-Based Initialization

To initialize the SDK using a URL, use a URL that contains the required configuration properties. The URL should be of the following format:

```
<scheme>://[host]?<parameter1>::=<value1>&<parameter2>::=<value2>&...&<parameterN>::=<valueN>
```

where:

- **scheme** - The scheme part of the URL. The same should be mentioned in the AndroidManifest.xml file as described in http://developer.android.com/guide/topics/manifest/data-element.html#scheme
- **parameterX, valueX** - Refer to Table 10–5 in Section 10.4, "Initialization Properties" for the list of the parameters and values that can be specified in the configuration URL

Following is a sample URL for initializing an app with Mobile and Social server details:

```
samplescheme://settings?AuthServerType::=OAMMSAuthentication&
ApplicationName::=SampleApp&
OAMMSURL::=http://www.example.com:14100&
OAMMSServiceDomain::=MobileServiceDomain
```

This URL can be sent to users by e-mail or another means and, once the user clicks the URL using their Android mobile device, the corresponding app is invoked. The intent, which contains the configuration URL, has to be passed to the following method. The method, in turn, parses the URL and extracts the SDK configuration properties:

```
Set<String> filters = new HashSet<String>();
filters.add(OM_PROP_AUTHSERVER_TYPE);
filters.add(OM_PROP_APPNAME);
filters.add(OM_PROP_OAMMS_URL);
filters.add(OM_PROP_OAMMS_SERVICE_DOMAIN);
OMMobileSecurityService.parseConfigurationURI(context, intent,
persistInSharedPreferences,keyInSharedPreference, filters)
```

Table 10–3 parseConfigurationURI() Method Parameters

Parameter Name	Description
context	The context of the calling application. Always passContext#getApplicationContext().
intent	The intent that contains the SDK configuration properties.
persistInSharedPreferences	Indicates if the SDK configuration properties have to be persisted in SharedPreferences. This has to be mentioned as true so that the constructor OMMobileSecurityService(Context context, String configurationPropertiesKey, OMMobileServiceCallback callback) can create an instance of OMMobileSecurityService based on the configuration properties stored in SharedPreferences.
keyInSharedPreference	The key against which the SDK configuration properties have to be stored.
filters	Set of property strings in the configuration URL to be returned in the Map of configuration properties and optionally persisted in SharedPreferences. The remaining parameters in the URL extracted from the configuration URL are ignored by the IDM Mobile SDK.

Once the configuration properties are stored in SharedPreferences by calling the parseConfigurationURI() method, an instance of OMMobileSecurityService can be created as follows:

```
OMMobileSecurityService mobileSecurityService = OMMobileSecurityService(Context
context, String configurationPropertiesKey, OMMobileServiceCallback callback);
```

Table 10–4 OMMobileSecurityService Parameters for URL-Based Initialization

Parameter	Description
`context`	The context returned by `Context#getApplicationContext()`
`configurationPropertiesKey`	The key against which the SDK configuration properties have been stored. This should be the same as the key that was mentioned as `keyInSharedPreference` in `OMMobileSecurityService.parseConfigurationURI(context, intent, persistInSharedPreferences,keyInSharedPreference)`.
`callback`	An instance of a class that has implemented the interface `OMMobileServiceCallback`. This is used by the SDK to return control to the application after calling `OMMobileSecurityService#setup()` and `OMMobileSecurityService#authenticate()`

10.4 Initialization Properties

Use the following initialization properties to initialize the SDK. The property names starting with `OM_PROP` are a part of the `OMMobileSecurityService` class. Use these properties to specify the entries being passed in the Map to the constructor: `OMMobileSecurityService(context, configProperties, callback)`. For URL-based initialization, the name of the parameters and their corresponding values are included in parentheses.

Table 10–5 Android Client SDK Initialization Properties

Property Name	Property Value(s)	Type

`OM_PROP_AUTHSERVER_TYPE`
`(AuthServerType)`
	■ `OMMobileSecurityService.AuthServerType.OAMMS`
	`(OAMMSAuthentication)`
	■ `OMMobileSecurityService.AuthServerType.HTTPBasicAuth`
	`(HTTPBasicAuthentication)`
		Enum
		`(OMMobileSecurityService.AuthServerType)`

`OM_PROP_OAMMS_URL` `(OAMMSURL)`	Required property. A String that contains the protocol, host, and port required to reach the Mobile and Social server. Only the HTTP and HTTPS protocols are supported. *<protocol>*`://`*<host>*`:`*<port>*	String
`OM_PROP_OAMMS_SERVICE_DOMAIN` `(OAMMSServiceDomain)`	Required property. Specifies the name of the service domain that has been created on the Mobile and Social server. The application profile that `OM_PROP_APPNAME` refers to should be defined as an application profile in this service domain. For more information, see "Defining Service Domains" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.	String

Table 10–5 (Cont.) Android Client SDK Initialization Properties

Property Name	Property Value(s)	Type
OM_PROP_APPNAME (ApplicationName)	Required property. 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 "Defining Application Profiles" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.	String
OM_PROP_MAX_LOGIN_ATTEMPTS (MaxLoginAttempts)	The maximum number of login attempts allowed by the user. Optional. If this property is not specified, the default is 3.	Integer
OM_PROP_AUTH_KEY (AuthKey)	The key that must be used to store the OMCredential object in shared preferences. This is an optional property. If it is not specified, it is stored using the <*login_url*>_<*applicationName*>_<*serviceDomain*> key if all the values are not null. Otherwise, it is stored using <*applicationName*> alone.	String
OM_PROP_LOCATION_UPDATE_ENABLED (LocationUpdateEnabled)	Use this property to enable or disable location updates. If it is enabled, then the SDK will collect location details of the device and send it to the server.	Boolean
OM_PROP_AUTO_LOGIN_ALLOWED (AutoLoginAllowed)	Optional. Specifies if auto-login is enabled. Either **true** or **false**. By default it is disabled.	Boolean
OM_PROP_REMEMBER_USERNAME_ALLOWED (Remember User name Allowed)	Specifies if the Remember User Name feature is enabled. Either **true** or **false**. By default it is disabled.	Boolean
OM_PROP_REMEMBER_CREDENTIALS_ALLOWED (RememberCredentialsAllowed)	Specifies if the Remember Credentials feature is enabled. Either **true** or **false**. By default it is disabled.	Boolean
OM_AUTO_LOGIN_DEFAULT (AutoLoginDefault)	Represents the default preference for the auto-login option box. Either **true** or **false**. Default value is false.	Boolean
OM_REMEMBER_USERNAME_DEFAULT (RememberUsernameDefault)	Represents the default preference for the Remember User Name option box. Either **true** or **false**. The default value is **false**.	Boolean
OM_REMEMBER_CREDENTIALS_DEFAULT (RememberCredentialDefault)	Represents the default preference for the remember credentials option box. Either **true** or **false**. Default value is false.	Boolean

The following OM_PROP_CRYPTO_SCHEME cryptography property is an optional property. If the application requires offline authentication and this property is not specified, the default crypto scheme is SSHA512. This is used to hash / encrypt the password and store it for offline authentication later, if required by the application.

This property can also be set using the Mobile and Social server console. Choose **Custom Settings** > **Mobile Custom Attributes** and use the Attribute Name and Attribute Value listed in the last two columns in the table.

For information about the cryptography module included in the Mobile and Social Android Client SDK, see Section 10.14, "Using the Cryptography APIs."

Table 10–6 Cryptography Scheme Property Attributes for the Android Client SDK

Property Name	Property enum Values	Attribute Name	Attribute Value
OM_PROP_CRYPTO_SCHEME (CryptoScheme)	PLAINTEXT("PlainText")	Enum(CryptoScheme)	PlainText
	AES("AES")		AES
	SHA1("SHA-1")		SHA-1
	SHA224("SHA-224")		SHA-224
	SHA256("SHA-256")		SHA-256
	SHA384("SHA-384")		SHA-384
	SHA512("SHA-512")		SHA-512
	SSHA1("SaltedSHA-1")		SaltedSHA-1
	SSHA224("SaltedSHA-224")		SaltedSHA-224
	SSHA256("SaltedSHA-256")		SaltedSHA-256
	SSHA384("SaltedSHA-384")		SaltedSHA-384
	SSHA512("SaltedSHA-512")		SaltedSHA-512

10.5 About Offline Authentication

Mobile and Social supports offline authentication. Use the enum `OMConnectivityMode` to specify how offline authentication should happen.

- **`OMConnectivityMode.ONLINE`** - Always authenticate with the server. Fails if device cannot reach the Internet.
- **`OMConnectivityMode.OFFLINE`** - Authenticates locally with cached credentials. Offline authentication happens even if the device is online and can reach the server.
- **`OMConnectivityMode.AUTO`** - Authentication happens with the server if the server is reachable and happens offline if the device is not connected to the Internet.

An example is shown here:

```
OMAuthenticationRequest authRequest = new OMAuthenticationRequest();
authRequest.setConnectivityMode(OMConnectivityMode.ONLINE);
View authView = authenticate(authRequest);
```

Because offline authentication is part of `OMAuthenticationRequest`, the setting is valid for the current request alone. During offline authentication if the number of failure attempts exceeds the value of the `OM_PROP_MAX_LOGIN_ATTEMPTS` property (discussed in the "Initialization Properties" section), the locally stored credentials are deleted and online authentication is attempted.

> **Note:** Offline authentication only works if the "Offline Authentication" setting in the server is set to "Allowed." See "Editing or Deleting an Application Profile" in the *Administrator's Guide for Oracle Access Management* for more information.

To support offline authentication, the SDK captures and stores the user credentials. The password of the user will either be hashed or encrypted based on the value set for the property `OM_PROP_CRYPTO_SCHEME`. If the application requires offline authentication and this property is not specified, the default crypto scheme is SSHA512.

10.6 Invoking Social Identity Authentication Using the Android Client SDK

This section provides sample code that shows how to do Social Identity Authentication (also called *Relying Party Authentication*) with the Mobile and Social Server. Refer to the "Configuring Social Identity" chapter in the *Administrator's Guide for Oracle Access Management* for information about configuring the Mobile and Social server.

To authenticate using a Relying Party, the data scheme in the main activity of the application should match the URL scheme configured in the Application profile. See the following page for details.

`http://developer.android.com/guide/topics/manifest/data-element.html`

Call `OMMobileSecurityService#authenticate()` as usual to authenticate. The SDK will retrieve the necessary settings from the server and authenticate with the Relying Party. Once authenticated, retrieve the authentication context from the mobile security service instance (`OMMobileSecurityService#retrieveAuthenticationContext()`). The retrieved authentication context object also contains Identity Provider specific information. The authentication context allows you to get the logged in user name, the identity provider name, and the access token of the identity provider. Please refer to the SDK documentation for more details.

> **Note:** Include the following code snippet in the main activity of an app that authenticates against Social Identity Providers like Facebook, Google, and so on. The main activity is the activity that is defined as the **Android Package** in the Application Profile settings on the Mobile and Social Server.
>
> The Android Package field should be set to the fully qualified name of the activity in the Android application. The activity should have `<data android:scheme="xyz" />` in its `<intent-filter>` and `xyz` should be the same as the URL scheme.

```
@Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        ...
        Intent intent = getIntent();
        if (intent.getExtras()!= null)
        {
```

```
            try
            {
                mMobileSecurityService.processSSORequest(intent);
            }
            catch (OMMobileSecurityException e)
            {
                Log.w(TAG, e.getLocalizedMessage());
            }
        }
        ...
    }
```

10.7 Invoking the Mobile Single Sign-on Agent App

This section describes how to use the Android Client SDK to interact with a mobile single sign-on agent app. For conceptual information about mobile single sign-on in Mobile and Social, see the "Understanding Mobile Single Sign-on (SSO) Capabilities" and "Understanding Mobile and Social" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

In order to create your own Mobile SSO Agent App, refer to Section 10.12, "Creating a Custom Mobile Single Sign-on Agent App Using the Android Client SDK."

10.7.1 Invoking the Mobile Single Sign-on Agent App from Another Application (SSO Client)

When applications that are configured as SSO clients make a call to `OMMobileSecurityService#authenticate()`, the Android Client SDK delegates the request to the SSO Agent application if it is already installed on the device. Otherwise it throws an exception. The Android Client SDK frames the authentication request and invokes the SSO Agent app using an intent.

Note: If an application (SSO client) needs to authenticate using the SSO Agent App, the main activity has to call `OMMobileSecurityService#processSSORequest(intent)`. This is required because after the SSO Agent app authenticates, it sends details to the main activity of the SSO Client App using some intent. The SDK will appropriately populate its data structures in the SSO Client App with the details present in the intent.

The main activity is the activity configured on the Mobile and Social server as the "Android Package" in the Application Profile.

Refer to the following sample code:

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    …
    …
    Intent intent = getIntent();
    if (intent.getExtras()!= null)
    {
        try
        {
            mMobileSecurityService.processSSORequest(intent);
```

```
            }
            catch (OMMobileSecurityException e)
            {
                Log.w(TAG, e.getLocalizedMessage());
            }
        }
        …
        …
    }
```

10.7.2 Invoking the Mobile Single Sign-on Agent App Using a Mobile 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.

   The Launch Pad opens.

2. Under **Access Manager** click **Authentication Schemes**, then click the **Create Authentication Scheme** 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 `/protectedRes`, and protect the resource using the created Authentication Scheme (MobileSSOScheme). This is the URL that will be accessed from the mobile web browser and protected by a WebGate.

When the protected resource is accessed using a mobile browser, the server will invoke the Mobile SSO Agent App. Because the agent app makes a call to `OMMobileSecurityService#processSSORequest(intent)` in the `onResume()` of the main activity, the agent app will delegate the control to the SDK to handle complexities like authenticating, obtaining the `OAM_ID` token, injecting it, and returning control back to the browser. The browser will now be able to access the protected resource as the required token has been set by the SSO Agent App.

10.8 Invoking User Profile Services With the Android Client SDK User Role Module

This section describes how to use the Android Client SDK to create, read, update, or delete users and groups in a directory server that has been configured to work with Mobile and Social. To configure the Mobile and Social server to work with a directory server, see "Defining, Modifying, or Deleting a User Profile Service Provider" and "Defining, Modifying, and Deleting a User Profile Service Profile" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

You can configure the User Role module to be used anonymously or only after authentication by adjusting the settings on the **Service Protection** tab in the Service Domain.

In both cases, you first need to obtain a reference to an instance of `OMUserProfileService` from the `OMMobileSecurityService` instance: `OMMobileSecurityService#getUserRoleProfileService(OMAuthenticationContext)`.

The second step is to obtain either an instance of `UserManager` or `RoleManager`, depending on the operations to be performed. An instance of `UserManager` can be used to get User Details, whereas an instance of `RoleManager` can be used to get Group details from the Identity Store associated with your application's domain on the Mobile and Social server.

1. Obtain an instance of `OMUserProfileService` by passing in the `AuthenticationContext` obtained after successful authentication. You can obtain the authentication context after authentication using `OMMobileSecurityService#retrieveAuthenticationContext()`.

   In the following sample, `mMobileSecurityService` is an instance of `OMMobileSecurityService`.

   ```
   OMUserRoleProfileService userProfileService =
   mMobileSecurityService.getUserRoleProfileService(authContext);
   ```

   If anonymous access to these services is needed, pass `null`.

   ```
   OMUserRoleProfileService userProfileService =
   mMobileSecurityService.getUserRoleProfileService(null);
   ```

2. Obtain either an instance of `UserManager` or `RoleManger` as shown.

   ```
   OMUserManager userManager = userProfileService.getUserManager();

   OMRoleManager roleManager = userProfileService.getRoleManager();
   ```

3. The following methods can be invoked using `userManager`. All these methods interact with the Mobile and Social server over a network, so these methods should be called from a background thread rather than a UI thread.

   ```
   public List<OMUser> searchUsers(Map<String, String> searchFilter, List<String>
   attrsToFetch, int startIndex, int size, boolean isSimpleSearch) throws
   OMMobileSecurityException

   public OMUser searchUser(String userId, List<String> attrsToFetch, boolean
   shouldPrefetchManager) throws IOException, JSONException,
   OMMobileSecurityException

   public OMUser modifyUser(String uid, String telephoneNumber, String mobile,
   String mail, String address) throws OMMobileSecurityException
   ```

```
public boolean deleteUser(String userId) throws OMMobileSecurityException


public boolean createUser(Map<String, String> attrVals) throws
OMMobileSecurityException
```

The following methods can be invoked using `roleManager`. As is the case with methods in `OMUserManager`, all the methods in `OMRoleManager` should also be called from a background thread.

```
public OMRole getRole(String roleName) throws IOException,
JSONException,OMMobileSecurityException

public boolean createRole(Map<String, String> attrVals) throws
OMMobileSecurityException

public OMRole modifyRole(String rolename, Map<String, String> attrVals) throws
OMMobileSecurityException

public boolean deleteRole(String roleName) throws OMMobileSecurityException

public OMUser getUserFromRole(String userId, String role) throws
OMMobileSecurityException

public void addUserToRole(String roleName, Map<String, String> attrVals) throws
OMMobileSecurityException

public boolean deleteUserFromRole(String roleName, String userId) throws
OMMobileSecurityException
```

10.9 Authenticating Using Client Certificate

IDM Mobile SDK supports 2-way SSL mutual authentication. In addition to the client validating the identity of the server, if the authentication server is configured to perform 2-way SSL, the server will then require the client certificate to validate the identity of the client. In other words, this is an additional authentication scheme where the server identifies the client, to avoid anonymous access.

IDM Mobile SDK provides various APIs for the client application to import the client certificates in the application keystore.

The client application can use this feature as a standalone authentication scheme or can club this feature with any other authentication schemes that are supported by the IDM Mobile SDK.

For this feature, IDM Mobile SDK introduces a new `AuthServerType.CBA` for applications that want to perform only standalone client certificate-based authentication.

Sample code:

```
configMap.put(OMMobileSecurityService.OM_PROP_APPNAME, "CBAApp");
configMap.put(OMMobileSecurityService.OM_PROP_AUTHSERVER_TYPE,
AuthServerType.CBA);
configMap.put(OMMobileSecurityService.OM_PROP_LOGIN_URL,
"https://myserver.com:1234/cba/");
```

Client certificate authentication can also be clubbed with other authentication modules such as Mobile and Social Authentication. The application is only required to pass the `OM_PROP_PRESENT_CLIENT_IDENTITY_ON_DEMAND` property as `true` during initialization

process and the SDK present the client identity when it is requested by the authentication server. If the client application does not provide this property, then the IDM Mobile SDK will not provide support for client certificate-based authentication with other authentication schemes. This is described in the following sample code:

```
configProp.put(OMMobileSecurityService.OM_PROP_APPNAME,
"OAuthTesterApp");
configProp.put(OMMobileSecurityService.OM_PROP_AUTHSERVER_TYPE,
OAUTH20);
configProp.put(OMMobileSecurityService.OM_PROP_BROWSER_MODE,
BrowserMode.EMBEDDED);
configProp
.put(OMMobileSecurityService.OM_PROP_LOGIN_FAILURE_URL,
.put(OMMobileSecurityService.OM_PROP_LOGIN_URL,"http://login.com");\\\\
configProp.put(OMMobileSecurityService.OM_PROP_LOGIN_SUCCESS_URL,
"http://loginsucess.coml");\\\\ configProp.put(OMMobileSecurityService.OM_PROP_
LOGOUT_URL,
"http://logout.com");\\\\ configProp.put(OMMobileSecurityService.OM_PROP_PRESENT_
CLIENT_IDENTITY_ON_DEMAND, true);
```

Note: If an app wants to import or preload some server or client certificates using the IDM Mobile SDK's import certificate functionality, it must be done before making any network connections with the SDK setup, authentication, or resource access.

OPEN ISSUES: Prior to Android L android webview has inability to respond to the 2-way SSL challenges during the network request, hence SDK does not support client certificate authentication in webview on devices running below android L. The same is being tracked as issues(https://code.google.com/p/android/issues/detail?id=53491) in Android Open source project.

10.9.1 Importing Certificates

This section describes the ability of the IDM Mobile SDK to import or preload the certificates before initializing any of the authentication processes.

10.9.1.1 Importing Server Certificates

The IDM Mobile SDK provides the `OMCertService` class that can be initialized by passing the application context in the constructor. This class supports importing both server and client certificates, along with this it also support various certificate operations as described in the following sections.

If the application imports the server certificate or the root certificate before starting the authentication process, then no "untrusted server certificate" warning will be shown by the SDK during the authentication process.

10.9.1.1.1 Importing the X509 Certificate

The IDM Mobile SDK provides the ability to import the server certificate as an "X509 Certificate" Object. Assuming that the application has bundled the certificate and now needs to import the certificate from the application resources during the initialization process of the application. The following sample code snippets describe this process:

```
OMCertService certService = new OMCertService(this.getApplicationContext());
X509Certificate cert;
            try
```

```
            {
                InputStream is = getAssets().open("Cert.cer");
                cert = readCertificateFromStream(is);
                certService.importServerCertificate(cert);
            }
            catch (CertificateException e2)
            {
                // TODO Auto-generated catch block
                e2.printStackTrace();
            }
            catch (IOException e2)
            {
                // TODO Auto-generated catch block
                e2.printStackTrace();
            }

X509Certificate readCertificateFromStream(final InputStream is)
            throws CertificateException
    {
        CertificateFactory certFactory = CertificateFactory
                .getInstance("X.509");
        DataInputStream dis = null;
        dis = new DataInputStream(is);
        try
        {
            byte[] certBytes = new byte[dis.available()];
            dis.readFully(certBytes);
            Certificate cert = certFactory
                    .generateCertificate(new ByteArrayInputStream(certBytes));
            if (cert instanceof X509Certificate)
                return (X509Certificate) cert;
            else
                throw new CertificateException();
        }
        catch (Exception e)
        {
            throw new CertificateException(e);
        }
        finally
        {
            if (dis != null)
            {
                try
                {
                    dis.close();
                }
                catch (IOException e)
                {
                    // do nothing
                }
            }
        }
```

10.9.1.1.2 Importing the Certificate File

The IDM Mobile SDK also supports the ability to import certificates from a "file" object within the device, such as mnt, sdcard or any other external storage location, or in app storage locations such as `/data/data/myapp/certFile.cer`. The following sample describe this feature:

```
OMCertService certService = new OMCertService(this.getApplicationContext());
```

```
//assuming that the certificate file is "Cert.cer" and is present in /mnt/sdcard
File file = new File(Environment.getExternalStorageDirectory(),"Cert.cer");
        try
        {
            certService.importServerCertificate(file);
        }

        catch (CertificateException e1)

         {

           // import cert failed, check for the cause.
         }
```

10.9.1.2 Importing Client Identity Certificates

Any application that uses IDM Mobile SDK can import a user or client identity certificate. The `OMCertService` class provides the ability for both UI and non-UI APIs to import the client certificates.

The two following APIs are available in the `OMCertService` class to import client certificates:

10.9.1.2.1 Importing Client Certificates With a Password

`public void importClientCertificate(File file, char[] pwd)`

This API imports the given client certificate file in the Keystore that is maintained locally by the application. To do so, the application must pass the "File" object and the password for the .p12 certificate file. The following sample code describes the usage of this API:

```
OMCertService certService = new OMCertService(getApplicationContext());
File file = new File(Environment.getExternalStorageDirectory() + "/"
+ "client.p12");
try
{
certService.importClientCertificate(file, "password".toCharArray());
}
catch (CertificateException e)
{
...
}
```

10.9.1.2.2 Importing Client Certificates Without a Password

`public void importClientCertificate(Activity activity, File file, OMCertServiceCallback callback)`

This API is also used to import the client certificate, but it does not accept a password. Instead, it alerts to the user to collect the password.

> **Note:** This API must always be called from the UI thread.

The following sample code describes the usage of this API:

```
certService.importClientCertificate(this,
new File(Environment.getExternalStorageDirectory() + "/"
+ "newuser.p12"), new OMCertServiceCallback()
```

```
{

@Override
public void processClientCertifcateImportResponse(CertificateInfo cert,
OMMobileSecurityException mse)
{
if (cert != null)
{
... .... .. ..
}

}
});
```

10.9.2 Performing Operations on Imported Certificates

The `OMCertService` class provides various APIs to perform operations on the imported certificates. The IDM Mobile SDK allows you to perform the following operations on imported Server and Client certificates:

- List all imported certificates.
- Get the "Certificate Info" of a particular certificate.
- Delete the imported certificates.

10.9.2.1 Server Certificates

The `OMCertService` class provides the following APIs to perform various operations on imported server certificates:

- `public List<OMCertInfo> getAllServerCertificateInfo()`

  This API returns the list of all the server certificates installed in the SDK truststore. These certificates are represented by OMCertInfo.

- `public void deleteAllServerCertificates()`

  This API deletes all the imported server certificates in the SDK truststore.

- `public void deleteServerCertificate(OMCertInfo)`

  This API deletes the specified server certificate represented by the passed `OMCertInfo` object.

10.9.2.2 Client Certificates

The `OMCertService` class provides the following APIs to perform various operations on imported client certificates:

- `public List<OMCertInfo> getAllClientCertificateInfo ()`

  This API returns the list of all the client certificates installed in the SDK KeyStore. These certificates are represented by OMCertInfo.

- `public void deleteAllClientCertificates()`

  This API deletes all the imported client certificates in the SDK truststore.

- `public void deleteClientCertificate(OMCertInfo)`

  This API deletes the specified client certificate represented by the passed `OMCertInfo` object.

10.10 Developing OAuth and Mobile OAuth Services Applications With the Android Client SDK

This section describes the capabilities of the OAuth Service. It includes the following topics:

10.10.1 Understanding OAuth2.0 for Android

The OAuth Service allows organizations to implement the open standard OAuth 2.0 Web authorization protocol in an Access Manager environment. OAuth enables a client to access Access Manager protected resources that belong to another user (that is, the resource owner). The OAuth Service allows organizations to implement OAuth 2.0 in a new or existing Access Manager environment so that OAuth clients can use OAuth 2.0 flows to access resources protected by Access Manager. An OAuth client can be an application or service created and controlled by your organization, or it can be an application or service created and controlled by another organization that requires access to resources protected by Access Manager.

> **Tip:** For detailed information about OAuth and its related concepts see the "Understanding the OAuth Service" chapter of the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

The IDM Mobile SDK provides authorization against the OAM Mobile and Social (M&S) OAuth Server in order to access protected resources.

After the initialization of the IDM Mobile SDK with the correct properties and authentication, the application will be able to get the access tokens in the same method as it did under other authentication modes.

Table 10–7 lists the new configuration properties that have been added to support the OAuth 2.0 service:

Table 10–7 Configuration Properties for OAuth 2.0

Configuration Property	Valid Value	Mandatory
OM_PROP_AUTHSERVER_TYPE (AuthServerType)	Enum (oracle.idm.mobile.OMMobileSecurityService.AuthServerType) AuthServerType.OAuth20 (OAuth20)	Yes

Table 10–7 (Cont.) Configuration Properties for OAuth 2.0

Configuration Property	Valid Value	Mandatory
OM_PROP_OAUTH_ AUTHORIZATION_GRANT (OAuthAuthZGrantType)	Enum(oracle.idm.OMMobileSecurityConfiguration.OAuthAuthorizationGrantType), the value can be one of the following: ■ IMPLICIT(OAuthImplicit) ■ AUTHORIZATION_ CODE(OAuthAuthorizationCode) ■ RESOURCE_ OWNER(OAuthResourceOwner) ■ ASSERTION("OAuthUserAssertion") - (OAM(M&S) oauth) ■ CLIENT_ CREDENTIALS("OAuthClientCredentials") - (OAM(M&S) oauth) ■ OAM_ CREDENTIAL("OAuthOAMCredential") - (OAM(M&S) oauth)	Yes
OM_PROP_OAUTH_ AUTHORIZATION_ ENDPOINT (OAuthAuthZEndpoint)	URL/String	This is mandatory for all client types using AUTHORIZATION_ CODE as grant type, except M&S Mobile OAuth Clients.
OM_PROP_OAUTH_TOKEN_ ENDPOINT (OAuthTokenEndpoint)	URL/String	This is mandatory for all client types except M&S Mobile OAuth Clients.
OM_PROP_OAUTH_ REDIRECT_ENDPOINT (OAuthRedirectEndpoint)	String	This is mandatory for all client types except M&S Mobile OAuth Clients.
OM_PROP_OAUTH_CLIENT_ ID (OAuthClientID)	String	Yes
OM_PROP_OAUTH_SCOPE (OAuthScope(s))	Set<String>	No

Table 10–7 (Cont.) Configuration Properties for OAuth 2.0

Configuration Property	Valid Value	Mandatory
OM_PROP_BROWSER_MODE (BrowserMode)	Enum (oracle.idm.mobile.OMMobileSecurityConfiguration.BrowserMode)	No. However, the default option will be EXTERNAL if nothing is set during initialization.
OM_PROP_OAUTH_CLIENT_SECRET (OAuthClientSecret)	String	No
OM_PROP_OAM_OAUTH_SERVICE_ENDPOINT (OAMOAuthServiceEndpoint)	String/URL	No. This is only required for OAM OAuth mobile clients.
OM_PROP_OAUTH_ASSERTION_JWT (OAuthUserJWTAssertionValue)	String	No
OM_PROP_OAUTH_ASSERTION_SAML2 (OAuthUserSAML2UserAssertionValue)	String	No
OM_PROP_OAUTH_CLIENT_ASSERTION_SAML2 (OAuthClientSAML2AssertionValue)	String	No
OM_PROP_OAUTH_CLIENT_ASSERTION_JWT (OAuthClientJWTAssertionValue)	String	No

10.10.2 Oracle Access Manager Mobile and Social (M&S) OAuth

The OAM M&S server supports the following types of OAuth clients that can be used with mobile.

- **Web Clients:** These clients can use the IDM Mobile SDK with the OAuth Generic flows as discussed below. The web-based clients are considered as trusted clients and possess a client secret. Therefore, the authorization server knows the identity of the client that is making the request. When using this client type, the application should pass the client secret during SDK initialization.
- **Mobile Clients :** Mobile clients are not considered to be confidential clients. So, they do not possess a client secret. The OAM OAuth server provides a functionality through which mobile clients can be registered dynamically by sending the device claims along with the username and password of the end user. This flow is proprietary to the OAM OAuth server. After dynamic client registration is performed, the IDM Mobile SDK receives a client assertion and user assertion (if SERVER SIDE SSO is not enabled). The client assertion token is required to be sent in all the subsequent requests for access token acquisition or

other token operations. IDM Mobile SDK manages the life cycle of the client assertion and does not return this to the client application.

New Token Type	Significance
Client Assertion	This token is generated after the dynamic client registration is performed. This token is must be sent with every request made to the authorization server. The server identifies the request or client from the client assertion and validates the same. Client assertions are usually JWT tokens.
User Assertion	Since dynamic client registration involves user authentication, M&S along with client assertion generates a user assertion. This user assertion marks the user session on the server side. The user assertion is returned by the server only when the SERVER SIDE SSO is disabled. The client or SDK can thus use the same assertion to acquire the access token for every resource access till the user session is valid on server side. Also, when the SERVER SIDE SSO is enabled, the SDK or client can still use the user assertion by adding the following parameter in every request: `use_server_side_device_store = true`

After configuring the server for the OAuth mobile client, the application can initialize the SDK using the same details. See the "Configuring OAuth Services" chapter of Fusion Middleware Administrator's Guide for Oracle Access Management for detailed information about configuring the server. See the sample code in Section 10.10.2.1.2, "Working With the Supported Grant Types and Getting Access tokens" for more information about initializing the SDK.

10.10.2.1 Authentication

After the completing the setup, the application can perform authentication which is to get an access token for accessing the OAuth protected resources. The authentication flow is not different from the other flows. The application only needs to invoke the OMMobileSecurityService#authenticate() API to start the authentication process. The SDK will invoke the OMMobileServiceCallback#processAuthenticationResponse after the authentication is done. If the authentication is successful, the context will contain the access token along with the other axillary tokens like user_assertion it if available. If the authentication was not successful, the authentication context will be null.

This flow of authentication involves the following steps:

- Section 10.10.2.1.1, "Dynamic Client Registration"
- Section 10.10.2.1.2, "Working With the Supported Grant Types and Getting Access tokens"

10.10.2.1.1 Dynamic Client Registration

Mobile clients are usually not considered to be confidential and so the server does not create any secrets for these clients. However, the OAM M&S server provides a functionality through which the client can register itself dynamically to get a client token (client_assertion) which is tied to the user and the device from which the registration is being performed. The device information is useful for the server to track the user and the device so that it can trigger the OAAM plug-in if applicable.

The OAM Mobile and Social server supports two methods to perform client registration. The client registration flow is decided by the SDK based on the grant type

provided by the application during initialization. The property used to define the grant type is OM_PROP_OAUTH_AUTHORIZATION_GRANT.

2-Legged Client Registration:

2-legged client registration is performed when the grant type is set as `RESOURCE_OWNER`, `ASSERTION`, `CLIENT_CREDENTIALS`, or `OAM_CREDENTIALS`. After the end of this flow, SDK receives the client_assertion along with user assertion token.

> **Note:** The availability of user_assertion on the client side is dependant on the value that is set for the SERVER_SIDE_SSO property in the server.

3-Legged Client Registration:

3-legged client registration is done when the grant type is set as `AUTHORIZATION_CODE`. This flow will invoke the external or embedded browser based on the initialization property. At the conclusion of this flow, IDM Mobile SDK will receive only the client_assertion token.

10.10.2.1.2 Working With the Supported Grant Types and Getting Access tokens

After the dynamic client registration is performed and the SDK has a valid client_assertion token as mentioned in Section 10.10.2.1.1, "Dynamic Client Registration," it can proceed to perform the access token acquisition flow. The access token can be acquired using the following grant type that is specified using the initialization property `OM_PROP_OAUTH_AUTHORIZATION_GRANT`. The following is the sample code for initialization:

```
Map<String, Object> configMap = new HashMap<String, Object>();
        configMap.put(OMMobileSecurityService.OM_PROP_APPNAME,
                "MSOAuthClient1");
        configMap.put(OMMobileSecurityService.OM_PROP_AUTHSERVER_TYPE,
                AuthServerType.OAuth20);
        configMap.put(OMMobileSecurityService.OM_PROP_OAUTH_CLIENT_ID,
                "1234567890987654321");

        configMap.put(
                    OMMobileSecurityService.OM_PROP_OAUTH_REDIRECT_ENDPOINT,
                    "idmtester://");
        configMap.put(OMMobileSecurityService.OM_PROP_OAUTH_AUTHORIZATION_GRANT_
TYPE,
OAuthAuthorizationGrantType.RESOURCE_OWNER);

        Set<String> scopeset = new HashSet<String>();

        scopeset.add("UserProfile.users");
        scopeset.add("UserProfile.secretkey.management");

        configMap.put(OMMobileSecurityService.OM_PROP_OAUTH_SCOPE, scopeset);

        configMap.put(
                    OMMobileSecurityService.OM_PROP_OAM_OAUTH_SERVICE_ENDPOINT,
                    "http://myhost.example.com:18465/ms_
oauth/oauth2/endpoints/oauthservice");
        configMap.put(OMMobileSecurityService.OM_PROP_BROWSER_MODE,
BrowserMode.EXTERNAL);
```

Resource Owner Grant Type

The value of the initialization property should be `OAuthAuthorizationGrantType.RESOURCE_OWNER`. For this grant type, the 2-legged client registration flow is performed (as mentioned above). The SDK collects the user name and password in order to complete the 2-legged client registration flow. Hence the SDK will reuse these user credentials to get the access token. The SDK does not honor the SERVER_SIDE_SSO property sent by the server in this flow and will always send the user credentials to get an access token.

If the client assertion is expired or invalidated, the SDK will again ask for the end user credentials to first renew the client assertion and then get the access token.

Client Credentials Grant Type

The value of the initialization property should be `OAuthAuthorizationGrantType.CLIENT_CREDENTIALS`. This grant type also involves the 2-legged client registration. In order to complete the authentication flow, the SDK uses the client_assertion obtained after the client registration to get the access token.

> **Note:** If the client assertion is expired or invalidated, the SDK will ask for the end user credentials to first renew the client assertion and then get the access token.

Assertion Grant Type

The value of the initialization property should be `OAuthAuthorizationGrantType.ASSERTION`.

This grant type also involves the 2-legged client registration flows, after which the SDK gets a client_assertion token and a user_assertion token. As mentioned earlier, the user_assertion token is made available by server only if the SERVER_SIDE_SSO is false. To complete the authentication flow the SDK uses the user assertion obtained after client registration to get the access token. This grant type honors the SERVER_SIDE_SSO configuration property from the server by appropriately forming the request based on the property value. If the SERVER_SIDE_SSO property is set to `true`, the SDK adds the `oracle_use_server_device_store=true` parameter in the access token request. In doing so, it requires the server to reuse the user session that was created at the server side.

Applications can however provide the custom assertion to be used for this flow using the initialization property `OM_PROP_OAUTH_ASSERTION_JWT` for JWT assertion and `OM_PROP_OAUTH_ASSERTION_SAML2` for saml2 assertions. The SDK will use this assertion for access token acquisition to complete the authentication flow.

Authorization Code Grant Type

The value of the initialization property should be `OAuthAuthorizationGrantType.AUTHORIZATION_CODE`. This grant type involves the 3-legged client registration flow. After this flow, the SDK receives only the client_assertion. During this flow, the SDK invokes either the external or embedded browser based on the initialization property `OM_PROP_BROWSER_MODE`. The SDK must invoke the browser twice to complete the authentication flow. The first time is for client registration and the second time is for getting the access token.

Note: If the application provides the value as `BrowserMode.EXTERNAL` for the OM_PROP_BROWSER_MODE property, the user may not have to enter the credentials twice in the external browser. This is because the server will set the OAM_ID cookie in the external browser during client registration flow itself, and so, during the access token acquisition the browser will not prompt for the login credentials.

OAM Credential Grant Type:

The value of the initialization property should be `OAuthAuthorizationGrantType.OAM_CREDENTIAL`. This grant also involves the 2-legged client registration flows after which the SDK will obtain a client_assertion and a user_assertion token based on the value set for the SERVER_SIDE_SSO property. To complete the authentication flows, SDK will first exchange the client_assertion to get the access token for OAuth protected resources, and then use the client_assertion and user_assertion (if available) to get the OAM_ID(USERTOKEN_MT) and OBSSOCookie(USERTOKEN) tokens. This grant type is provided by the server for the clients who use the OAM_ID cookie to access the webgate protected resources.

10.10.3 Standard Flows(Generic Implementation)

IDMMobile SDK supports authorization against any OAuth2.0 compliant server .The support is added in order to fit with current authentication architecture followed by the IDMMobile SDK for other types of authentications . The SDK will also work against any OAuth2.0 generic server provided it supports the below mentioned grant types for mobile clients.

The Current implementation supports the following grant types:

- Implicit grant type.(client id and token endpoint are must)
- Authorization code grant type .(client_id,authorization end point and token endpoint are must)
- Resource Owner grant type (client_id and token end point is must)
- Client Credentials (for this client_id and client_secret is must)
- Assertion (For this the application should provide an assertion value using any one of the OM_PROP_OAUTH_ASSERTION_JWT or OM_PROP_OAUTH_ ASSERTION_SAML2 property based on the type of assertion)

10.10.3.1 Authentication

In order to authenticate the user (or get a valid access token) with the OAuth2.0 server, the SDK accepts a set of scopes.

After user authentication and user consent with the authorization server, the access token returned from the authorization server is retrieved by the IDMMobile SDK and is tied with the scopes provided during initialization.

Hence, before starting any authentication process, the SDK checks for any valid access token for the requested scopes. If there is a valid access token for the requested scopes or there is an access token already present with a wider scope or scopes, the SDK will not perform authentication again. Instead, it will return the same authentication context (access token) .

The IDMMobile SDK supports the refreshing of the access token if it is supported by the authorization server.

The life cycle of an access token is controlled by the SDK. So if any access token is invalid, the SDK will internally try to refresh the access token if the access token had a refresh token value returned from the server initially.

Sample code for initialization :

```
            configProp.put(OMMobileSecurityService.OM_PROP_AUTHSERVER_TYPE,
                    OMMobileSecurityService.AuthServerType.OAuth20);
            configProp.put(OMMobileSecurityService.OM_PROP_OAUTH_CLIENT_ID,
                    "myhost.example.com");
            configProp
                    .put(OMMobileSecurityService.OM_PROP_OAUTH_AUTHORIZATION_
GRANT,
                            OAuthAuthorizationGrantType.AUTHORIZATION_CODE);
            configProp
                    .put(OMMobileSecurityService.OM_PROP_OAUTH_AUTHORIZATION_
ENDPOINT,
                            "https://accounts.example.com/o/oauth2/auth");
            configProp.put(
                    OMMobileSecurityService.OM_PROP_OAUTH_TOKEN_ENDPOINT,
                    "https://accounts.example.com/o/oauth2/token");
            configProp.put(OMMobileSecurityService.OM_PROP_BROWSER_MODE,
                    BrowserMode.EMBEDDED);
            configProp.put(OMMobileSecurityService.OM_PROP_OAUTH_REDIRECT_
ENDPOINT,
                    "http://localhost");

           List<String> scope1 = new ArrayList<String>();
           scope1.add("https://www.example.com/auth/userinfo.email");
           scope1.add("https://www.example.com/auth/userinfo.profile");

           configProp.put(OMMobileSecurityService.OM_PROP_OAUTH_SCOPE, scope1);
           configProp.put(OMMobileSecurityService.OM_PROP_APPNAME, appId);
```

After initializing, you can call the authenticate() api on MSS after performing the setup.

The following is a sample code to authenticate with a custom OMAuthenticationRequest:

```
 OMAuthenticationRequest authRequest1 = new OMAuthenticationRequest();
           List<String> scope3 = new ArrayList<String>();
           scope3.add("https://www.example.com/auth/calendar");
           authRequest1.setOAuthScopes(scope3));
               try
               {
                   mAuthView = getMobileSecuritySevice().authenticate(
                           authRequest1);
               }
               catch (OMMobileSecurityException e)
               {
                   e.printStackTrace();
               }
           RelativeLayout.LayoutParams params1 = new RelativeLayout.LayoutParams(
           RelativeLayout.LayoutParams.MATCH_PARENT,
           RelativeLayout.LayoutParams.MATCH_PARENT);
           rl.addView(mAuthView, params1);
```

10.10.4 New APIs

- `OMAuthenticationContext#getTokens(List<String> scopes):`

The API returns a list of tokens that are valid and match the passed list of scopes. The passed scopes should match or be a subset of the scopes associated with the access token held by the SDK. If `null` is passed, then the SDK returns all the non-expired access tokens at a given time.

- `OMAuthenticationContext#isValid(List<String> scopes boolean refreshExpiredToken)`:

  This API returns boolean `true` or `false` if the access token for the given scope is valid or expired respectively. If the access token is expired, the SDK will try to refresh the same internally based on the value set for the refreshExpiredToken passed in the function call. If this value is set as `true` the SDK will refresh the access token. The refresh is possible if the following conditions are met:

  – The server supports refreshing for the mobile clients and given authorization grant types.

  – If SDK has a valid refresh token value associated with the access token retrieved earlier. This shows that refreshing the access token is possible only in `OAuthAuthorizationGrantType.Authorization_Code`.

10.10.5 Getting the Tokens From SDK

The SDK will expose all the tokens obtained during the OAuth2.0 flows both through `OMAuthenticationContext#getTokens()` which returns the map of tokens and `OMAuthenticationContext#getTokens(Set<String>)` which returns the list of tokens matching the passed scopes. Refer to the following table to obtain the token from the SDK.

Token Type	Constant value used as name
access_token	OMSecurityConstants.OAUTH_ACCESS_TOKEN ("oauth_access_token")
user_assertion	OMSecurityConstants.OM_OAUTH_USER_ASSERTION_TOKEN ("user_assertion")
usertoken	OMSecurityConstants.USER_TOKEN ("USERTOKEN")
oam_id	OMSecurityConstants.OAM_ID ("OAM_ID")

10.10.6 Using the External Browser

The initialization and the call to authenticate is for same for both browser modes, embedded or external. However, if the application uses the external browser to complete authentication, there are a few additional steps involved. This is because the response must be retrieved from the authorization server back to the application so that the SDK can complete the authentication flow. The following steps are followed:

- You must register the application or activity with the data or scheme. This is the same as the redirect url mentioned during initialization.
- After the activity is registered for the url scheme similar to redirect url, you must invoke `processSSORequest(intent)` in `NewIntent()`(single instance activity) or on `Resume()` of the activity. In doing so, the SDK will internally parse the response

from the server and return the result `Authcontext` or `OMMobileSecurityException` via the `processAuthenticationResponse` response.

This model is similar to SSO-based authentication. The sample codes are provided in this section.

Add intent filters in the AndroidManifest.xml to register the app for the redirect URL so that the application catches the response from the authorization server if an external browser is used. You must add the intent filter in the activity on which you want to get the control back and complete the authentication process.

If you want to register for redirect url like : example://

```
<intent-filter>
                <data android:scheme="geektech" />
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.BROWSABLE" />
                <category android:name ="android.intent.category.DEFAULT"/>
            </intent-filter>
```

If you want to register for redirect url like :http://localhost

```
<intent-filter>
<data android:scheme="http" android:host="localhost" />
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.BROWSABLE" />
<category android:name ="android.intent.category.DEFAULT"/>
</intent-filter>
```

In order to complete the authentication, you must get the intent used by the external browser to invoke your activity whose url scheme is registered as the redirect url, which was mentioned during the initialization. Then you must call `processSSORequest` on the `OMMobileSecurityService` instance.

```
    @Override
    protected void onNewIntent(Intent intent)
    {
        super.onNewIntent(intent);
        System.out.println("MainActivity--- onNewIntent()");
        if (intent != null && intent.getData() != null)
        {
            try
            {
                getMobileSecuritySevice().processSSORequest(intent);
            }
            catch (OMMobileSecurityException e)
            {
                e.printStackTrace();
            }
        }
    }
```

10.10.7 Accessing Protected Resources

After successful authentication with the OAuth server, having a valid access token for a particular set of scopes, the application can now access the resources protected by the authorization server. The following new APIs must be used in order to perform this request:

- OMHTTPRequest#executeHTTPRequest(HttpRequest httpRequest
- OMHTTPRequestCallback callback, List<String> scopes, boolean isAsync)

This method can be used to access any web resource protected by a OAuth2.0 server. This method will inject the access token based on the scopes passed in the request and successfully retrieve the response from the server. If the access token for the requested scopes is expired, then this method will internally refresh the token. This can be performed in an asynchronous way, providing a `{@link OMHTTPRequestCallback}` object. The response will be delegated through this call back after the request is executed.

```
*@param request: an instance of {@link HttpRequest} object
* @param callback
* an instance of {@link OMHTTPRequestCallback} , by on which the
* SDK will delegate the response if the request is asynchronous
* @param isAsync
* boolean if the request is to be performed asynchronously or
* not .
* @return an instance of {@link HttpResponse} If the request is
* asynchronous the result will be sent via
* {@link OMHTTPRequestCallback} instance initially registered.
```

This method will be called by the SDK once `OMHTTPRequest#executeHTTPRequest(org.apache.http.HttpRequest, OMHTTPRequestCallback, boolean)`, `OMHTTPRequest#executeHTTPRequest(org.apache.http.HttpRequest, OMHTTPRequestCallback, java.util.List, boolean)` is executed. The SDK will delegate the result of the execution through this callback.

```
* @param request
* {@link OMHTTPRequest} instance on which the request is
* executed .
* @param response
* {@link HttpResponse} object obtained after execution of the
* request .This will be null when there is failure during
* execution.
* @param exception
* {@link OMMobileSecurityException} This contains the failure
* details if any.
*/
```

Sample code:

```
String authReq1 = "https://www.googleapis.com/oauth2/v1/userinfo";
  List<String> scope1 = new ArrayList<String>();
           scope1.add("https://www.googleapis.com/auth/userinfo.email");
           scope1.add("https://www.googleapis.com/auth/userinfo.profile");
OMHTTPRequest omrequest = new OMHTTPRequest(
                        getMobileSecuritySevice());
                HttpRequest httpRequest = new HttpGet(authReq1);
                try
                {
                       omrequest1.executeHTTPRequest(httpRequest1,
                            new OMHTTPRequestCallback()
                            {
                                @Override
                                public void processHTTPResponse(
                                        OMHTTPRequest request,
                                        HttpResponse response,
                                        OMMobileSecurityException exception)
                                {
                                    //Handle Your code for parsing response here .
                                    //not adding the source for parsing the
```

```
response as its up to the app dev how they want to make use of this.

                                }
                            }, scope1, true);
                }
                catch (OMMobileSecurityException e)
                {
                    e.printStackTrace();
                }
```

10.10.8 Credential Collection

For IMPLICIT, and AUTHORIZATION_CODE grant types, the SDK invokes an external or embedded browser. The authorization server will load the login page in the browser. Hence, the end user submits the login credentials directly to the authorization server.

For RESOURCE_OWNER, CLIENT_CREDENTIALS, ASSERTION and OAM_CREDENTIAL grant types, the SDK will by default throw a native UI to collect the login credentials of the end user. However the application can direct the SDK to throw a custom view or UI for credential collection. See Section 10.13, "Login View and KBA View Customization," for detailed information.

> **Note:** Only the remember user name flag will be entertained in case of OAuth2.0 (as in resource owner grant type SDK collects the credentials). The Remembering Credentials section provides detailed information.

10.11 Invoking REST Web Services

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 and in local storage.

The Mobile and Social Android Client SDK provides the following method to access REST Web services or any Web resource protected by Access Manager:

```
OMMobileSecurityService#processHttpRequest(HttpRequest httpRequest)
```

To access them, however, the user should first authenticate. The Android Client SDK gets access tokens for any given resource using the user token obtained during authentication, provided the resource is protected by an Access Management 11g R2 WebGate.

In the following sample code, if a valid user token is not found for the current application (for example, if the user is not authenticated), the code throws an exception. Because this method contacts the server across the network, execute it in a background thread.

```
HttpRequest httpRequest = new HttpGet("http://abc.example.com:7778/index.html");
// It is important to set the User-Agent to the values configured in the OAM 11g R2
// WebGate user defined parameters.
// We need to configure OAM 11g R2 WebGate with these parameters:
//  i) OAMAuthUserAgentPrefix=<Prefix for User-Agent HTTP header> (Example: OIC)
// ii) OAMAuthAuthenticationServiceLocation=<OIC Server URL> (Example:
//     http://host123.us.example.com:14100/oic_rest/rest/mobileoamauthentication)
httpRequest.setHeader("User-Agent", "OAMMS-Agent");
try
{
```

```
    HttpResponse response = mss.processHttpRequest(httpRequest);
    // The developer may extract the useful information from the "response" object
}
catch (OMMobileSecurityException mse)
{
    Log.d("processHttpRequest", mse.getLocalizedMessage());
}
```

The `OMMobileSecurityService#processHttpRequest(HttpRequest)` method 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 `OAMAuthUserAgentPrefix` and `OAMAuthAuthenticationServiceLocation` properties. The same property values must be specified by the mobile application in its header as shown in the sample code.

The following steps describe the internal flow when this method is called:

1. `OMMobileSecurityService#processHttpRequest(HttpRequest)` 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 and accesses the protected URL.

In version 11.1.2.2 of the SDK, an option to execute the Web service request asynchronously was added. To execute the request in the background, the SDK accepts a call back `OMHTTPRequestCallback` object. The SDK notifies the application when the request is executed and the REST request response is propagated to the application using this callback object. The `OMHTTPRequest` class receives and responds to the request. Following is the method signature.

```
OMHTTPRequest#executeHTTPRequest(HttpRequest httpRequest , OMHTTPRequestCallback
callback , boolean isAsync)
```

Where:

- `httpRequest` is the HttpRequest object
- `OMHTTPRequestCallback` is the callback object that the SDK uses to notify the app that the request is completed. The response/error is sent using this callback object by this method:

  ```
  OMHTTPRequestCallback#processHTTPResponse(OMHTTPRequest request, HttpResponse
  response, OMMobileSecurityException exception)
  ```

The application needs to implement this interface and handle the response/error handling logic.

- `isAsync` is a Boolean that tells if the request need to be executed asynchronously or not. If false then the execution is similar to `OMMobileSecurityService#processHttpRequest(HttpRequest)`.

Following is sample code that demonstrates an asynchronous REST request.

```
OMHTTPRequest omrequest1 = new OMHTTPRequest(getMobileSecuritySevice());
HttpRequest httpRequest1 = new HttpGet("http://abc.example.com:7777/restService")
;

try
{
omrequest1.executeHTTPRequest(httpRequest1,
new OMHTTPRequestCallback()
  {

    @Override
    public void processHTTPResponse( OMHTTPRequest request, HttpResponse response,
OMMobileSecurityException exception)
    {
     String data1="";
     if (response != null)
      {
       try
       {
        data1 = parseHttpResponse(response);
       }
       catch (IOException e)
       {

        // TODO Auto-generated catch block
        e.printStackTrace();
       }
       catch (JSONException e)
       {
        // TODO Auto-generated catch block
        e.printStackTrace();
       }
      }
     else
     {
       if (exception != null)
       {
        data1 = exception.getErrorMessage();
       }
     }
     if(data1 != null)
     {
       Message msg = handler.obtainMessage(MSG_SHOW_DIALOUGE);
       msg.obj = data1;
       handler.sendMessage(msg);
     }

   }
  }, true);
}
catch (OMMobileSecurityException e)
{
```

```
e.printStackTrace();
}
```

10.12 Creating a Custom Mobile Single Sign-on Agent App Using the Android Client SDK

This section covers how a custom mobile single sign-on (SSO) agent app can be created using the Android Client SDK.

When applications that are configured as SSO clients make a call to `OMMobileSecurityService#authenticate()`, the Android Client SDK delegates the request to the SSO agent application if it is already installed on the device. Otherwise, it throws an exception. The Android Client SDK frames the authentication request and invokes the SSO agent app using an intent. In order to process the authentication request, the SSO Agent app needs to retrieve this intent and call `OMMobileSecurityService#processSSORequest(intent)`. The SDK internally processes and validates the authentication request and returns control back to the client app with a valid authentication context.

You can extend the `Application` class to keep a singleton instance for OMMobileSecurityService that accepts either a null intent or one that contains a configuration URL. The following code snippet shows how to initialize an OMMobileSecurityService instance using the configuration URL based mechanism for a new intent and using the config properties stored in `SharedPreferences` for subsequent cases.

```
public OMMobileSecurityService getMobileSecurityService(Context context,
            Intent intent, OMMobileServiceCallback callback)
            throws JSONException, OMMobileSecurityException
{
     if (!mssInitialized)
     {
            if (intent == null || intent.getData() == null)
            {
                mss = new OMMobileSecurityService(context, callback);
            }
            else
            {
                Map<String, Object> configMap = OMMobileSecurityService
                        .parseConfigurationURI(context, intent, true, null);

                mss = new OMMobileSecurityService(context, configMap, callback);
            }
            if (mss == null)
            {
                throw new OMMobileSecurityException(
                        OMErrorCode.INITIALIZATION_FAILED, null, context);
            }
            else
            {
                mssInitialized = true;
            }
     }
     else
     {
            mss.registerCallback(callback);
     }
     return mss;
}
```

The following code snippet shows the checks that you should perform in the agent app before processing the client requests.

```
@Override
        protected void onResume()
        {
            super.onResume();
            mMobileSecurityService.registerActivity(this);
            if (!(SSOApplication) getApplication().isSetupCompleted())
            {
                mMobileSecurityService.setup();
            }

            Intent intent = getIntent();

            if (intent.getData() != null)
            {
                 boolean isConfigurationIntent = false;
                 /* Check if the intent URI contains the configuration parameters
                    which can be passed to the the SSOApplication class. */

                 if (isConfigurationIntent == true)
                 {
                      mMobileSecurityService = (SSOApplication)
getApplication.getMobileSecurityService(getApplicationContext(), intent, new
SSOAgentCallback();
                 }
                 else
                 {
                      mMobileSecurityService = (SSOApplication)
getApplication.getMobileSecurityService(getApplicationContext(), null, new
SSOAgentCallback();
                 }
            }
            if ((intent.getData() != null && !isConfigurationIntent) ||
intent.getExtras() != null)
            {
                try
                {
                    authView = mMobileSecurityService.processSSORequest(intent);
                    if (authView != null)
                    {
                        setContentView(authView);
                        // The developers have full control on how they want to
                        // display this view in their app.
                    }
                }
                catch (OMMobileSecurityException e)
                {
                    Log.w(TAG, e.getErrorMessage());
                }
            }
        }
```

Please note that you can use any design approach. The sample code is included to demonstrate the usage of the SDK.

```
class SSOAgentCallback implements OMMobileServiceCallback
{
    @Override
```

```
    public void processSetupResponse(OMApplicationProfile profile,
            OMMobileSecurityException mse)
    {
        if (profile != null)
        {
            // Setup successful you may go for authentication now
            ((SSOApplication) getApplication()).setSetupCompleted(true);
            Toast.makeText(SSOActivity.this, "Setup Done",
                    Toast.LENGTH_SHORT).show();
        }
        else
        {
            Toast.makeText(SSOActivity.this, "Setup failed",
                    Toast.LENGTH_LONG).show();

        }
    }

    @Override
    public void processAuthenticationResponse(
            OMAuthenticationContext context, OMMobileSecurityException mse)
    {
        /*
         * Handle post authentication in case of SSO self auth request, i.e.
         * when the SSO Agent app calls
         * OMMobileSecurityService#authenticate(). Note: If authentication
         * request comes from a client application , we will not get this
         * call back, as the sdk will redirect to the client app after
         * successful authentication
         */
    }
}
```

10.13 Login View and KBA View Customization

The SDK displays a basic login view and knowledge-based answer view by default. To create custom user-defined layouts do the following:

1. The `OMMobileSecurityService` class provides a method with the following signature:

   ```
   public void setCredentialCollector(OMAuthenticationServiceType serviceType,
   OMCredentialCollector viewHandler)
   ```

   This method can be used to set a custom view as follows:

   ```
   mobileSecurityService.setCredentialCollector(OMAuthenticationServiceType.DEVICE
   _AUTHENTICATION, new MyLoginView(getApplicationContext()));
   mobileSecurityService.setCredentialCollector(OMAuthenticationServiceType.KBA_
   AUTHENTICATION, new MyKBAView(getApplicationContext()));
   ```

   Follow this approach to set custom views for any of the authentication service types supported by `OMAuthenticationServiceType`.

2. `MyLoginView` is a custom class that implements the `OMCredentialCollector` interface. The custom view to be displayed by the SDK has to be returned from this method:

   ```
   OMCredentialCollector#processViewRequest(Map<String, String>
   inputParams, OMCredentialCollectorCallback callback)
   ```

Write this method to collect a user name and password or security answer, then pass these values to the SDK. Put the values in an object of type `Map<String, String>` and pass the object to the SDK using `callback.processLoginResponse(Map<String, String>)`.

For Log in View customization, use the following keys for the user name and password values:

```
OMSecurityConstants.USERNAME
OMSecurityConstants.PASSWORD
```

For KBA View customization, use the following key for the security answer:

```
OMSecurityConstants.ANSWER_STR
```

If any exception happens in a request, the SDK will populate it in the `inputParams` map, which is an input parameter to the following method:

```
OMCredentialCollector#processViewRequest(Map<String, String> inputParams,
OMCredentialCollectorCallback callback)
```

Design the custom view such that, if the SDK sends an error message, it should be shown in the view. Use the following key to obtain the error message:

```
OMSecurityConstants.ERROR_MESSAGE
```

For the KBA view, the question will be available as part of the `inputParams` map with the key as follows:

```
OMSecurityConstants.QUESTION_STR
```

Define `OMCredentialCollector#processViewRequest(Map<String, String> inputParams, OMCredentialCollectorCallback callback)` such that it will retrieve the question from the `inputParams` map and display the question in the view.

Refer to the following sample code for Log in View customization:

```
class MyLoginView implements OMCredentialCollector
{
    ...
    @Override
    public View processViewRequest(Map<String, String> inputParams,
OMCredentialCollectorCallback callback)
    {
        // Create the basic auth view
        LayoutInflater inflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        basicAuthView = inflater.inflate(R.layout.mybasicauth, null);

        ...
        // You can enable username and password EditText Views, which may be
disabled when the log in button is pressed

        Button button = (Button) basicAuthView.findViewById(R.id.login);
        button.setOnClickListener(new AuthenticationLoginListener(callback));

        Button cancelButton = (Button) basicAuthView.findViewById(R.id.cancel);
        cancelButton.setOnClickListener(new
AuthenticationCancelListener(context, callback));

        String errorMsg = inputParams.get(ERROR_MESSAGE);
```

```
        TextView errorMsgTv = (TextView)
basicAuthView.findViewById(R.id.errorMsg);
        if (errorMsg != null && !errorMsg.isEmpty())
        {
            errorMsgTv.setVisibility(View.VISIBLE);
            errorMsgTv.setText(errorMsg);
        }
        else
        {
            errorMsgTv.setVisibility(View.INVISIBLE);
        }
        inputParams.remove(ERROR_MESSAGE);

        return basicAuthView;
    }
    //--------------------------------------------------------------------------
    // Inner class which handles the OnClick event
    //--------------------------------------------------------------------------

    private class AuthenticationLoginListener implements OnClickListener
    {
        OMCredentialCollectorCallback callback;

        AuthenticationLoginListener(OMCredentialCollectorCallback callback)
        {
            this.callback = callback;
        }

        @Override
        public void onClick(View v)
        {
            // The developer may show a progress bar and disable the Login and
Cancel buttons
            Map<String, String> outputParams = new HashMap<String, String>();

            outputParams.put(USERNAME, username);
            outputParams.put(PASSWORD, password);

            basicAuthView.invalidate();

            callback.processLoginResponse(outputParams);
        }
    }
}
// This class is a part of the samples for both Login View and KBA View
// Customization

class AuthenticationCancelListener implements OnClickListener
{
    private OMCredentialCollectorCallback callback;
    private Context context;

   AuthenticationCancelListener(Context context, OMCredentialCollectorCallback
callback)
    {
        this.context = context;
        this.callback = callback;
    }

    @Override
```

```
    public void onClick(View view)
    {
        callback.processCancelResponse();
    }
}
```

Sample code for KBA View customization has been given below:

```
class KBAView implements OMCredentialCollector
{
    ...
    @Override
    public View processViewRequest(Map<String, String> inputParams,
            OMCredentialCollectorCallback callback)
    {
        // Create the view for KBA
        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        kbaAuthView = inflater.inflate(R.layout.mykbaview, null);

        Button button = (Button) kbaAuthView.findViewById(R.id.casubmit);
        button.setOnClickListener(new AuthenticationLoginListener(callback));

        Button cancelButton = (Button) kbaAuthView.findViewById(R.id.cacancel);
        cancelButton.setOnClickListener(new AuthenticationCancelListener(
                context, callback));

        TextView question = (TextView) kbaAuthView.findViewById(R.id.cques);
        question.setText(inputParams.get(QUESTION_STR));

        return kbaAuthView;
    }

    //--------------------------------------------------------------------------
    // Inner class which handles the OnClick event
    //--------------------------------------------------------------------------

    private class AuthenticationLoginListener implements OnClickListener
    {
        OMCredentialCollectorCallback callback;

        AuthenticationLoginListener(OMCredentialCollectorCallback callback)
        {
            this.callback = callback;
        }

        @Override
        public void onClick(View v)
        {
           // You can show a progress bar and disable the submit and
           // cancel buttons

            String answer = chalngAns.getText().toString();
            Map<String, String> outputParams = new HashMap<String, String>();
            outputParams.put(ANSWER_STR, answer);

            callback.processLoginResponse(outputParams);
        }
    }
}
```

10.14 Using the Cryptography APIs

This section covers the cryptography APIs that are exposed by the SDK. The SDK uses these APIs to protect and store credentials for offline authentication. For detailed information about API methods, please refer to the API documentation included in the SDK download.

`OMCryptoService` is the class that supports all of the features like hashing, encryption, decryption, and matching whose instance can be obtained using `mss` as follows:

```
OMCryptoService cryptoService =  mss.getCryptoService();
```

> **Note:** In the 11.1.2.2 release, the `OMCryptoService` added support for passing a custom key for features such as encryption, decryption, and matching, while keeping support for legacy features that work independent of any key. Applications can now manage the life cycle of the crypto key themselves. They do not have to depend on the SDK generated key. It is the application's duty to provide the same key for encryption and decryption of the same data. The custom key that the SDK accepts is a `byte[]` array. The application must provide a compliant key size based on the encryption algorithm being used.

Following is a high-level summary of the functionality provided by the cryptography API.

Hashing

Use the hashing capability to produce a hash of plain text based on the crypto scheme algorithm. If it is a salted algorithm, then it uses the salt length that is given as input. The last parameter in the API determines whether to prefix the algorithm name.

The output is of the following format:

<Algorithm Name><HashedContent><Salt>

- The hashed content and salt are Base-64 encoded.
- The algorithm name will be prefixed only if it is requested.
- The salt will be present only if the algorithm is a salted algorithm.

```
cryptoService.hash(plainText, scheme, 10, true);
```

Symmetric Key Encryption

This API helps perform symmetric key encryption and decryption of data. The key used for encryption is either generated automatically and stored in the credential store (the SDK default key), or a custom key provided by the application.

The output of encryption is of the following format:

<scheme/mode/padding> <Initialization Vector><Encrypted text>:

Both the initialization vector and the encrypted text are Base-64 encoded. The initialization vector will be used later for decryption as well.

```
cryptoService.encrypt(plainText, scheme, msc.getCryptoMode(), msc.getCryptoPadding(), true);
```

If using a custom key:

```
cryptoService.encrypt(plainText, scheme, msc.getCryptoMode(), msc.getCryptoPadding(), true,
customKey);
```

The key for decryption is automatically retrieved from the credential store automatically (that is, the SDK default Key), or provided by the application if the encryption was done using a custom key. The decrypted text is given as output by the following method. The SDK default key is generated automatically by the SDK when the `encrypt` method is called for the first time.

```
cryptoService.decrypt(encodedText);
```

If using a custom key:

```
cryptoService.decrypt(encodedText,customKey);URL
```

Matching of Plain Text and Encrypted/Hashed Text

The following method will check the encoded text given as input and find the `oracle.idm.mobile.crypto.CryptoScheme` that was used to encrypt or hash it. If it was encrypted, it will decrypt the encoded text and check whether it matches with plain text. It it was hashed, it will perform hashing on the plain text and check whether it matches with the encoded text. Depending on the match, it will return true or false.

Note: If the text is encrypted using a custom Key, the application should pass the same key in the `match()` method.

```
cryptoService.match(password, passwordStored, mss.getMobileSecurityConfig().getSaltLength());
```

If using a custom key:

```
cryptoService.match(password, passwordStored, mss.getMobileSecurityConfig().getSaltLength(),
customKey);
```

10.15 Using the Auto Login and the Remember Credentials Features

The Mobile and Social Android Client SDK provides APIs that can securely store user credentials and play them back to a login server with or without user interaction. Deploy this feature in apps where security is not critically important to make it easier for users to log in. This feature requires at least version 11.1.2.2.0 of the Mobile and Social Client SDK.

To use this feature, the user selects one of up to three option boxes on the login screen. (You can choose to enable one, two, or all three of these options in your app.) If more than one option is selected, the higher priority feature takes precedence. The priority of the options is as follows:

1. Auto Login
2. Remember Credentials
3. Remember User Name Only

So for example, if Auto Login and Remember Credentials are selected, then Auto Login takes priority.

The three options are described as follows:

- Auto Login - The Mobile and Social Android Client SDK securely caches the user's credentials and automatically supplies them at the login screen. This option does not require any user interaction to log the user in. The user see a progress screen that provides feedback while authentication is underway.

- Remember Credentials - The Mobile and Social Android Client SDK securely caches the user's credentials and auto-fills the user name and password fields on the login screen. The user has to click the Login button to proceed with the authentication process. The user can enter different credentials and make changes to the option boxes if necessary.
- Remember User Name Only - The Mobile and Social Android Client SDK securely caches the user name and auto-fills the user name on the login screen. The user has to input the password and click the login button to proceed with the authentication process. The user can enter different user name and make changes to the option boxes if necessary.

Enabling the Feature

This is a client-side only feature that does not require server configuration to deploy. To enable the features described, the following SDK configuration parameters should be added to your code:

Table 10–8 Configuration parameters used to enable auto login and remember credentials features

Parameter	Description
`OM_PROP_AUTO_LOGIN_ALLOWED`	Boolean value that enables/disables the Auto Login feature.
`OM_PROP_REMEMBER_CREDENTIALS_ALLOWED`	Boolean value that enables/disables the Remember Credentials feature.
`OM_PROP_REMEMBER_USERNAME_ALLOWED`	Boolean value that enables/disables the Remember User Name Only feature.

Handling User Preferences

Pass the following properties while initializing the `mss` object to pre-populate the option boxes with a default value. Only one of the following options can be true. If all the option box states are false, all of the option boxes on the login screen will be empty.

Table 10–9 Configuration parameters used to set the option box default values

Parameter	Description
`OM_AUTO_LOGIN_DEFAULT`	Boolean value that specifies the default value of the "Auto Login" option box.
`OM_REMEMBER_CREDENTIAL_DEFAULT`	Boolean value that specifies the default value of the "Remember Credential" option box.
`OM_REMEMBER_USERNAME_DEFAULT`	Boolean value that specifies the default value of the "Remember User Name Only" option box.

Persist option box states as key-value pairs in `SharedPreferences`. Use the combination of the server URL and the application identifier as a key so that the user preferences for each login connection can be uniquely identified.

Clearing Credentials and Preferences From Mobiles Devices

Authentication failure can result if the user credentials were changed since the last login, or if the mobile device is not able to reach the authentication service due to a network or server issue. If authentication fails using the stored credentials, the SDK deletes the stored password (if present) from the shared preferences. The login page will then either revert to the Remember User Name Only feature, or control will revert to the mobile app using `OMMobileServiceCallback`. This decision is based on an

SDK-level parameter named `OM_PROP_MAX_LOGIN_ATTEMPTS`, which is a numeric value that stores how many incorrect attempts for authentication are allowed before control is given back to the mobile app.

The SDK will clear a stored user password from the mobile device in the following scenarios:

- The user authentication fails (for example, if the server password is no longer valid, the user is blocked, or if the user authentication fails for another reason).
- The Mobile and Social Android Client SDK `logout` method is called
- A session time-out is detected

The SDK will clear the stored user name, password, and option box states from the mobile device in the following scenario:

- The Mobile and Social Android Client SDK logout method is called and the `forgetDevice` parameter is set to `true`

Creating a Custom Login Screen

The Mobile and Social Android Client SDK sends flags in the input params map in the `processViewRequest` callback that you use to inflate your custom views. (See Section 10.13, "Login View and KBA View Customization" for more information.) These flags are specific to the Auto Login and Remember Credentials features and indicate if a given feature is enabled or disabled during initialization, and what the UI preference for these features should be set to. The SDK also sends the user name and the password as needed for the Remember User Name and Remember Credentials options.

If you want to create a custom login screen instead of using the basic login view provided by the SDK, do the following:

- Receive and interpret the Auto Login / Remember Credentials flags correctly
- Control the visibility of the option boxes for flags that indicate if a feature is enabled
- For flags that indicated UI preferences, update the state of the option boxes when presenting the view to the user
- After the submit button is pressed, read the state of the option boxes and send the same

The following table lists the keys that you need to use to access credential properties in the authentication `inputParams` Map.

Table 10–10 Keys used to access credential properties in the inputParams Map

Key	Description
`OMMobileSecurityService.OM_PROP_AUTO_LOGIN_ALLOWED`	Boolean value that enables/disables the Auto Login feature.
`OMMobileSecurityService.OM_PROP_REMEMBER_CREDENTIALS_ALLOWED`	Boolean value that enables/disables the Remember Credentials feature
`OMMobileSecurityService.OM_PROP_REMEMBER_USERNAME_ALLOWED`	Boolean value that enables/disables the Remember User Name Only feature
`OMSecurityConstants.OM_AUTO_LOGIN_PREF`	Boolean value that specifies the user's preference regarding the Auto Login feature based on the last authentication.

Table 10–10 (Cont.) Keys used to access credential properties in the inputParams Map

Key	Description
OMSecurityConstants.OM_REMEMBER_CREDENTIALS_PREF	Boolean value that specifies the user's preference regarding the Remember Credentials feature based on the last authentication.
OMSecurityConstants.OM_REMEMBER_USERNAME_PREF	Boolean value that specifies the user's preference regarding the Remember User Name Only feature based on the last authentication.
OMSecurityConstants.USERNAME	String value that specifies the user's user name.
OMSecurityConstants.PASSWORD	String value that specifies the user's password.

The following code sample shows how you can control the visibility and UI state of the option boxes using the remember credentials flags.

```
// RC
CheckBox autoLogin = (CheckBox) basicAuthView.findViewById(R.id.autoLoginCB);
CheckBox rememberCredentials = (CheckBox)
basicAuthView.findViewById(R.id.rememberCredentialsCB);
CheckBox rememberUsername = (CheckBox)
basicAuthView.findViewById(R.id.rememberUsernameCB);
// RC
button.setOnClickListener(new AuthenticationLoginListener(callback));

Button cancelButton = (Button) basicAuthView.findViewById(R.id.cancel);
cancelButton.setOnClickListener(new AuthenticationCancelListener(asm,callback));

EditText username = (EditText) basicAuthView.findViewById(R.id.username);
username.setEnabled(true);
EditText password = (EditText) basicAuthView.findViewById(R.id.password);
password.setEnabled(true);
String usernameFromRCStore = null;
String passwordFromRCStore = null;
String identityFromRCStore = null;
boolean rcUseCase = false;
// Try to get the user name or password stored by the SDK.
// Note: This is possible only if the Remember Credentials feature is
// enabled.
if (inputParams.containsKey(USERNAME))
{
  usernameFromRCStore = (String) inputParams.get(USERNAME);
}
if (inputParams.containsKey(PASSWORD))
{
  passwordFromRCStore = (String) inputParams.get(PASSWORD);
}

Object autoLoginAllowedObj = inputParams.get(OMMobileSecurityService.OM_PROP_AUTO_
LOGIN_ALLOWED);
if (autoLoginAllowedObj != null)
{
  boolean autoLoginAllowed = ((Boolean) autoLoginAllowedObj);
  // feature enabled now set the visibility to true.
  if (autoLoginAllowed)
  {
```

```
    autoLogin.setVisibility(View.VISIBLE);
    Object alChecked = inputParams.get(OMSecurityConstants.OM_AUTO_LOGIN_PREF);
    if (alChecked != null)
    {
      autoLogin.setChecked((Boolean)alChecked);
    }
    rcUseCase = true;
  }
}

Object remCredObj = inputParams.get(OMMobileSecurityService.OM_PROP_REMEMBER_
CREDENTIALS_ALLOWED);
if (remCredObj != null)
{
  boolean rememberCredentialsAllowed = ((Boolean) remCredObj);
  if (rememberCredentialsAllowed)
  {
    // feature enabled now set the visibility to true.
    rememberCredentials.setVisibility(View.VISIBLE);
    Object rcChecked = inputParams.get(OMSecurityConstants.OM_REMEMBER_
CREDENTIALS_PREF);
    if (rcChecked != null)
    {
      rememberCredentials.setChecked((Boolean)rcChecked);
    }
    rcUseCase = true;
  }
}

Object remUserObj = inputParams.get(OMMobileSecurityService.OM_PROP_REMEMBER_
CREDENTIALS_ALLOWED);
if (remUserObj != null)
{
  boolean rememberUsernameAllowed = ((Boolean) remUserObj);
  if (rememberUsernameAllowed)
  {
    // feature enabled now set the visibility to true.
    rememberUsername.setVisibility(View.VISIBLE);
    Object ruChecked = inputParams.get(OMSecurityConstants.
OM_REMEMBER_USERNAME_PREF);
    if (ruChecked != null)
    {
      rememberUsername.setChecked((Boolean)ruChecked);
    }
    rcUseCase = true;
  }
}
// pre filling
if (rcUseCase)
{
  if (usernameFromRCStore != null && usernameFromRCStore.length() > 0)
  {
    username.setText(usernameFromRCStore);
  }
  if (passwordFromRCStore != null && passwordFromRCStore.length() > 0)
  {
    password.setText(passwordFromRCStore);
  }
}
// RC
```

This next code sample shows how to control the visibility and UI state of the option boxes when the submit (or log on) button is pressed.

```
// Do the registration and then perform the authentication
EditText username = (EditText) basicAuthView.findViewById(R.id.username);
username.setEnabled(false);

EditText password = (EditText) basicAuthView.findViewById(R.id.password);
password.setEnabled(false);


Button button = (Button) basicAuthView.findViewById(R.id.login);
button.setEnabled(false);

Button cancelButton = (Button) basicAuthView.findViewById(R.id.cancel);
cancelButton.setEnabled(false);
// RC
CheckBox autoLogin = (CheckBox) basicAuthView.findViewById(R.id.autoLoginCB);
CheckBox rememberCredentials = (CheckBox)
basicAuthView.findViewById(R.id.rememberCredentialsCB);
CheckBox rememberUsername = (CheckBox)
basicAuthView.findViewById(R.id.rememberUsernameCB);
// RC
InputMethodManager imm = (InputMethodManager)
asm.getApplicationContext().getSystemService(Context.INPUT_METHOD_SERVICE);
imm.hideSoftInputFromWindow(username.getWindowToken(), 0);

String uname = username.getText().toString();
String pwd = password.getText().toString();

Map<String, Object> outputParams = new HashMap<String, Object>();
//sending the username passord for the authentication purpose.
outputParams.put(USERNAME, uname);
outputParams.put(PASSWORD, pwd);
// RC
//sending UI preferences for the current auth /login screen to the SDK
outputParams.put(OMSecurityConstants.OM_AUTO_LOGIN_PREF,autoLogin.isChecked());
outputParams.put(OMSecurityConstants.OM_REMEMBER_CREDENTIALS_
PREF,rememberCredentials.isChecked());
outputParams.put(OMSecurityConstants.OM_REMEMBER_USERNAME_
PREF,rememberUsername.isChecked());
// RC

basicAuthView.invalidate();

callback.processLoginResponse(outputParams);
```

10.16 Invoking the CredentialStoreService With the Android Client SDK Secure Storage Module

The SDK `CredentialStoreService` stores and retrieves sensitive data. Internally, the SDK uses the `SharedPreferences` class provided in the Android SDK. For more information, see the following Web page:

http://developer.android.com/reference/android/content/SharedPreferences.html

Start using `CredentialStoreService` by getting a reference to an instance of `OMCredentialStore` using `OMMobileSecurityService#getCredentialStoreService()`.

Once the reference to `CredentialStoreService` is obtained, credentials can be added, updated, deleted, or read from the Credential Store. Please refer to the SDK documentation for more details.

Following is a sample snippet of code illustrating the usage:

```
OMCredentialStore credentialStore = mss.getCredentialStoreService();
credentialStore.addCredential("sampleKey", "sampleUsername", "samplePassword",
"sampleTenantName", properties);
// properties is a Map<String, String>. This method will convert all the
// parameters like username, password, tenant name and properties into a JSON
// string and store the same in CredentialStore
OMCredential credential = credentialStore.getCredential("sampleKey");
// This method will retrieve the credential information on supplying the correct
// key
```

10.17 Error Codes

This section describes the error codes and messages that are thrown by the Android Client SDK. The error codes and corresponding messages are coded as an enum and exposed as a public API.

Table 10–11 Mobile and Social Android Client SDK Error Codes and Messages

Error Code	Description
`COULD_NOT_CONNECT_TO_SERVER(1, "Could not establish a connection to the server.")`	This exception is thrown when the SDK cannot connect to the server. This may occur because of a URISyntaxException, an IllegalArgumentException, and so on.
`CHALLENGE_INVALID(2, "Challenge answer is invalid.")`	This exception is thrown when the user does not provide a challenge answer in the case of knowledge-based authentication (KBA).
`UN_PWD_INVALID(3, "Username and Password is invalid.")`	This exception is thrown when the user does not provide a user name and password in the UI before executing the authentication call with the server, as well as when the response from the server is 401 Unauthorized.
`TOKENS_NOT_MATCHED(4, R.string.tokensNoMatch)`	This exception is thrown if the required tokens that you requested are not present in the tokens returned by the server.
`COULD_NOT_PARSE_RESPONSE_FROM_SERVER(5, "Could not parse the response sent from the server.")`	This exception is thrown when the `HttpResponse` from the server cannot be parsed by the SDK.
`DEVICE_NOT_AUTHENTICATED(6, "This device is not yet authenticated with Mobile and Social Server.")`	This exception is thrown when the authentication with `RestAuthenticationService` finds out that the device registration token is not found.
`USER_CANCELED_CERTIFICATE(7, R.string.certificateNotAccepted)`	This exception is thrown if the untrusted SSL certificate is rejected by the user.
`NO_AUTHENTICATION_SCHEME(8, "No authentication scheme is specified.")`	This exception is thrown when there is no authentication scheme provided in the mobile security configuration object.
`INVALID_BASIC_AUTH_URL(9, R.string.invalidBasicUrl)`	This exception is thrown if the supplied authentication URL is not valid.
`USER_AUTHENTICATION_FAILED(10, "Authentication request failed for user %1$s.")`	This exception is thrown when the user authentication fails due to some internal errors from the server.

Table 10–11 (Cont.) Mobile and Social Android Client SDK Error Codes and Messages

Error Code	Description
`UN_PWD_TENANT_INVALID(11, R.string.unPwdTenantMissing)`	This exception is thrown if authentication is not successful due to an incorrect user name or password.
`UNABLE_TO_OPEN_RP_AUTHENTICATION_URL(12, "Unable to open RP Authentication URL.")`	This exception is thrown when the SDK is not able to construct a valid RP login URL.
`UNABLE_TO_OPEN_SSO_AUTHENTICATION_URL(13, "Unable to open SSO Authentication URL.")`	This exception is thrown when the business application is not able to open an SSO agent application because it might not have been installed on the device. Another possible reason is that the Android Application signatures entered in the Application Profiles of the SSO Agent and the SSO Client on the server do not match the signatures of the SSO Agent and SSO Client installed on the device.
`NOT_AUTHORIZED_TO_PARTICIPATE_IN_SSO(14, "Client application is not authorized to participate in SSO.")`	This exception is thrown when the business application is not able to self-authenticate or is not able to participate in SSO. This is due to an invalid configuration in the server console.
`SETUP_FAILED_MS(16, "Application profile cannot be downloaded due to missing configuration")`	This exception is thrown when the application profile cannot be downloaded from the Mobile & Social Server. This happens when the server URL, application ID, or service domain is null or empty.
`TOKENS_NOT_AVAILABLE(17, R.string.tokensNotAvailable)`	This exception is thrown if the `OAM_ID` token is not available when accessing a protected URL through a browser that is configured to invoke the SSO agent app for authentication. This may result if the authentication scheme is not set as MobileOAMAuthentication.
`SETUP_NOT_INVOKED(21, "Setup API is not invoked.")`	This exception is thrown when the `setup()` method of the SDK is not invoked.
`SERVICE_DOMAIN_MISMATCH(22, "Authentication is rejected by the agent as the service domain does not match.")`	This exception is thrown when the service domain of the SSO application does not match the service domain of the SSO client application.
`USER_NOT_YET_AUTHENTICATED(23, "The user is not yet authenticated.")`	This exception is thrown when the user is trying to execute `processHttpRequest()` to access a resource before authenticating.
`APP_SIGNATURE_INVALID(24, "Failed in validating calling application signature.")`	This exception is thrown when the SSO application is not able to validate the application signature of the calling business application.
`INITIALIZATION_FAILED(25, "Initialization of the SDK failed, as the required server information is not supplied.")`	This exception is thrown when any application is initialized through the URL scheme and the appropriate fields are not supplied.
`INTERNAL_ERROR(26, "Internal Error")`	This exception is thrown for all other exceptions raised from the SDK.

11

Developing Applications Using the Social Identity Client SDK

This chapter describes how to use the Social Identity Client SDK to integrate Mobile and Social with supported web and mobile applications. (Prior to version 11.1.2.2, Social Identity was named Internet Identity Services.) This chapter includes the following topics:

11.1 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.

11.2 Introduction to Developing Social Identity Applications

This section covers concepts and requirements that apply to Social Identity application development in general.

Social Identity 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 Social Identity. 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.

11.2.1 About the Social Identity Client SDK

The primary Java package that you will use when working with the Social Identity 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`

11.3 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 Social Identity" 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 11–1 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 11–1 (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 11–1 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);
    }
}
```

11.4 Integrating Social Identity With a Web Application Running on a Server

The Social Identity SDK supports web applications, such as portal sites and consumer-driven web sites that run on Java-compliant application servers.

To integrate Social Identity with a web application, first define the web application on the Mobile and Social server, then integrate the Social Identity 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 Social Identity Login Page With the Web Application
- Handling User Registration
- Handling the Final Return Response

11.4.1 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.

11.4.2 Integrating the Social Identity Login Page With the Web Application

To integrate Social Identity, use the Social Identity Client SDK (`oic_clientssdk.jar`) and modify its login page (`login.jsp`). The Social Identity Client SDK was previously named the Internet Identity Services Client SDK.

There are two ways that you can integrate the Social Identity 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.

11.4.2.1 Adding the Pre-built Social Identity 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 11.3, "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 11–1 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 11–2 Pre-built Login Screen Without Local Login Support

11.4.2.2 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
FBOTQ3M0VBQkE3QzUzQzc0MzM4ODk3RUY3NEJFQkJFOTg2QjE4QkU5RUFGQURFRDE3NUMyRTg0NjRDRjNDQzJGN0Y1QTU4RDlD
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>
```

11.4.3 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

11.4.3.1 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.

11.4.3.2 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 11–3.

Configure the User Registration Properties on the Server

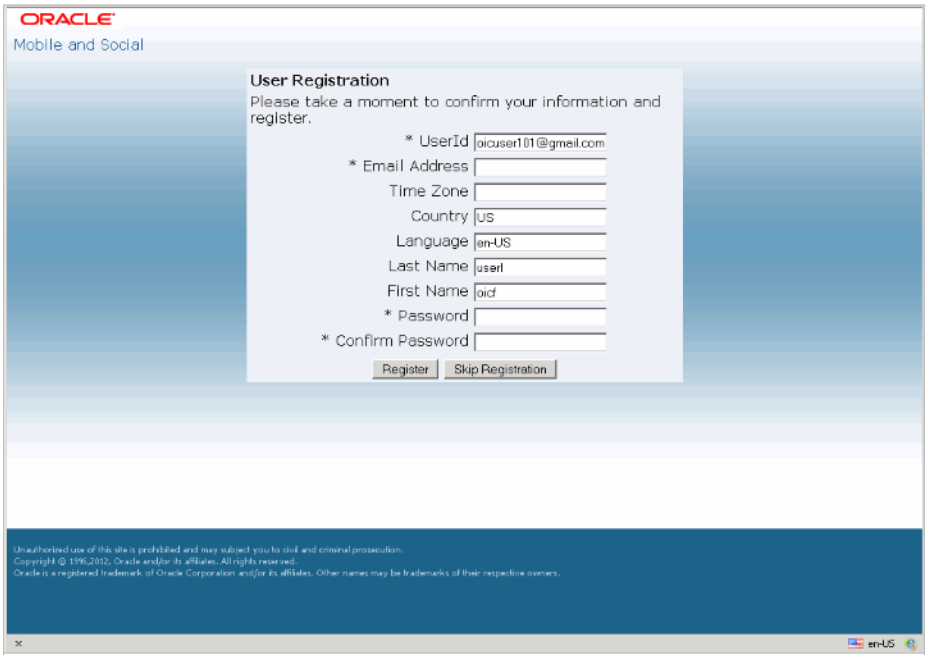
Use the Mobile and Social system administration console to configure the following User Registration properties for the application:

- User Registration (Choose **Enabled**)
- Registration URL (Set to the URL provided with the default Mobile and Social Social Identity 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 11–3 The Mobile and Social Built-In User Registration Page



11.4.4 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 11.4.4.1, "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
NsZS5vaWMudG9rZW4udXNlc19kbiI6Im1haWx0bzpvaWN1c2VyMTAxQGdtYWlsLmNvbSBodHRwczovL3d
3dy5nb29nbGUuY29tL2FjY291bnRzL284L2lkIn0.K4d_gewCE6N61Ng8DsB31BIEEG1S1wMHWpt5CC8u
yfdMgtpEjzs77yFYzNz5vS6x6lsa8fH2FrSgOlxoyoiOnRF8UfJgCzk3VJIDNoDd_2BsYrBQVLGQzWTbs
avO5x09macq6rmOOT1Jbg9napfol8plqgTHPZgm3CH7Rt7jmxA, preferredlanguage=en-US,
lastname=doe, commonname=johndoe, oicInternetIdentityProvider=Google, password=}

See Section 11.4.4.1, "Secured Attribute Exchange (SAE) Token Response Attributes," for information about response attributes.

11.4.4.1 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 11–2 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 11–2 (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.

11.5 Integrating With an Access Manager Protected Web Application

You do not have to write code to integrate Social Identity 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 Social Identity" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

11.6 Integrating Social Identity 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 Social Identity 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.

11.6.1 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.

12

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 and Social Services. This chapter includes the following topics:

- Create a new Authentication Services Provider for Mobile and Social Services
- Create a new Identity Service Provider for Internet Identity Services

12.1 Create a new Authentication Services Provider for Mobile and Social 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

12.1.1 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.

12.1.1.1 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.

12.1.1.2 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.

12.1.2 Building the Custom Authentication Service Provider

Build the custom authentication Service Provider as follows.

12.1.2.1 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
```

12.1.3 Deploying the Custom Authentication Service Provider

Deploy the custom authentication Service Provider as follows.

12.1.3.1 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 and Social 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 and Social 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 and Social 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 and Social 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.

12.2 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

12.2.1 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.

12.2.2 Building the Custom Identity Service Provider

Build the custom Identity Service Provider as follows.

12.2.2.1 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
   ```

12.2.3 Deploying the Custom Identity Service Provider

Deploy the custom authentication Service Provider as follows. The following steps use *XYZProvider* as an example.

12.2.3.1 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.

13

Customizing Oracle Mobile Authenticator

The Oracle Mobile Authenticator is a mobile device app that uses Time-based One Time Password (TOTP) and push notifications to authenticate users. The Oracle Mobile Authenticator mobile device app is customer-facing and thus can be customized to represent your company.

This chapter describes procedures that can be used to brand the Oracle Mobile Authenticator with your company's logo and colors. It contains the following sections.

13.1 Understanding the Oracle Mobile Authenticator

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

13.2 Customizing Oracle Mobile Authenticator on iOS

The Oracle Mobile Authenticator (OMA) is distributed as a ZIP archive which contains OMA (as a framework), OMA resources bundle and strings files. Developers can use Xcode IDE to customize the OMA. This section contain information on how to do this. The following resources are required to customize OMA.

- `oamms_sdk_for_ios.zip` is the Identity Management Mobile SDK for iOS. It contains:
  - libIDMMobileSDK.a
  - Public Headers
  - Public Resources
- `OMACustomizable-11_1_2_3_0.zip` contains the following customizable OMA files:
  - OMALibrary.framework

- OMAResources.bundle
- Localization files

> **Note:** The `ofm_oma_clients_11.1.2.3.0.zip` contains the `OMACustomizable-11_1_2_3_0.zip` and `OracleMobileAuthenticator-11_1_2_3_0.apk` files. The latter is used in Section 13.3.1, "Using apktool."

The following sections contain more information.

- Using Xcode
- Customizing Oracle Mobile Authenticator

13.2.1 Using Xcode

The minimum version required is Xcode 6 with iOS SDK 8.0.

1. Open Xcode.
2. Click on Create a new Xcode Project.
3. Under iOS select Application.
4. Choose Single View Application and click Next.
5. Enter values for the following fields.
   - Product Name: Acme Authenticator, for example
   - Organization Name: Acme, for example
   - Organization Identifier: This value is the same as the identifier defined in Apple Developer.
   - Language: Objective-C
   - Devices: Choose Universal/iPhone/iPad depending on the devices on which this customized version of OMA will execute.
6. Click Next and then Create.

   This will open a new window where the Acme Authenticator project will be displayed.
7. In the Project Navigator menu click on Acme Authenticator project.

   The Acme Authenticator.xcodeproj tab will show the Project and Targets.
8. Under Targets click Acme Authenticator.
9. Click Build Settings.
10. Under Linking find Other Linker Flags and add -ObjC -all_load as its value.
11. Under Acme Authenticator.xcodeproj tab click General.
12. Add the following frameworks and libraries
    - Security.framework
    - SystemConfiguration.framework
    - CoreLocation.framework
    - libsqlite3.dylib

13. Under Project Navigator click Acme Authenticator and choose Add files to Acme Authenticator.
14. Add libIDMMobileSDK.a, Public Headers, Public Resources, OMALibrary.framework, OMAResources.bundle, Localization files and directories.
15. Click on AppDelegate.h file
16. Import OMALibrary app delegate by using #import <OMALibrary/OAAppDelegate.h>
17. Replace @interface AppDelegate : UIResponder <UIApplicationDelegate> with

```
@interface AppDelegate : OAAppDelegate
```

18. Click on AppDelegate.m file and remove all the UIApplicationDelegate methods.
19. Under Supporting Files right click on Info.plist file and choose Open As Source Code
20. Under the dict tag add the following tags.

```
<key>CFBundleDisplayName</key>
<string>Acme Authenticator</string>
```

21. Distribute the customized app.

    The customized Xcode project can be used for distributing the Acme Authenticator by following the guideleines in the Apple App Distribution Guide available at https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html

13.2.2 Customizing Oracle Mobile Authenticator

The following sections contain information about what can be customized.

- Changing the Application Art
- Modifying the Application Name and Text
- Toggling Online and Offline Mode
- Changing the Application Version
- Signing the Application

13.2.2.1 Changing the Application Art

Artwork used inside OMA is located in the OMAResources.bundle folder. These art files can be replaced with files of the same name. Table 13–1 contains a listing of the files. An app icon can be chosen by following the Technical Q&A QA1686 : App Icons on iPad and iPhone available at https://developer.apple.com/library/ios/qa/qa1686/_index.html

Table 13–1 Customizable Artwork

File Name	File Size	Description
check_57.fw.png	57x57 png file	Notification history screen when a notification is accepted
copy.png	57x57 png file	One-time password screen for copying OTP

Table 13–1 (Cont.) Customizable Artwork

File Name	File Size	Description
cross_57.fw.png	57x57 png file	Notification history screen when a notification was rejected
delete.png	57x57 png file	One-time password screen for deleting OTP account
edit.png	57x57 png file	One-time password screen for editing OTP account
gears_60.png	60x60 png file	Current configurations screen header
keyboard.png	57x57 png file	Add account screen and Offline configuration screen for offline account creation
notifications_57.png	57x57 png file	Notification prompt and history screen header
keyboard.png	57x57 png file	Add account screen and Online configuration screen for online account creation

13.2.2.2 Modifying the Application Name and Text

The app name can be changed by updating the value of the CFBundleDisplayName tag in the Info.plist file. The other text used in the app is pulled from the following files available under the Localization folder. This text can also be modified.

- help.html: Help file text
- privacy.html: Privacy policy text
- eula.txt: End user license agreement
- OALocalizable.strings: Messages shown in the app

13.2.2.3 Toggling Online and Offline Mode

The OMA supports both online and offline mode. This feature can be enabled or disabled by modifying the OMAResources.bundle/OAProperties.plist file.

13.2.2.4 Changing the Application Version

The Application Version can be changed by updating the CFBundleShortVersionString value in Info.plist file.

13.2.2.5 Signing the Application

App can be signed by following the instructions in the Apple App Distribution Guide available at
https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html

13.3 Customizing Oracle Mobile Authenticator on Android

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

- Using apktool
- Customizing Options

13.3.1 Using apktool

The `apktool` installation and usage guide can be accessed from the apktool project home at `https://code.google.com/p/android-apktool/`. The following sample command is used to decompile an Android app package.

```
apktool d "..\bin\OracleMobileAuthenticator-11_1_2_3_0.apk" -o d:\oma_smali_out
```

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

```
apktool b -f -a "..\Android_SDK\build-tools\20.0.0\aapt.exe"
  ..\oma_smali_out -o ..\oma_recompiled\temp.apk
```

Note: The `ofm_oma_clients_11.1.2.3.0.zip` contains the `OMACustomizable-11_1_2_3_0.zip` and `OracleMobileAuthenticator-11_1_2_3_0.apk` files. The former is used in Section 13.2, "Customizing Oracle Mobile Authenticator on iOS."

13.3.2 Customizing Options

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

- Changing Application Icons
- Modifying the Application Name and Text
- Toggling Online and Offline Mode
- Modifying the Version and Code Number
- Signing the Application

13.3.2.1 Changing Application Icons

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

Based on the requirement the OMA application icons can also be updated in the corresponding drawable folder. In order to customize the application icons replace the old icons with the new icons without changing the icon name. Table 13–2 describes the application icons that can be customized. Again, be sure not to change the Icon name.

Table 13–2 Customizable Application Icons

Application Icon and Description	Icon Name (Do Not Modify)
App Launcher / Oracle name with padlock	ic_launcher.png
Icon to add more accounts / plus sign	add.png
Icon to initiate bar code scanning / generic barcode	barcode.png
Icon for showing notification as accepted / check mark	check.png
Icon for showing notification as canceled / x mark	cross.png
Icon for delete account / trash can	delete.png

Table 13–2 (Cont.) Customizable Application Icons

Application Icon and Description	Icon Name (Do Not Modify)
Icon for showing error alert messages / exclamation mark	error_alert.png
Icon for copy OTP (in action bar) / two paper images	ic_action_copy.png
Icon for edit account / pencil image	ic_action_edit.png
Icon to show keyboard / keyboard image	keyboard.png
Icon to show notification / globe with text balloon	notification.png
Icon for settings / generic gears image	setting.png
Icone for sign-in / generic person image	signin.png

13.3.2.2 Modifying the Application Name and Text

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

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

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

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

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

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

13.3.2.3 Toggling Online and Offline Mode

The Oracle Mobile Authenticator supports both online and offline mode. This feature can be enabled or disabled by modifying the /res/raw/prop.txt file. For example, to support only offline mode the content of the `prop.txt` file is defined as in Example 13–1.

Example 13–1 Customizing Oracle Mobile Authenticator Mode

```
{
"configuration":
{
"online":"no",
"offline":"yes"
}
}
```

13.3.2.4 Modifying the Version and Code Number

Modify the version and code number of the application by changing details in the `apktool.yml` located in the directory where the .apk file content has been de-compiled. (See "Using apktool.") The apktool.yml file can be viewed and modified in any text editor. The versionCode and versionName parameters are located under the

versionInfo property as illustrated in Example 13–2. In this example, the version name has been changed to test.xx.x.x from the default value 11.1.2.3.0.

Example 13–2 Changing the Android Version and Code Number

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

13.3.2.5 Signing the Application

Android requires that all apps be digitally signed before they can be installed. Android uses the certificate to identify the author of the app. The certificate does not need to be signed by a certificate authority so Android apps often use self-signed certificates. Additional details on this Android requirement and its process, including the procedure you can use to sign your apps, are described at http://developer.android.com/tools/publishing/app-signing.html#signing-manually

14

Using the Mobile and Social REST API

This chapter describes the Oracle Access Management Mobile and Social REST API. This chapter includes the following topics:

Notes About Using cURL

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

Using cURL to send REST calls to the server can help you better understand how the 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 the JWTAuthentication or the 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-CREDENTIAL

Use to supply the extra credential required for token exchange.
"X-Idaas-Rest-Subject-Type" has to be specified as `TOKEN` in this case.

Where to use This Attribute

- Query parameter
- JSON body

Attribute Type

- Request only

Sample cURL Command

```
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-CREDENTIAL":"12345",...}'
```

Sample Request

```
curl -H "Content-Type: application/json" --request POST
http://host.example.com:14100/oic_rest/rest/jwtoamauthentication/authenticate
-d
'{"X-Idaas-Rest-New-Token-Type-To-Create":["USERTOKEN::OAMUT","USERTOKEN::OAMMT"],
  "X-Idaas-Rest-Subject-Value":"JWT-Token-Value",
  "X-Idaas-Rest-Subject-Type":"TOKEN",
  "X-Idaas-Rest-Subject-CREDENTIAL":"12345"}'
```

Comments

Use this attribute value in a token exchange use case, for example `jwtoamauthentication` or `mobilejwtoamauthentication`. It is not used otherwise.

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 a 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...Gk5A%3D%3D",
     "X-Idaas-Rest-Application-Context":"75sSbBZZKJiUOAWikZxsKA==",
     "X-Idaas-Rest-Application-Resource":"http://h5.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...5XFSQk5A%3D%3D",
     "X-Idaas-Rest-Application-Context":"75sSbBZZKJiUOAWikZxsKA==",
     "X-Idaas-Rest-Application-Resource":"http://h5.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 Service Domain name can be specified as a query parameter or a header using X-IDAAS-SERVICEDOMAIN. Otherwise, Mobile and Social assumes the default Service Domain.
- 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 Service Domain name can be specified as a query parameter or a header using X-IDAAS-SERVICEDOMAIN. Otherwise, Mobile and Social assumes the default Service Domain.
- Note that the token value in the query param is URL-encoded, but the same value in the header is not.
- The Application Profile has to be defined with a unique name that cannot be applied to any other authentication service. For example:

  ```
  <ApplicationProfile description="OIC Client 5" name="profileid3">
  </ApplicationProfile>
  ```

Mobile and Social 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
- Authentication for Multiple Tokens
- Get or Validate a (Client) Token
- Delete a Token
- Authorization

The following calls are valid when used with the JWT-OAM Authentication Service Provider:

- Create a JWT User Token
- Create a JWT User Token, OAM User Token, and OAM Master Token
- Exchanging a JWT Token for OAM Tokens
- Testing the JWT-OAM + PIN Token Service Provider (Mobile Case)
- Testing the JWT-OAM + PIN Token Service Provider (Desktop Case)
- Create an OAM Access Token Using an OAM User Token
- Validate a JWT USER TOKEN
- Validate an OAM USER TOKEN
- Delete an OAM USER TOKEN

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 Service Domain name can be specified as a query parameter or a header using X-IDAAS-SERVICEDOMAIN. Otherwise, Mobile and Social assumes the default Service Domain.

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 Service Domain name can be specified as a query parameter or a header using X-IDAAS-SERVICEDOMAIN. Otherwise, Mobile and Social assumes the default Service Domain.

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":"vTBI8jN8eYsmHCU..5XFSQA%3D%3D",
     "X-Idaas-Rest-Application-Context":"75sSbBZZKJiUOAWikZxsKA==",
     "X-Idaas-Rest-Application-Resource":"http:/wgte2.example.com:779/index.html",
     "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 Service Domain name can be specified as a query parameter or a header using X-IDAAS-SERVICEDOMAIN. Otherwise, Mobile and Social assumes the default Service Domain.

Authentication for Multiple Tokens

Shows how to send a REST call requesting multiple tokens, for example a User Token and a Master Token.

cURL Command

```
curl -i -H
"Content-Type: application/json"
--request POST http://host12.example.com:1801/idaas_
rest/rest/jwtauthentication/authenticate
-d '{
     "X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
     "X-Idaas-Rest-Subject-Username":"testuser",
     "X-Idaas-Rest-Subject-Password":"userpassword",
     "X-Idaas-Rest-New-Token-Type-To-Create":["USERTOKEN","USERTOKEN::OAMMT"]}'
```

Expected Output

```
{"TokensList":
     [
      {
          "X-Idaas-Rest-Token-Value":"eyJhbGciOiJSUzx...GbC7cswpZN1ep8up3E34",
          "X-Idaas-Rest-Token-Type":"USERTOKEN",
          "X-Idaas-Rest-User-Principal":"testuser",
          "X-Idaas-Rest-Provider-Type":"JWT",
          "handles":
               {"DebugDummyHandleName1":
                 {"expirationTSInSec":1332192041,"value":"DebugDummyHandleValue1"}
               }
      },
      {"X-Idaas-Rest-Token-Type":"USERTOKEN::OAMMT"}
     ]
}
```

Comments

- You can specify the Mobile and Social Token Type by using the `X-Idaas-Rest-New-Token-Type-To-Create` parameter. Must be one of the following:
  - `CLIENTTOKEN`
  - `USERTOKEN`
  - `ACCESSTOKEN`
  - `USERTOKEN::OAMMT`
- If the authentication service provider can issue a Master Token, the client will get two tokens: a User Token and the Master Token.

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 Service Domain name can be specified as a query parameter or a header using X-IDAAS-SERVICEDOMAIN. Otherwise, Mobile and Social assumes the default Service Domain.
- Note that the token value in the query param is URL-encoded, but the same value in the header is not.

Delete a Token

Shows how to send the REST call required to delete a token.

cURL Command

Sample Request 1

```
curl -H "Content-Type:application/json"
--request DELETE
http://localhost:18001/oic_rest/rest/jwtauthentication/tokens/info
-d '{
     "X-Idaas-Rest-Subject-Value":"YHEGjRP5eewNeXeK9v%2F3YBX...tvMJW9p%3D",
     "X-Idaas-Rest-Subject-Type":"TOKEN"}'
```

Sample Request 2

```
curl -H "Content-Type: application/json"
--request DELETE
http://localhost:14100/oic_rest/rest/oamauthentication/tokens
-d '{
     "X-Idaas-Rest-Subject-Value":"jdoe",
     "X-Idaas-Rest-Subject-Type":"UID"}'
-H "Authorization: 01wIWzki0cF0Z...6hwVYV4fZ2CAMSXZHKPKD8="
```

Expected Output

HTTP Status: 204 No Content

Comments

- You can use `X-Idaas-Rest-Subject-Type` to specify either `TOKEN` or `UID`. Use `X-Idaas-Rest-Subject-Value` to specify either the token or UID value.
- To delete a single token when the subject type is TOKEN, use either the service endpoint `~/tokens/info` or `~/delete`.
- To delete all the tokens belonging to the token owner when the subject type is TOKEN, use the service endpoint `~/tokens`. Use the `-H "AUTHORIZATION User Token Value"` header to validate the request.
- If the subject type is UID, use the service endpoint `~/tokens` to delete all the tokens belong to theUID. Use the `-H "AUTHORIZATION User Token Value"` header to validate the request.
- A Service Domain name can be specified as a query parameter or a header using `X-IDAAS-SERVICEDOMAIN`. Otherwise, Mobile and Social assumes the default Service Domain.
- Note that the token value in the query param is URL-encoded, but the same value in the header is not.

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 Service Domain name can be specified as a query parameter or a header using `X-IDAAS-SERVICEDOMAIN`. Otherwise, Mobile and Social assumes the default Service Domain.
- Note that the token value in the query param is URL-encoded, but the same value in the header is not.

Create a JWT User Token

The following call is valid when used with the JWT-OAM Authentication Service Provider.

cURL Command

```
curl -H "Content-Type: application/json"
--request POST http://host:port/oic_rest/rest/jwtoamauthentication/authenticate
-d '{
     "X-Idaas-Rest-New-Token-Type-To-Create":"USERTOKEN",
     "X-Idaas-Rest-Subject-Password":"password555",
     "X-Idaas-Rest-Subject-Username":"webuser1234",
     "X-Idaas-Rest-Subject-Type":"USERCREDENTIAL"}'
```

Expected Output

```
{
"X-Idaas-Rest-Token-Value":"eyJhbGciOiJ...YLSmUkto",
"X-Idaas-Rest-User-Principal":"weblogic",
"X-Idaas-Rest-Token-Type":"USERTOKEN",
"X-Idaas-Rest-Provider-Type":"JWT"}
```

Create a JWT User Token, OAM User Token, and OAM Master Token

The following calls are valid when used with the JWT-OAM Authentication Service Provider.

cURL Command

JWT-OAM Authentication Service Provider

```
curl -H "Content-Type: application/json"
--request POST http://host:port/oic_rest/rest/jwtoamauthentication/authenticate
-d '{
     "X-Idaas-Rest-New-Token-Type-To-Create":["USERTOKEN","USERTOKEN::OAMMT",
"USERTOKEN::OAMUT"],
     "X-Idaas-Rest-Subject-Password":"password555",
     "X-Idaas-Rest-Subject-Username":"webuser1234",
     "X-Idaas-Rest-Subject-Type":"USERCREDENTIAL"}'
```

Mobile JWT-OAM Authentication Service Provider

```
curl -H "Content-Type: application/json"
--request
POST http://host:port/oic_rest/rest/mobilejwtoamauthentication/authenticate
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-Idaas-Rest-Authorization: UIDPASSWORD cred="T0lDU1NPQ...WZHQ0RnPQ==="'
-d '{
     "X-Idaas-Rest-New-Token-Type-To-Create":["USERTOKEN","USERTOKEN::OAMUT",
"USERTOKEN::OAMMT"],
     "X-Idaas-Rest-Subject-Password":"password555",
     "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":"weblogic",
     "X-Idaas-Rest-Subject-Type":"USERCREDENTIAL"}'
```

Expected Output

JWT-OAM Authentication Service Provider

```
{
"TokensList":[
     {
      "X-Idaas-Rest-Token-Value":"eyJhbGciOiJSUz...FfxrkN9xM",
```

```
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN",
      "X-Idaas-Rest-Provider-Type":"JWT"
     },
     {
      "X-Idaas-Rest-Token-Value":"cL9fR2ASSB...iTaNs8c=",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN::OAMMT",
      "X-Idaas-Rest-Provider-Type":"OAM_11G"
     },
     {
      "X-Idaas-Rest-Token-Value":"VERSION_4%7EAn29pwsWv...ZMwLw%3D%3D",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN::OAMUT",
      "X-Idaas-Rest-Provider-Type":"OAM_11G"
     }]
}
```

Mobile JWT OAM Authentication Service Provider

```
{
"TokensList":[
     {
      "X-Idaas-Rest-Token-Value":"eyJhbGciOiJ...lxizU",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN",
      "X-Idaas-Rest-Provider-Type":"JWT"
     },
     {
      "X-Idaas-Rest-Token-Value":"0fY4apw0Cfw...+edij0M=",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN::OAMUT",
      "X-Idaas-Rest-Provider-Type":"OAM_11G"
     },
     {
      "X-Idaas-Rest-Token-Value":"VERSION_4%7EBSTnEU5eDhsK%2FS%...mt5j4w%3D%3D",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN::OAMMT",
      "X-Idaas-Rest-Provider-Type":"OAM_11G"
     }]
}
```

Exchanging a JWT Token for OAM Tokens

The following calls are valid when used with the JWT-OAM Authentication Service Provider.

The token exchange input here is a JWT User Token and the token exchange output is an OAM User Token and an OAM Master Token.

cURL Command

JWT-OAM Authentication Service Provider

```
curl -H "Content-Type: application/json"
--request POST http://host:port/oic_rest/rest/jwtoamauthentication/authenticate
-d '{
     "X-Idaas-Rest-New-Token-Type-To-Create":["USERTOKEN","USERTOKEN::OAMUT"],
     "X-Idaas-Rest-Subject-Value":"<JWT USER TOKEN>",
     "X-Idaas-Rest-Subject-Type":"TOKEN"}'
```

Note - You can also use the following for X-Idaas-Rest-New-Token-Type-To-Create:

```
"X-Idaas-Rest-New-Token-Type-To-Create":["USERTOKEN::JWTUT","USERTOKEN::OAMUT"]
```

Mobile JWT-OAM Authentication Service Provider

```
curl -H "Content-Type: application/json"
--request POST http://host:port/oic_
rest/rest/mobilejwtoamauthentication/authenticate
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-Idaas-Rest-Authorization: UIDPASSWORD cred="<BASE 64 Encoding Client ID :
CRH>"'
-d '{
     "X-Idaas-Rest-New-Token-Type-To-Create":["USERTOKEN","USERTOKEN::OAMUT",
"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"
          },
     "X-Idaas-Rest-Subject-Value":"<JWT USERTOKEN>",
     "X-Idaas-Rest-Subject-Type":"TOKEN"}'
```

Expected Output

JWT-OAM Authentication Service Provider


```
{
 "TokensList":[
     {
      "X-Idaas-Rest-Token-Value":"eyJhbGciOiJSU...o6JOao3s",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN",
      "X-Idaas-Rest-Provider-Type":"JWT"
     },
     {
      "X-Idaas-Rest-Token-Value":"ipZ45ey55BAkb...G0tuDGyfdY=",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN::OAMUT",
      "X-Idaas-Rest-Provider-Type":"OAM_11G"
     },
     {
      "X-Idaas-Rest-Token-Value":"VERSION_4%7ESTFLB3gGSZrdy6...2SbdLQ%3D%3D",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN::OAMMT",
      "X-Idaas-Rest-Provider-Type":"OAM_11G"
     }]
}
```


Mobile JWT OAM Authentication Service Provider


```
{
 "TokensList":[
     {
      "X-Idaas-Rest-Token-Value":"eyJhbGciOiJ...-mVLKLtpONChYs",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN",
      "X-Idaas-Rest-Provider-Type":"JWT"
     },
     {
      "X-Idaas-Rest-Token-Value":"BsL1V2s...nbGXUF4nPfHFPqs=",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN::OAMUT",
      "X-Idaas-Rest-Provider-Type":"OAM_11G"
     },
     {
      "X-Idaas-Rest-Token-Value":"VERSION_4%7E3Bbc0YHd4upKZfjt3...M6ZORc3Q%3D%3D",
      "X-Idaas-Rest-User-Principal":"weblogic",
      "X-Idaas-Rest-Token-Type":"USERTOKEN::OAMMT",
      "X-Idaas-Rest-Provider-Type":"OAM_11G"
     }]
}
```

Testing the JWT-OAM + PIN Token Service Provider (Mobile Case)

For more information, see "Configuring OAM to use the JWT-OAM + PIN Token Service Provider" in the *Administrator's Guide for Oracle Access Management*.

Get AppProfile Data from OAMMS

The following sample command tests accessing the Mobile and Social server and getting the AppProfile for mobileapp1. This step is only required for mobile use cases. The URL follows this format:

```
http://oamms-host:port/oic_rest/rest/AppProfiles/mobileapp1?serviceDomain=
MobileServiceDomain&osType=iPhone%20OS&osVer=4.0&clientSDKVersion=11.1.2.0.0
```

The following sample response contains the AppProfile data:


```
{
   {
      "CRHDelivery": "HTTP",
      "SSOConfig": [
      {
         "mobileapp1":
         {
            "AndroidAppSignature": null,
            "AndroidPackage": null,
            "IOSBundleID": null,
            "SSOInclusion": false,
            "SSOPriority": -1,
            "URLScheme": null
         }
      }],
      "accessService": "/oic_rest/rest/mobilejwtoamauthentication/access",
      "clientId": "mobileapp1",
      "deleteService": "/oic_rest/rest/mobilejwtoamauthentication/delete",
      "jailBreakingDetectionPolicy":
      {
         "autoCheckPeriodInMin": 60,
         "clientSDKVersion": "11.1.2.0.0",
         "detectionLocation": [
         {
            "action": "exists",
            "filePath": "/bin/bash",
            "success": true
         },
         {
            "action": "exists",
            "filePath": "/Applications/Cydia.app",
            "success": true
         },
         {
            "action": "exists",
            "filePath": "/Applications/limera1n.app",
            "success": true
         },
         {
            "action": "exists",
            "filePath": "/Applications/greenpois0n.app",
            "success": true
         },
         {
```

```
            "action": "exists",
            "filePath": "/Applications/blackra1n.app",
            "success": true
         },
         {
            "action": "exists",
            "filePath": "/Applications/blacksn0w.app",
            "success": true
         },
         {
            "action": "exists",
            "filePath": "/Applications/redsn0w.app",
            "success": true
         },
         {
            "action": "exists",
            "filePath": "/Applications/sn0wbreeze.app",
            "success": true
         }],
         "osType": "iPhone OS",
         "osVer": "4.0",
         "policyExpirationInSec": 3600
      },
      "mobileAppConfig":
      {
         "AllowOfflineAuthentication": "false",
         "AndroidPackage": null,
         "AuthenticationRetryCount": "3",
         "ClaimAttributes": null,
         "IOSBundleID": null,
         "ProfileCacheDuration": "60",
         "RPWebView": "Embedded"
      },
      "mobileAuthStyle": "MOBILESERVICEAUTH",
      "mobileCredLevelForRegApp": "USERTOKEN",
      "registerService": "/oic_rest/rest/mobilejwtoamauthentication/register",
      "rpLoginPage": "/oic_rp/login.jsp",
      "serviceDomain": "MobileServiceDomain",
      "userAuthnConfig": "JWT_UT+PIN",
      "UserAuthenticationOutput": "USERTOKEN::JWTUT",
      "TokenExchangeOutput": "USERTOKEN::OAMUT,USERTOKEN::OAMMT"
   }, "userAuthnService": "/oic_
rest/rest/mobilejwtoamauthentication/authenticate",
   "userProfileService": "/oic_rest/rest/userprofile",
   "validateService": "/oic_rest/rest/mobilejwtoamauthentication/validate"
}
```


Generating the CRED Value for the Authorization Header

The following format should be used to generate the CRED value for the authorization header:

```
CRED = base64{appprofile-name:clientRegHandle}
```

For example:

```
base64{mobileapp1:eyJvcmFjbGU6aWRtOmNsYWltczpjbGllbnQ6c2Rr...}
```

Authenticate the User and get the JWT User Token

The following sample mobile command tests getting a JWT user token upon authenticating the user.

```
curl -H "Content-Type: application/json"
--request POST
http://oamms-host:portnumber/oic_rest/rest/mobilejwtoamauthentication/authenticate
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-Idaas-Rest-Authorization: UIDPASSWORD
cred="bW9iaWxlYXBwMTpleUp2Y21GamJHVTZhV1....x2UT0="'
-d '{
     "X-Idaas-Rest-New-Token-Type-To-Create":"USERTOKEN::JWTUT",
     "X-Idaas-Rest-Subject-Password":"password123",
     "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":"weblogic",
     "handles":{},
     "X-Idaas-Rest-Subject-Type":"USERCREDENTIAL"
   }'
```

The response should be similar to the following:

```
{
   "X-Idaas-Rest-Token-Value": "eyJhbGciOiJSUzUnR.....JeaVZGIP_YjMuPrgt82XXk0E",
   "X-Idaas-Rest-User-Principal": "weblogic",
   "X-Idaas-Rest-Provider-Type": "JWT",
   "X-Idaas-Rest-Token-Type": "USERTOKEN"
}
```

Token Exchange

The following sample command (for mobile) tests getting an OAM user token and OAM master token by exchanging a JWT user token and including the cred (or PIN) value.

```
curl - H "Content-Type: application/json"
--request POST
http: //oamms-host:portnumber/oic_
rest/rest/mobilejwtoamauthentication/authenticate
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-Idaas-Rest-Authorization: UIDPASSWORD
cred="bW9iaWxlYXBwMTpleU...hEdWtySHVGRnBkdkFORXZxMmx2UT0="'
- d '{
```

```
        "X-Idaas-Rest-New-Token-Type-To-Create": ["USERTOKEN::OAMUT",
"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"
        },
        "X-Idaas-Rest-Subject-Value": "eyJhbGciOiJSUzUxMp0aSI6...ZGIP_
YjMuPrgt82XXk0E",
        "X-Idaas-Rest-Subject-Type": "TOKEN",
        "X-Idaas-Rest-Subject-Credential": "12345"
     }
    '
```

The response should be similar to the following:

```
{
   "TokensList": [
   {
      "X-Idaas-Rest-Token-Value":
"NIpXEwPuE0TLy2tM2WkKb/ZAg9k/uwgPx...kMsM4F+Vhv",
      "X-Idaas-Rest-User-Principal": "weblogic",
      "X-Idaas-Rest-Provider-Type": "OAM_11G",
      "X-Idaas-Rest-Token-Type": "USERTOKEN::OAMUT"
    },
   {
      "X-Idaas-Rest-Token-Value": "VERSION_
4%7EctnTQTYMFtnxPhei8IJu...h1Pg1lh6bFw",
      "X-Idaas-Rest-User-Principal": "weblogic",
      "X-Idaas-Rest-Provider-Type": "OAM_11G",
      "X-Idaas-Rest-Token-Type": "USERTOKEN::OAMMT"
    }]
}
```

Testing the JWT-OAM + PIN Token Service Provider (Desktop Case)

For more information, see "Configuring OAM to use the JWT-OAM + PIN Token Service Provider" in the *Administrator's Guide for Oracle Access Management*.

Authenticate the User and get a JWT User Token

The following sample desktop command tests getting a JWT user token upon authenticating the user.

```
curl -H "Content-Type: application/json"
--request POST
http://oamms-host:portnumber/oic_rest/rest/jwtoamauthentication/authenticate
-d '
{
   "X-Idaas-Rest-New-Token-Type-To-Create": "USERTOKEN::JWTUT",
   "X-Idaas-Rest-Subject-Password": "password123",
   "X-Idaas-Rest-Subject-Username": "weblogic",
   "X-Idaas-Rest-Subject-Type": "USERCREDENTIAL"
}
'
```

The response should look like the following:

```
{
   "X-Idaas-Rest-Token-Value": "eyJhbGciOiJSUz...m-5_-tPpDeINkYlhfTkkrfnarUhp2R0",
   "X-Idaas-Rest-User-Principal": "weblogic",
   "X-Idaas-Rest-Provider-Type": "JWT",
   "X-Idaas-Rest-Token-Type": "USERTOKEN"
}
```

Token Exchange (Desktop)

The following sample desktop command tests getting an OAM user token and OAM master token by exchanging a JWT user token and including the cred (or PIN) value.

```
curl -H "Content-Type: application/json"
--request POST
http://oamms-host:portnumber/oic_rest/rest/jwtoamauthentication/authenticate
-d '
{
   "X-Idaas-Rest-New-Token-Type-To-Create": "USERTOKEN::OAMUT",
   "X-Idaas-Rest-Subject-Value": "eyJhbGciOiJSUz...m-5_
-tPpDeINkYlhfTkkrfnarUhp2R0"",
   "X-Idaas-Rest-Subject-Type": "TOKEN",
   "X-Idaas-Rest-Subject-Credential": "12345"
}
'
```

The response should look like the following:

```
{
   "X-Idaas-Rest-Token-Value": "KkraNeuiD6N+Jas2...WzaJSR/wzPy41Ekq/+NpGd/asagLl",
   "X-Idaas-Rest-User-Principal": "weblogic",
   "X-Idaas-Rest-Provider-Type": "OAM_11G",
   "X-Idaas-Rest-Token-Type": "USERTOKEN"
}
```

Create an OAM Access Token Using an OAM User Token

The following calls are valid when used with the JWT-OAM Authentication Service Provider.

cURL Command

JWT-OAM Authentication Service Provider

```
curl -H "Content-Type: application/json"
--request POST http://host:port/oic_rest/rest/jwtoamauthentication/access
-d '{
     "X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN",
     "X-Idaas-Rest-Application-Resource":"http:\/\/a6.example.com:7777\/idx.html",
     "X-Idaas-Rest-Subject-Value":"<OAM USER TOKEN>",
     "X-Idaas-Rest-Application-Context":"encquery%3DA1%2BnxGqYJxtmcteKYUux%2F7%2FV
aRBrBVRByRl81YM89Rv1940CTWlcddShowo2r516MLCa%2BHcPjgGNDeGVSagzGmV84GKybdiFtzrwd8ms
i9nRr4ijlW7%2BznCmb6C5xYiEXg6RBpI1Eud9Ce2VjNyrYY%2F3Ig7ntdhbF1NbznmV%2BwGf9S6ogxKR
abbKl2yOD5NO%2FC7NkmJOoDSisQb9IR9DnUxm1uBfHkKpE34RAyvpqg4xeGx2r%2Fuo0F0upeZ8KbsT%2
FugszrOdPR0S5O9%2BbPzV%2BNfzuFH25M0qriKbVj9EixNb0gzSEf2bCBmP9tXbWXDdG%20agentid%3D
adc2171186_11gwebgateprofile%20ver%3D1",
     "X-Idaas-Rest-Subject-Type":"TOKEN"}'
```

Mobile JWT-OAM Authentication Service Provider

```
curl -H "Content-Type: application/json"
--request POST http://host:port/oic_rest/rest/mobilejwtoamauthentication/access
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-Idaas-Rest-Authorization: UIDPASSWORD cred="BASE 64 Encoding
(ClientID:CRH)"'
-d '{
     "X-Idaas-Rest-New-Token-Type-To-Create":"ACCESSTOKEN",
     "X-Idaas-Rest-Application-Resource":"http:\/\/6.example.com:7777\/idx.html",
     "X-Idaas-Rest-Subject-Value":"<OAM USER TOKEN>",
     "X-Idaas-Rest-Application-Context":"encquery%3DNgSQHsqHQeDgTwiOnZCqB3io74D2c
VJjuw0lf1LhvS%2F1L29aOBFehYXHFB%2Bhfd4XNHt21pqFLC5HdA%2Fi0ScENG3Tq7YK3Uv2yde1tCec
ojHmryb8zpTriUex3kYg83VRzg1gBmIJnTVpiCVgaVlBhe3mKE7liqYcJXmsXFudsjUn%2FcUuXuWdWXP
Qzi1d3WJ3lwdq0DPRnXUFGg%2BzsO%2BarKcreIg3BmGsmxZE7lLL6b9Wf9jbhOwlk1wsq2nqdFPDDS3O
Yz3T9o9ZtsO1xnKuHsLwoMaNtM%2FSIjxpcmrntyQw2w7i8NWxnVP7w1RJDvu7%20agentid%3Dadc217
1186_11gwebgateprofile%20ver%3D1",
     "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-Type":"TOKEN"}'
```

Expected Output

JWT-OAM Authentication Service Provider

```
{
 "X-Idaas-Rest-Token-Value":"lroMQ%2Bwj7Ji4daRdXfGb8%2FG...AzWPTM%3D",
 "X-Idaas-Rest-User-Principal":"weblogic",
 "X-Idaas-Rest-Token-Type":"ACCESSTOKEN",
 "X-Idaas-Rest-Provider-Type":"OAM_11G"
}
```

Mobile JWT OAM Authentication Service Provider

```
{
 "X-Idaas-Rest-Token-Value":"xGhOiD%2FLVrnyU...nYgo%3D",
 "X-Idaas-Rest-User-Principal":"weblogic",
 "X-Idaas-Rest-Token-Type":"ACCESSTOKEN",
 "X-Idaas-Rest-Provider-Type":"OAM_11G"
}
```

Validate a JWT USER TOKEN

The following calls are valid when used with the JWT-OAM Authentication Service Provider.

cURL Command

JWT-OAM Authentication Service Provider

```
curl -i --request GET
"http://host:port/oic_rest/rest/jwtoamauthentication/tokens/info?X-Idaas-Rest-
Subject-Value=<JWT USER TOKEN>&X-Idaas-Rest-Subject-Type=TOKEN"
```

Mobile JWT-OAM Authentication Service Provider

```
curl -i --request GET
"http://host:port/oic_
rest/rest/mobilejwtoamauthentication/tokens/info?X-Idaas-Rest-Subject-Value=<JWT
USER TOKEN>&X-Idaas-Rest-Subject-Type=TOKEN"
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-Idaas-Rest-Authorization: UIDPASSWORD
cred=" BASE 64 Encoding(CLIENTID:CRH) "'
```

Expected Output

JWT-OAM Authentication Service Provider

```
{
"X-Idaas-Rest-Token-Value":"eyJhbGciO...fsOn3BIo6JOao3s",
"X-Idaas-Rest-User-Principal":"weblogic",
"X-Idaas-Rest-Token-Type":"USERTOKEN",
"X-Idaas-Rest-Provider-Type":"JWT"}
```

Mobile JWT OAM Authentication Service Provider

```
{
"X-Idaas-Rest-Token-Value":"eyJhbGci....mVLKLtpONChYs",
"X-Idaas-Rest-User-Principal":"weblogic",
"X-Idaas-Rest-Token-Type":"USERTOKEN",
"X-Idaas-Rest-Provider-Type":"JWT"}
```

Validate an OAM USER TOKEN

The following calls are valid when used with the JWT-OAM Authentication Service Provider.

cURL Command

JWT-OAM Authentication Service Provider

```
curl -i --request GET
"http://host:port/oic_rest/rest/jwtoamauthentication/tokens/info?X-Idaas-Rest-
Subject-Value=<OAM USER TOKEN>&X-Idaas-Rest-Subject-Type=TOKEN"
```

Mobile JWT-OAM Authentication Service Provider

```
curl -i --request GET
"http://host:port/oic_
rest/rest/mobilejwtoamauthentication/tokens/info?X-Idaas-Rest-
Subject-Value=<OAM** USER TOKEN>&X-Idaas-Rest-Subject-Type=TOKEN"
-H 'X-IDAAS-SERVICEDOMAIN:MobileServiceDomain'
-H 'X-Idaas-Rest-Authorization: UIDPASSWORD cred=" BASE 64 Encoding(CLIENTID:CRH)
"'
```

Expected Output

JWT-OAM Authentication Service Provider

```
{
"X-Idaas-Rest-Token-Value":"ipZ45ey55BAk...NqM3YsycmdG0tuDGyfdY=",
"X-Idaas-Rest-User-Principal":"weblogic",
"X-Idaas-Rest-Token-Type":"USERTOKEN",
"X-Idaas-Rest-Provider-Type":"OAM_11G"}
```

Mobile JWT OAM Authentication Service Provider

```
{
"X-Idaas-Rest-Token-Value":"8C2wieU9h7VfQM...UmubmxvJ+SpL5fLZYpbU=",
"X-Idaas-Rest-User-Principal":"weblogic",
"X-Idaas-Rest-Token-Type":"USERTOKEN",
"X-Idaas-Rest-Provider-Type":"OAM_11G"}
```

Delete an OAM USER TOKEN

The following calls are valid when used with the JWT-OAM Authentication Service Provider.

cURL Command

```
curl -H "Content-Type: application/json"
--request DELETE "http://host:port/oic_rest/rest/jwtoamauthentication/tokens/info"
-d '{
     "X-Idaas-Rest-Subject-Value":"<OAM USER TOKEN>",
     "X-Idaas-Rest-Subject-Type":"TOKEN"}'
```

Expected Output

```
HTTP Response 204
```

Mobile and Social 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 and Social 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:/wg12.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="..." '
--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 and Social 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 and Social 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
- Read a "members" Relationship
- Delete a "members" Relationship

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)

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.example.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":"password555",
     "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_23552f26e974030dc160...c363d47a01918caf2f97"},
            "oaam.session":
                 {
                  "expirationTSInSec":1350609323,
                  "value":"561_419dc5ee6b325535dd0...b73c74573a49dec233a"
                 },
            "oic.multiStepAuthnSessionHandle":
                 {
                  "expirationTSInSec":1350606623,
                  "value":"eyJvcmlnU2VjdXJpdHlFdlsiU..1hclb2FtYXdGljYXRpb24ifQ=="
                 }
           },
      "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.example.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":"password555","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"}}}
```

Specifying the Tenant Name in the Header

The client can specify the tenant name as shown in the following example:

```
curl -H "Content-Type: application/json" ****-H "MY-MT-NAME: sales"****
--request POST http://localhost:18001/oic_rest/rest/jwtauthentication/authenticate
-d '{
     "X-Idaas-Rest-Subject-Type":"USERCREDENTIAL",
     "X-Idaas-Rest-Subject-Username":"profileid3",
     "X-Idaas-Rest-Subject-Password":"clientpassword",
     "X-Idaas-Rest-New-Token-Type-To-Create":"CLIENTTOKEN"}'
```

Also see "Enabling the REST Client to Specfy the Tenant Name" in the *Administrator's Guide for Oracle Access Management*.

Error Messages

Mobile and Social REST API error messages are documented in the *Oracle Fusion Middleware Error Message Reference*. The "IDAAS" prefix designates Mobile and Social messages.

Part IV

Developing with the OAuth Service

This part discusses developing applications using the Oracle Access Management OAuth Service.

It includes the following chapters:

- Chapter 15, "Using the OAuth Services API"
- Chapter 16, "Customizing the OAuth Services"

15

Using the OAuth Services API

This chapter describes the Oracle Access Management OAuth Services API. This chapter includes the following topics:

- Using REST in Standard 3-Legged OAuth Services Flows
- Using REST in Standard 2-Legged OAuth Services Flows
- Getting Identity Tokens
- Validating an Access Token
- Performing Access Token Introspection
- Revoking an Access Token
- Administering a Secret Key
- Administering the OAuth Services User Profile Service with REST
- Administering OAuth Services Consent Management Services with REST
- Using REST in OAuth Services Mobile Client 3-Legged Flows
- Using REST in OAuth Services Mobile Client 2-Legged Flows
- Using Credentials, PIN and Assertions to Get Tokens

Notes About Using cURL

This chapter uses cURL to demonstrate the REST calls that the OAuth client sends to the Mobile and Social OAuth Services. cURL is free software that you can download from the cURL website at `http://curl.haxx.se/`

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

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

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

Available Java API References

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

Using REST in Standard 3-Legged OAuth Services Flows

This section documents the REST calls for the 3-legged OAuth Services flows. For more information, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

Sample Request

The following sample request has two parts:

- Part One: The Front-Channel Request - takes place between the resource owner (or end user) and the OAuth Services server.
- Part Two: The Back-Channel Request - takes place between the OAuth Services server and the client application.

Part One: The Front-Channel Request

The client application redirects the user (the owner of the resource being requested) to the OAuth Services server's authorization endpoint using a browser. The user needs to authenticate with OAuth Services and, optionally, authorize access to the requested resources by providing consent. Once the user interaction completes successfully, OAuth Services issues an authorization code which the client application then uses to request an Access Token as documented in Part Two: The Back-Channel Request.

Sample Authorization Code Request

```
curl - i
--request GET "https://host:port/ms_oauth/oauth2/endpoints/oauthservice/authorize?
response_type=code
&client_id=54321id
&redirect_uri=http://client.example.com/return
&scope=user_read
&state=xyz"
```

Table 15–1 Request Parameters

Name	Description	Required
`response_type`	Value must be `code` for this flow.	Required
`client_id`	A client identifier given by the authorization server. The authorization server validates the `client_id` value with the configuration (the client registry). If the value is invalid, an error response is sent to the user-agent.	Required
`redirect_uri`	The client app's redirect URI authorization code. If not sent, then the configuration/client registry is checked to see if a `redirect_uri` value is defined. Else, an error response is sent to the user-agent.	Optional
`scope`	Defines scope values in the configuration/scope registry. If no scope is sent, or if an invalid scope is specified, an error response is sent to the client app's `redirect_uri`. Use space-separated values.	Required
`state`	An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter should be used to prevent cross-site forgery requests.	Recommended

Sample Authorization Code Response

If the resource owner grants access, the OAuth Services server issues an authorization code and delivers it to the client by adding the applicable parameters to the query component of the redirection URI using the `application/x-www-form-urlencoded` format. The parameters are documented in Table 15–2.

```
https: //client.example.com/return?code=eyJhbG...rWWk8hbs_o6uY&state=xyz
```

Table 15–2 Response Parameters

Name	Description
code	Includes the following: ■ Expiry (15 minutes by default. To change this value, open the OAuth Service Profile Configuration page and update the **Expires** setting under **Token Settings**.) ■ Client_id ■ Redirect_uri
state	Same value specified in the authorization request. Only included if it was specified in the authorization request.

Sample Error Response

If validation errors are found, a JSON response containing error codes and descriptions is sent.

```
{"error_code":"invalid_client", "error_description":"client identifier invalid"}
```

The following list documents some error codes and their descriptions.

- server_error - runtime processing error
- invalid_scope - requested scope is invalid, unknown, or malformed
- invalid_redirect_uri - redirect URI does not match with client app
- access_denied - end-user denied authorization
- invalid_client - client identifier invalid

Part Two: The Back-Channel Request

This flow is between OAuth Services (the authorization server) and the client application. The sample shows how to exchange the authorization code for an Access Token.

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://hostname:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
    redirect_uri=http%3A%2F%2Fclient.example.com:17001%2Freturn
    &grant_type=authorization_code
    &code=eyJhbG...rWWk8hbs_o6uY
   '
```

The grant_type parameter value must be authorization_code, and the code parameter value must be the authorization code generated by the authorization endpoint. You must send the redirect_uri token if the redirect_uri parameter was included in the authorization request. The value must be the same.

Sample Response

```
{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "Bearer",
  "expires_in": 3600,
```

```
  "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA"
}
```

Using REST in Standard 2-Legged OAuth Services Flows

This section documents the REST calls for the 2-legged OAuth Services flows. It provides sample REST requests that show how to get a resource access token. When no resource is sent in the request, the resulting token can be used as an Identity Token. For more information, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

Sample Response

The following response is typical for the requests documented in this section.

> **Note:** The `refresh_token` element is included in the server response if a requested scope is designated as an offline scope.The `refresh_token` element is not sent if none of the scopes is offline.

```
HTTP/1.1 200 OK

Cache-Control: no-cache, no-store, must-revalidate

Date: Wed, 04 Dec 2013 21:52:03 GMT

Pragma: no-cache

Transfer-Encoding: chunked

Content-Type: application/json

X-ORACLE-DMS-ECID: 09edd9b26949554d:-1f8be51:142bf50a0dc:-8000-0000000000001b27

X-Powered-By: Servlet/2.5 JSP/2.1

{
  "expires_in":3600,
  "token_type":"Bearer",
  "access_token":"<access token value>",
  "refresh_token":"<refresh token value>"
}
```

Using Client Credentials

The following sample shows how to use client credentials to get an access token.

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://hostname:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
    grant_type=client_credentials
    &scope=scope1%20scope2
   '
```

Using the Resource Owner Credentials

The following sample shows a resource owner request that includes user ID and password credentials, as well as a client ID and secret in an HTTP Basic header.

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://hostname:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=password
    &username=userxyz
    &password=pwd123xyz
    &scope=scope1%20scope2'
```

Using a Refresh Token

The following sample shows using a refresh token with `clientid:clientsecret` in the basic authorization header.

```
curl -i
-H 'Authorization: Basic dGVzdDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://hostname:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=refresh_token
   &refresh_token=<refresh-token-value>'
```

This next example shows using the client assertion as a client credential.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=refresh_token
   &refresh_token=<refresh-token-value>
   &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer
   &client_assertion=<client-assertion-value>'
```

Using a SAML Client Assertion

The following sample shows a client credentials request that uses a SAML client assertion generated by a third party.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'client_id=54321id
    &grant_type=client_credentials
    &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Asaml2-bearer
    &client_assertion=<SAML client assertion value>
    &scope=scope1%20scope2'
```

Using a JWT Client Assertion

The following sample shows an authorization code request that uses a JWT client assertion generated by the IDM OAuth Server or a third party.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'client_id=54321id
   &grant_type=client_credentials
   &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
   &client_assertion=<JWT client assertion value>
   &scope=scope1%20scope2'
```

Using User ID/Password Credentials and ClientID+Secret in an HTTP Basic Header

The following sample shows a resource owner request that uses user ID and password credentials, plus a ClientID and secret in the HTTP Basic header.

```
curl -i
-H 'Authorization: Basic <base64encoded(clientID:Secret)>'
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
    grant_type=password
    &username=user123
    &password=password123
  '
```

Using User ID/Password Credentials and a JWT Client Assertion

The following sample shows a resource owner request that uses user ID and password credentials, and a JWT client assertion generated by a third party.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=password
    &username=userxyz
    &password=pwd123xyz
    &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
    &client_assertion=<JWT client assertion value>
    &scope=scope1%20scope2'
```

Using UserID/Password Credentials and a SAML Client Assertion

The following sample shows an authorization code request that uses user ID and password credentials, and a SAML client assertion generated by a third party.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=password
    &username=userAbc123
    &password=passwordAbc123
    &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Asaml2-bearer
    &client_assertion=<SAML client assertion value>
    &scope=scope1%20scope2'
```

Using a SAML User Assertion Credential and ClientID+Secret in an HTTP Basic Header

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer
    &assertion=<SAML user assertion value>'
    &scope=scope1%20scope2
```

Using a SAML User Assertion Credential and a SAML Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer
    &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Asaml2-bearer
    &client_assertion=<SAML client assertion value>
    &assertion=<SAML user assertion value>
    &scope=scope1%20scope2'
```

Using a SAML User Assertion Credential and a JWT Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer
    &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
    &client_assertion=<JWT client assertion value>
    &assertion=<SAML user assertion value>
    &scope=scope1%20scope2'
```

Using a JWT User Assertion Credential and ClientID+Secret in an HTTP Basic Header

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
    &assertion=<JWT user assertion value>
    &scope=scope1%20scope2'
```

Using a JWT User Assertion Credential and a SAML Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
    &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Asaml2-bearer
    &client_assertion=<SAML client assertion value>
    &assertion=<JWT user assertion value>
    &scope=scope1%20scope2'
```

Using a JWT User Assertion Credential and a JWT Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
    &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
    &client_assertion=<JWT client assertion value>
    &assertion=<JWT user assertion value>
    &scope=scope1%20scope2'
```

Getting Identity Tokens

This section demonstrates how to get an access token (that is, an identity token for client and user) from OAuth Services. It includes the following sections.

Getting a Client Identity Token

This section shows multiple ways to get a client identity token.

- Using Client Credentials
- Using a Third-Party Generated SAML Client Assertion
- Using a Third-Party Generated JWT Client Assertion

Using Client Credentials

This sample includes the ClientID+Secret in the HTTP Basic Auth header.

```
curl - i
- H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
- H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http: //host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=client_credentials'
```

Sample Response

```
{
  "oracle_client_assertion_type":
"urn:ietf:params:oauth:client-assertion-type:jwt-bearer",
  "expires_in": 604800,
  "token_type": "Bearer",
  "oracle_tk_context": "client_assertion",
  "access_token": "access token value" > ,
}
```

Using a Third-Party Generated SAML Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   client_id=54321id
   &grant_type=client_credentials
   &client_assertion_type=
urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Asaml2-bearer
   &client_assertion=<SAML client assertion value>
   '
```

Refer to the sample response in the first example.

Using a Third-Party Generated JWT Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   client_id=54321id
   &grant_type=client_credentials
   &client_assertion_type=
urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
   &client_assertion=<JWT client assertion value>
   '
```

Refer to the sample response in the first example.

Getting a User Identity Token

The samples in this section demonstrate how to get a user identity token, also referred to as an access token or user assertion. All of the requests receive a response similar to the following:

```
{
  "expires_in": 28800,
  "token_type": "Bearer",
  "oracle_tk_context": "user_assertion",
  "oracle_grant_type": "urn:ietf:params:oauth:grant-type:jwt-bearer",
  "access_token": "<access token value>"
}
```

The following sections contain the samples.

- Getting a User Identity Token With a User ID and Password and Varying Client Credentials
- Getting a User Identity Token With a SAML User Assertion Credential and Varying Client Credentials
- Getting a User Identity Token With a JWT User Assertion Credential and Varying Client Credentials

Getting a User Identity Token With a User ID and Password and Varying Client Credentials

This category has three samples.

Using UserID/Password Credentials and a ClientID+Secret in the HTTP Basic Header

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   grant_type=password
   &username=sampleuser
   &password=samplepassword
   '
```

Using UserID/Password Credentials and a Third-Party JWT Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   grant_type=password
   &username=sampleuser
   &password=samplepassword
   &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
   &client_assertion=<JWT client assertion value>
   '
```

Using UserID/Password Credentials and a Third-Party SAML Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   grant_type=password
```

```
   &username=sampleuser
   &password=samplepassword
   &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Asaml2-bearer
   &client_assertion=<SAML client assertion value>'
```

Getting a User Identity Token With a SAML User Assertion Credential and Varying Client Credentials

This category has three samples.

Using a Third-Party SAML User Assertion Credential and a ClientID+Secret in the HTTP Basic Header

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer
   &assertion=<SAML user assertion value>'
```

Using a Third-Party SAML User Assertion Credential and a SAML Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer
   &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Asaml2-bearer
   &client_assertion=<SAML client assertion value>
   &assertion=<SAML user assertion value>'
```

Using a Third-Party SAML User Assertion Credential and a JWT Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer
   &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
   &client_assertion=<JWT client assertion value>
   &assertion=<SAML user assertion value>'
```

Getting a User Identity Token With a JWT User Assertion Credential and Varying Client Credentials

This category has three samples.

Using a Third-Party JWT User Assertion Credential and a ClientID+Secret in the HTTP Basic Header

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
   &assertion=<JWT user assertion value>'
```

Using a Third-Party JWT User Assertion Credential and a SAML Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
   &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Asaml2-bearer
   &client_assertion=<SAML client assertion value>
   &assertion=<JWT user assertion value>'
```

Using a Third-Party JWT User Assertion Credential and a JWT Client Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
   grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
   &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
   &client_assertion=<JWT client assertion value>
   &assertion=<JWT user assertion value>'
```

Validating an Access Token

This section provides sample REST requests that show how to validate a resource access token. It includes the following examples:

- Using the Client ID and Secret in an HTTP Basic Header
- Using a Client Assertion

Using the Client ID and Secret in an HTTP Basic Header

The following sample shows an access token validation request that includes the client ID and secret in an HTTP Basic header. Note that the assertion value is not a normal string, but a JSON. The value of `assertion` is the access token.

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Fresource-access-token%2Fjwt
    &oracle_token_action=validate
    &scope=UserProfile.users
    &assertion=
eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsIng1dCI6Im51T0JSSjhOVDRNTEZYRVkwZEVZb3d2MjhrWS
IsImtpZCI6Im9yYWtleSJ9.eyJzdWIiOiJ3ZWJsb2dpYyIsIm9yYWNsZS5vYXV0aC51c2VyX29yaWdpbl9
pZF90eXBlIjoiTERBUF9VSUQiLCJvcmFjbGUub2F1dGggudXNlcl9vcmlnaW5faWQiOiJ3ZWJsb2dpYyIsI
mlzcyI6Im9yYWtleSIsIm9yYWNsZS5vYXV0aC5zdmNfcF9uIjoiT0F1dGhTZXJ2aWNlUHJvZmlsZSIsIml
hdCI6MTQyMjAzODEyNTAwMCwib3JhY2xlLm9hdXRoLnBybi5pZF90eXBlIjoiTERBUF9VSUQiLCJvcmFjb
GUub2F1dGggudGtfY29udGV4dCI6InJlc291cmNlX2FjY2Vzc190ayIsImV4cCI6MTQyMjA0MTcyNTAwMCw
icHJuIjoid2VibG9naWMiLCJqdGkiOiIxMDA0MTMyZC03MTBkLTRlMGEtOGI1OS01NzI0ZTFlMmI0Y2UiL
CJvcmFjbGUub2F1dGguY2xpZW50X29yaWdpbl9pZCI6ImM4MDcwMGNlYTJkNDQ1ZjFiOGQ2OWVkZDEyMDY
1ODY1Iiwib3JhY2xlLm9hdXRoLnNjb3BlIjoiVXNlclByb2ZpbGUubWUiLCJ1c2VyLnRlbmFudC5uYW1lI
joiRGVmYXVsdERvbWFpbiIsIm9yYWNsZS5vYXV0aC5pZF9kX2lkIjoiMTIzNDU2NzgtMTIzNC0xMjM0LTE
yMzQtMTIzNDU2Nzg5MDEyIn0.PymkviRiSGjkGtY9eT1BzRLXc_
kbaPMSq-SK5FY5CF6RHH5O7DLqYY0uYLd0EF8fI2zpX5AqD9B5p-12IqNox-hfR7BoPs11lgi2U-j1gZfT
XqHu7SsI3sMgwiTRrllfMD1MjoGRFYbi446C-rBiVXzUgRRaEMf9oic14O26xm4
```

Response

```
{"successful":true}
```

Using a Client Assertion

The following sample shows an access token validation request that gets a JWT client assertion using the client credentials grant type, which is used as a credential.

```
curl -i
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Fresource-access-token%2Fjwt
    &oracle_token_action=validate
    &scope=ConsentManagement.grant
    &assertion=<access token value>
    &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer
    &client_assertion=<JWT client assertion value>'
```

Response

```
{"successful":true}
```

Performing Access Token Introspection

This section provides sample REST requests that show how to query OAM OAuth Services to determine meta-information about an OAuth token. This process, called OAuth introspection, is the same as access token validation but additional claims data is included inside the access token as part of the response.

To request that the server return additional token claims data in its response, include the `oracle_token_attrs_retrieval` parameter. This parameter takes the following space-separated claims names:

```
iss aud exp prn jti exp iat oracle.oauth.scope oracle.oauth.client_origin_id
oracle.oauth.user_origin_id oracle.oauth.user_origin_id_type
oracle.oauth.tk_context oracle.oauth.id_d_id oracle.oauth.svc_p_n
```

This section includes the following examples:

- Using the Client ID and Secret in the HTTP Basic Header
- Using a Client Assertion

Using the Client ID and Secret in the HTTP Basic Header

The following token introspection sample shows the first access token validation request shown previously in the Validating an Access Token section, but with the addition of the `oracle_token_attrs_retrieval` parameter.

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Fresource-access-token%2Fjwt
    &oracle_token_action=validate
    &scope=UserProfile.users
    &oracle_token_attrs_retrieval=iss%20aud%20exp%20prn%20jti%20exp%20iat
%20oracle.oauth.scope%20oracle.oauth.client_origin_id
%20oracle.oauth.user_origin_id%20oracle.oauth.user_origin_id_type
%20oracle.oauth.tk_context%20oracle.oauth.id_d_id%20oracle.oauth.svc_p_n
    &assertion=<access token value>'
```

Response

```
{"successful":true,
 "oracle_token_attrs_retrieval":
 {"oracle.oauth.tk_context":"resource_access_tk",
  "exp":1386276668000,
  "iss":"www.oracle.example.com",
  "prn":"54321id",
  "oracle.oauth.client_origin_id":"54321id",
  "oracle.oauth.scope":"ConsentManagement.grant",
  "jti":"0fb4eef6-44ce-46ac-9230-7a335c05bf0f",
  "oracle.oauth.svc_p_n":"OAuthServiceProfile",
  "iat":1386273068000,
  "oracle.oauth.id_d_id":"12345678-1234-1234-1234-123456789012"
 }
}
```

Using a Client Assertion

The following token introspection sample shows the second access token validation request shown previously in the Validating an Access Token section, but with the addition of the `oracle_token_attrs_retrieval` parameter.

```
curl -i
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Fresource-access-token%2Fjwt
    &oracle_token_action=validate
    &scope=ConsentManagement.grant
    &oracle_token_attrs_retrieval=iss%20aud%20exp%20prn%20jti%20exp%20iat
%20oracle.oauth.scope%20oracle.oauth.client_origin_id
%20oracle.oauth.user_origin_id%20oracle.oauth.user_origin_id_type
%20oracle.oauth.tk_context%20oracle.oauth.id_d_id%20oracle.oauth.svc_p_n
    &assertion=<access token value>
    &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer
    &client_assertion=<JWT client assertion value>'
```

Response

```
{"successful":true,
 "oracle_token_attrs_retrieval":
 {"oracle.oauth.tk_context":"resource_access_tk",
  "exp":1386276668000,
  "iss":"www.oracle.example.com",
  "prn":"54321id",
  "oracle.oauth.client_origin_id":"54321id",
  "oracle.oauth.scope":"ConsentManagement.grant",
  "jti":"0fb4eef6-44ce-46ac-9230-7a335c05bf0f",
  "oracle.oauth.svc_p_n":"OAuthServiceProfile",
  "iat":1386273068000,
  "oracle.oauth.id_d_id":"12345678-1234-1234-1234-123456789012"
 }
}
```

Revoking an Access Token

This section provides sample REST requests that show how to revoke a resource access token. It includes the following examples:

- Revoking an Access Token with Client ID and Secret in an HTTP Basic Header
- Revoking an Access Token with a Client Assertion

Revoking an Access Token with Client ID and Secret in an HTTP Basic Header

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Fresource-access-token%2Fjwt
    &oracle_token_action=delete
    &assertion=<access token value>'
```

Response

```
{"successful":true}
```

Revoking an Access Token with a Client Assertion

```
curl -i
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Fresource-access-token%2Fjwt
    &oracle_token_action=delete
    &assertion=<access token value>
    &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer
    &client_assertion=<JWT client assertion value>'
```

Response

```
{"successful":true}
```

Administering a Secret Key

The following sections document the API for administering the secret key.

- Creating a Secret Key
- Getting a Secret Key
- Deleting a Secret Key
- Creating a Secret Key Using Basic Authentication

Creating a Secret Key

To create a secret key use the following REST API.

```
curl -i --request POST $SERVER_URL/ms_oauth/resources/userprofile/secretkey
  -H "Authorization: Bearer $access_token"
```

Getting a Secret Key

To retrieve a secret key use the following REST API.

```
curl -i --request GET $SERVER_URL/ms_oauth/resources/userprofile/secretkey
  -H "Authorization: Bearer $access_token"
```

A typical response would be:

```
{
    "uri": "\/ms_oauth\/resources\/userprofile\/secretkey\/weblogic",
    "secret_key": "7OWZSV2OYFZSJZWT"
}
```

Deleting a Secret Key

To delete a secret key use the following REST API.

```
curl -i --request DELETE $SERVER_URL/ms_oauth/resources/userprofile/secretkey
  -H "Authorization: Bearer $access_token"
```

Creating a Secret Key Using Basic Authentication

To create a secret key using Basic Authentication, use the following REST API.

```
curl -i -H "Content-Type: application/json"
  --request POST $SERVER_URL/ms_oauth/resources/userprofile/secretkey
  -H 'Authorization: Basic d2VibG9naWM6d2VsY29tZTE='
```

Administering the OAuth Services User Profile Service with REST

The following User Profile Service REST commands are documented in this section.

- Read My Profile
- Update My Profile
- Create a User Profile
- Read a User Profile
- Update a User Profile
- Delete a User Profile
- Create a Group Profile
- Read a Group Profile
- Update a Group Profile
- Delete a Group Profile

Read My Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.me.read`.

```
curl -i
--request GET
"http://host:port/ms_oauth/resources/userprofile/me"
-H 'Authorization:<OAUTH ACCESS TOKEN>'
```

Response

```
{
  "uid": "weblogic",
  "description": "This user is the default administrator.",
  "lastname": "Doe",
  "commonname": "John",
  "uri": "\/ms_oauth\/resources\/userprofile\/me\/weblogic"
}
```

Update My Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.me.write`.

```
curl -H
"Content-Type: application/json"
--request PUT "http://host:port/ms_oauth/resources/userprofile/me"
-H 'Authorization:<OAUTH ACCESS TOKEN>'
-d '{
     "description": "user2description"
    }'
```

Response

```
{
  "uid": "weblogic",
  "description": "user2description",
  "lastname": "Doe",
  "commonname": "John",
  "uri": "\/ms_oauth\/resources\/userprofile\/me\/weblogic"
}
```

Create a User Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.users.write`.

```
curl -H
"Content-Type: application/json"
--request POST
http://host:port/ms_oauth/resources/userprofile/users
-H 'Authorization:<OAUTH ACCESS TOKEN>'
-d '{
     "uid": "John",
     "description": "test user",
     "lastname": "Anderson",
     "commonname": "John Anderson",
     "firstname": "John"
    }'
```

Response

```
{
  "uid": "John",
  "guid": "FE1D7BD0590111E1BFDCF77FB8E715D5",
  "description": "test user",
  "name": "John",
  "lastname": "Anderson",
  "commonname": "John Anderson",
  "loginid": "John",
  "firstname": "John",
  "uniquename": "FE1D7BD0590111E1BFDCF77FB8E715D5",
  "uri": "\/ms_oauth\/resources\/userprofile\/people\/John"
}
```

Read a User Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.users.read`.

```
curl -i
--request GET
-H 'Authorization:<OAUTH ACCESS TOKEN>'
http://host:port/ms_oauth/resources/userprofile/users/John
```

Response

```
{
  "uid": "John",
  "guid": "FE1D7BD0590111E1BFDCF77FB8E715D5",
  "description": "test user",
  "name": "John",
  "lastname": "Anderson",
  "commonname": "John Anderson",
  "loginid": "John",
  "firstname": "John",
  "uniquename": "FE1D7BD0590111E1BFDCF77FB8E715D5",
  "uri": "\/ms_oauth\/resources\/userprofile\/people\/John"
}
```

Update a User Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.users.write`.

```
curl -H "Content-Type: application/json"
--request PUT
http://host:port/ms_oauth/resources/userprofile/users/John
-H 'Authorization:<OAUTH ACCESS TOKEN>'
-d '{
     "description":"test user1"
    }'
```

Response

```
{
  "uid": "John",
  "guid": "FE1D7BD0590111E1BFDCF77FB8E715D5",
  "description": "test user1",
  "name": "John",
  "lastname": "Anderson",
  "commonname": "John Anderson",
  "loginid": "John",
  "firstname": "John",
  "uniquename": "FE1D7BD0590111E1BFDCF77FB8E715D5",
  "uri": "\/ms_oauth\/resources\/userprofile\/people\/John"
}
```

Delete a User Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.users.write`.

```
curl -i
--request DELETE
-H 'Authorization:<OAUTH ACCESS TOKEN>'
http://host:port/ms_oauth/resources/userprofile/users/John
```

Response

No Response.

Create a Group Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.users.write`.

```
curl -H "Content-Type: application/json"
--request POST
http://host:port/ms_oauth/resources/userprofile/groups
-H 'Authorization:<OAUTH ACCESS TOKEN>'
-d '{
     "description":"group1 testing",
     "commonname":"group1"
    }'
```

Response

```
{
  "guid": "2259C6C0592011E1BFDCF77FB8E715D5",
  "description": "group1 testing",
  "name": "group1",
  "commonname": "group1",
  "uniquename": "2259C6C0592011E1BFDCF77FB8E715D5",
  "uri": "\/ms_oauth\/resources\/userprofile\/groups\/group1"
}
```

Read a Group Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.groups.read`.

```
curl -i
--request GET "http://host:port/ms_oauth/resources/userprofile/groups/group1"
-H 'Authorization:<OAUTH ACCESS TOKEN>'
```

Response

```
{
  "guid": "2259C6C0592011E1BFDCF77FB8E715D5",
  "description": "group1 testing",
  "name": "group1",
  "commonname": "group1",
  "uniquename": "2259C6C0592011E1BFDCF77FB8E715D5",
  "uri": "\/ms_oauth\/resources\/userprofile\/groups\/group1"
}
```

Update a Group Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.groups.write`.

```
curl -H "Content-Type: application/json"
--request PUT http://host:port/ms_oauth/resources/userprofile/groups/group1
-H 'Authorization:<OAUTH ACCESS TOKEN>'
-d '{
     "description":"group11 testing"
    }'
```

Response

```
{
  "guid": "2259C6C0592011E1BFDCF77FB8E715D5",
  "description": "group11 testing",
  "name": "group1",
  "commonname": "group1",
  "uniquename": "2259C6C0592011E1BFDCF77FB8E715D5",
  "uri": "\/ms_oauth\/resources\/userprofile\/groups\/group1"
}
```

Delete a Group Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.groups.write`.

```
curl -i
--request DELETE "http://host:port/ms_oauth/resources/userprofile/groups/group1"
-H 'Authorization:<OAUTH ACCESS TOKEN>'
```

Response

Delete a User Profile

This resource server URI is protected by an OAuth Access Token. To complete this action using the default configuration, the OAuth Access Token requires a scope of `userProfile.users.write`.

```
curl -i
--request DELETE
-H 'Authorization:<OAUTH ACCESS TOKEN>'
http://host:port/ms_oauth/resources/userprofile/users/John
```

Response

No Response.

Administering OAuth Services Consent Management Services with REST

Use this interface to customize the consent experience by rendering a custom user interface and driving the user consent process. This interface retrieves the client's consent status for all users and scopes with the POST/consentmanagement/retrieve grant. Using this interface you can enable the client to show a user all of the scopes they have previously granted.

For details on enabling user consent, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*. Configure the permissions in the Scopes section as needed. The following topics are covered in this section:

- Getting an Access Token with Client Credentials and Scope
- Accessing the Consent Management Server to Grant Consent
- Accessing the Consent Management Server to Retrieve Consent
- Accessing the Consent Management Server to Revoke Consent
- Granting the Client Permission to Access the a UserProfile Resource
- Getting the Access Token for a User's UserProfile Resource
- Accessing a User's UserProfile Resource with the Access Token

Getting an Access Token with Client Credentials and Scope

The following sample shows how to get an access token using the `client_credentials` grant type.

- Set the Authorization attribute using a "Basic" base 64 encoded (clientId:<secret>) in the request header.
- Add grant_type=client_credentials and scope=ConsentManagement.retrieve+ConsentManagement.grant+ConsentManagement.revoke to the request query.
- POST the request to the http://<host>:<port>/ms_oauth/oauth2/endpoints/oauthservice/tokens endpoint.

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
    grant_type=client_credentials
   &scope=ConsentManagement.retrieve+
          ConsentManagement.grant+
          ConsentManagement.revoke'
```

Response

The expected output is OK 200 and a valid token.

```
{
  "expires_in": 3600,
  "token_type": "Bearer",
  "access_token": "eyJhbGciOiJSyfecz3p...nYlReMjATbLs"
}
```

Accessing the Consent Management Server to Grant Consent

This cURL command illustrates how to use an access token (from Getting an Access Token with Client Credentials and Scope) to grant consent.

- Set the Authorization attribute using a "Bearer" and the previously obtained access token AT_1
- Add oracle_user_id=[a user id] (in example, weblogic)
- Add client_id=[a client id] (in example 54321id)
- Add scope=[a list of scope space separated] (in example, "samplePhotoServer.photo.read samplePhotoServer.photo.write" is used)
- POST the request to the http://<host>:<port>/ms_oauth/resources/consentmanagement/grant endpoint.

```
curl -i -H 'Authorization: Bearer AT_1'
-H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8'
--request POST http://host:port/ms_oauth/resources/consentmanagement/grant
-d 'oracle_user_id=weblogic&
scope=samplePhotoServer.photo.write+samplePhotoServer.photo.read&
lang=en&
client_id=54321id'
```

Response

The expected output is an enhanced token for samplePhotoServer.photo with the client_credentials grant type and a scope of samplePhotoServer.photo.write+samplePhotoServer.photo.read.

Accessing the Consent Management Server to Retrieve Consent

This cURL command illustrates how to use the token to retrieve the consent.

- Set the Authorization attribute using a "Bearer" and the previously obtained access token AT_1 (from Getting an Access Token with Client Credentials and Scope)
- Add oracle_user_id=[a user id] (in example, weblogic)
- Add client_id=[a client id] (in example, 54321id)
- POSt the request to the http://<host>:<port>/ms_oauth/resources/consentmanagement/retrieve endpoint.

```
curl -i -H 'Authorization: Bearer AT_1'
-H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8'
--request POST http://host:port/ms_oauth/resources/consentmanagement/retrieve
-d 'oracle_user_id=weblogic&
lang=en&
client_id=54321id'
```

Accessing the Consent Management Server to Revoke Consent

This cURL command illustrates how to use the token to revoke consent.

- Set the Authorization attribute using a "Bearer" and the previously obtained access token AT_1 (from Getting an Access Token with Client Credentials and Scope)
- Add oracle_user_id=[a user id] (in example, weblogic)
- Add client_id=[a client id] (in example, 54321id)
- Add scope=[a list of scope space separated] (in example, "samplePhotoServer.photo.read samplePhotoServer.photo.write")
- POST the request to the http://<host>:<port>/ms_oauth/resources/consentmanagement/revoke endpoint.

```
curl -i -H 'Authorization: Bearer AT_1'
-H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8'
--request POST http://host:port/ms_oauth/resources/consentmanagement/revoke
-d 'oracle_user_id=weblogic&
 scope=samplePhotoServer.photo.write+samplePhotoServer.photo.read&lang=en&
 client_id=54321id'
```

Granting the Client Permission to Access the a UserProfile Resource

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/resources/consentmanagement/grant
-d '
     scope=UserProfile.me.read
    &client_id=54321id
    &oracle_user_id=weblogic
    &lang=en
   '
-H 'Authorization: eyJhbGciOiJSUzUxM...3OxH7jIRqGL-6w'
```

Response

```
HTTP/1.1 200 OK
```

Getting the Access Token for a User's UserProfile Resource

```
curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d '
     grant_type=password
    &username=weblogic
    &password=password123
    &scope=UserProfile.me.read'
```

Response

```
{
  "expires_in": 3600,
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJSUzUxM...t7ihyNjqbb6Q9bCwE",
  "access_token": "eyJhbGciOiJSUzUxM...MIXI0ztb6NfOBMb4A"
}
```

Accessing a User's UserProfile Resource with the Access Token

The following sample demonstrates an unauthorized request and the response.

```
curl -i
--request GET "http://host:port/ms_oauth/resources/userprofile/me" -H
'Authorization: eyJhbGciOiJSUzUxM...MIXI0ztb6NfOBMb4A'
```

Response

```
HTTP/1.1 401 Unauthorized
Date: Fri, 16 Aug 2013 18:47:44 GMT
Transfer-Encoding: chunked
Content-Type: application/json
X-ORACLE-DMS-ECID: 316690b8df2db0a3:-794ed83e:140885d3651:-8000-000000000000005e
X-Powered-By: Servlet/2.5 JSP/2.1

{
  "message":
 "oracle.security.idaas.oauth.resourceserver.jaxrs.userprofile.Me.getMyProfile:
resource uri is not protected",
  "oicErrorCode": "IDAAS-20027 :
 oracle.security.idaas.rest.jaxrs.OICExceptionMapper : [ No error code is
 available from the underlying exception ]"
}
```

Using REST in OAuth Services Mobile Client 3-Legged Flows

This section documents the REST calls for 3-legged mobile client flows. For more information, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

> **Note:** All attribute names and values are case-sensitive.

The following topics are covered in this section:

- Getting an Application Profile
- Requesting a Mobile Device Client Verification Code
- Requesting an Authorization Code for Device Registration
- Creating a Client Assertion and JWT User Assertion
- Creating a Client Assertion and JWT User Assertion Using Social Authentication
- Requesting a Verification Code for Mobile Client Registration
- Requesting an Authorization Code for Mobile Device Registration
- Creating an Access Token
- Creating an Access Token Using Social Authentication
- Logging Out

Getting an Application Profile

Beginning with this 11.1.2.3.0 release, the OAM Server returns the allowed grant types in response to a Get Application Profile request. The response is returned whether server side SSO is enabled or not to inform the client of how it is configured so that the client can make correct calls to the server. Following are some example responses.

```
curl -i
--request GET 'http://host:port/ms_oauth/oauth2/
endpoints/oauthservice/appprofiles/MobileApp1?device_os=iPhone%20OS&os_
ver=7.000000'
```

Response Without Jail-Breaking Detection Policies

```
{
 "allowedGrantTypes": [
 "urn:ietf:params:oauth:grant-type:jwt-bearer",
 "client_credentials",
 "oracle-idm:/oauth/grant-type/mobile-client-registration-key",
 "password"
 ],
 "client_id":"MobileApp1",
 "mobileAppConfig":{
  "claimAttributes":[
    "oracle:idm:claims:client:geolocation",
    "oracle:idm:claims:client:imei",
    "oracle:idm:claims:client:jailbroken",
    "oracle:idm:claims:client:locale",
    "oracle:idm:claims:client:networktype",
    "oracle:idm:claims:client:ostype",
    "oracle:idm:claims:client:osversion",
    "oracle:idm:claims:client:phonecarriername",
    "oracle:idm:claims:client:phonenumber",
    "oracle:idm:claims:client:sdkversion",
    "oracle:idm:claims:client:udid",
    "oracle:idm:claims:client:vpnenabled",
    "oracle:idm:claims:client:fingerprint",
    "oracle:idm:claims:client:iosidforvendor",
    "oracle:idm:claims:client:iosidforad"
  ]
 },
 "oauthAuthZService":"/ms_oauth/oauth2/endpoints/oauthservice/authorize",
 "oauthNotificationService":"/ms_oauth/oauth2/endpoints/oauthservice/push",
 "oauthTokenService":"/ms_oauth/oauth2/endpoints/oauthservice/tokens",
 "oracleMobileSecurityLevel":"LOW",
 "userConsentService":["/ms_oauth/resources/consentmanagement"],
 "userProfileService":["/ms_oauth/resources/userprofile"],
 "oracleConsentServiceProtection":"OAM"
}
```

Response With Jail-Breaking Detection Policies

```
{
 "allowedGrantTypes": [
 "urn:ietf:params:oauth:grant-type:jwt-bearer",
 "client_credentials",
 "oracle-idm:/oauth/grant-type/mobile-client-registration-key",
 "password"
 ],
```

```
"client_id":"ACMEStock",
"jailBreakingDetectionPolicy":
{
 "autoCheckPeriodInMin":60,
 "detectionLocation":
 [
  {"action":"exists",
   "filePath":"/bin/bash",
   "success":true
  },
  {"action":"exists",
   "filePath":"/Applications/Cydia.app",
   "success":true
  },
  {"action":"exists",
   "filePath":"/Applications/limera1n.app",
   "success":true
  },
  {"action":"exists",
   "filePath":"/Applications/greenpois0n.app",
   "success":true
  },
  {"action":"exists",
   "filePath":"/Applications/blackra1n.app",
   "success":true
  },
  {"action":"exists",
   "filePath":"/Applications/blacksn0w.app",
   "success":true
  },
  {"action":"exists",
   "filePath":"/Applications/redsn0w.app",
   "success":true
  },
  {"action":"exists",
   "filePath":"/Applications/sn0wbreeze.app",
   "success":true
  }
 ],
 "device_os":"iPhone OS",
 "os_ver":"7.000000",
 "policyExpirationInSec":3600
},
"mobileAppConfig":
{
 "claimAttributes":[
  "oracle:idm:claims:client:geolocation",
  "oracle:idm:claims:client:imei",
  "oracle:idm:claims:client:jailbroken",
  "oracle:idm:claims:client:locale",
  "oracle:idm:claims:client:networktype",
  "oracle:idm:claims:client:ostype",
  "oracle:idm:claims:client:osversion",
  "oracle:idm:claims:client:phonecarriername",
  "oracle:idm:claims:client:phonenumber",
  "oracle:idm:claims:client:sdkversion",
  "oracle:idm:claims:client:udid",
  "oracle:idm:claims:client:vpnenabled",
  "oracle:idm:claims:client:fingerprint",
  "oracle:idm:claims:client:iosidforvendor",
```

```
    "oracle:idm:claims:client:iosidforad"
   ]
  },
  "oauthAuthZService":"/ms_oauth/oauth2/endpoints/oauthservice/authorize",
  "oauthNotificationService":"/ms_oauth/oauth2/endpoints/oauthservice/push",
  "oauthTokenService":"/ms_oauth/oauth2/endpoints/oauthservice/tokens",
  "oracleMobileSecurityLevel":"LOW",
  "userConsentService":["/ms_oauth/resources/consentmanagement"],
  "userProfileService":["/ms_oauth/resources/userprofile"],
  "oracleConsentServiceProtection":"OAM"
}
```

Requesting a Mobile Device Client Verification Code

This section shows the REST request for a mobile client verification code for device registration.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=client_credentials
   &oracle_device_profile=<Base 64 Encoding Device Profile>
   &client_id=<MobileApp1>
   &oracle_requested_assertions=oracle-idm:/oauth/assertion-type/client-identity/
mobile-client-pre-authz-code-client'
```

Response

```
{
 "expires_in":300,
 "token_type":"Bearer",
 "oracle_tk_context":"pre_azc",
 "access_token":"eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImt...5_LsQwlg7y-D8TW_0Q"
}
```

Requesting an Authorization Code for Device Registration

To request an authorization code for device registration, the user-agent uses the URL shown below. In return, the authorization service sends an authorization code to the client using the redirection URI.

```
http://host:port/ms_oauth/oauth2/endpoints/oauthservice/
authorize?client_id=MobileApp1&redirect_uri=<Mobile App URL Scheme>
&response_type=code
&oracle_requested_assertions=urn:ietf:params:oauth:client-assertion-type:
  jwt-bearer
&oracle_pre_authz_code=<Mobile Device Client Verification Code >
```

Response

```
<Mobile App URL Scheme>?code=eyJhbGciOiJSUzUxMiIsIns93I6...A0qenJQX5rrtRpdZJl50bS0
```

Creating a Client Assertion and JWT User Assertion

This request creates a mobile client assertion and a JWT user assertion. The JWT user assertion is stored in the server-side device store.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=authorization_code
    &code=<Authorization Code for Device Registration>
    &client_id=<MobileApp1>
    &redirect_uri=<Mobile App URL Scheme>
    &oracle_device_profile=<Base 64 Encoding Device Profile>
```

Response

```
{
 "oracle_client_assertion_
type":"urn:ietf:params:oauth:client-assertion-type:jwt-bearer",
 "expires_in":604800,
 "token_type":"Bearer",
 "oracle_tk_context":"client_assertion",
 "refresh_token":"eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6...7iEID1pLavdMsIg"
}
```

Creating a Client Assertion and JWT User Assertion Using Social Authentication

This request creates a mobile client assertion and a JWT user assertion. The Social Identity Provider sends an Access Token in the response. The JWT user assertion is stored in the server-side device store.

```
curl - i
- H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8'
- H 'Cache-Control: no-cache, no-store, must-revalidate'
--request POST http: //host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=authorization_code
   &code=<Authorization Code for Device Registration>
   &client_id=<MobileAppName>
   &redirect_uri=<Mobile App URL Scheme>
   &oracle_device_profile=<Base 64 Encoding of Device Profile>'
```

Response

```
{
 "oracle_client_assertion_
type":"urn:ietf:params:oauth:client-assertion-type:jwt-bearer",
 "social_payload":
     "{
       "UserProfile":
            {
             "mail":"exampleuser@yahoo.com",
             "lastname":"",
             "commonname":"Scott",
             "firstname":"Scott",
             "loginid":"exampleuser@yahoo.com",
             "password":"",
             "displayname":"Scott"
            },
       "IdentityProvider":"Facebook",
       "Protocol":"OAuth",
       "oauth_access_token":
           "{
             "access_token":"CAAUh80zH...wwHKZCAu",
             "expiry":5183984,
             "consumer":"OAuthMobileApplication",
             "provider":"Facebook"
             }"
     }",
 "expires_in":604800,
 "token_type":"Bearer",
 "oracle_tk_context":"client_assertion",
 "refresh_token":"eyJh.....",
 "access_token":"eyJhbGciOiJSUzUxMiIs......."
}
```

Requesting a Verification Code for Mobile Client Registration

This section shows the REST request for a mobile client verification code (if required) for device registration.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:14100/ms_
oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=client_credentials
   &oracle_device_profile=<Base 64 Encoding Device Profile>
   &client_id=<MobileApp1>
   &oracle_requested_assertions=oracle-idm:/oauth/assertion-type/client-identity
/mobile-client-pre-authz-code-access'
```

Response

```
{
 "expires_in":300,
 "token_type":"Bearer",
 "oracle_tk_context":"pre_azc",
 "access_token":"eyJhbGciOiJSUzUxMiI4sInR5h4cCI6IkpXVCIsIm...NQXXd5_LsQy-D8TW_0Q"
}
```

Requesting an Authorization Code for Mobile Device Registration

To request an authorization code for device registration, the user-agent uses the URL shown below. In return, the authorization service sends an authorization code to the client using the redirection URI.

```
http://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/
authorize?client_id=MobileApp1&redirect_uri=<Mobile App URL Scheme>
&response_type=code
&scope=<Resource Scope>
&oracle_pre_authz_code=<optional Mobile Device Client Verification Code>
```

Response

```
<Mobile App URL Scheme>?code=eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVC...m_
7FMwRXyEJI8J4JmPDf8RFdM7MP4_x3IBmK9amUAPRFJRNg
```

Creating an Access Token

The following request creates an OAuth Access Token if the JWT User Assertion is valid in the server-side device store.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:14100/ms_oauth/oauth2/endpoints
/oauthservice/tokens
-d 'grant_type=authorization_code
   &code=<Authorization Code for Access Token>
   &client_id=<MobileApp1>
   &redirect_uri=<Mobile App URL Scheme>
   &oracle_device_profile=<optional base 64 encoding device profile>
   &client_assertion=<Mobile Client Assertion>
   &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer'
```

Response

```
{
 "expires_in":3600,
 "token_type":"Bearer",
 "refresh_token":"eyJhbGiiIsInR5cCI6IkpXVCmtaWRfdHlwZSI6IBfVUDM5Qi00Q0U3LUxyJ6ndU"
}
```

Creating an Access Token Using Social Authentication

The following request creates an OAuth Access Token if the JWT User Assertion is valid in the server-side device store. The Social Identity Provider also sends an Access Token in the response.

```
curl -i
-H 'Accept: */*'
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=authorization_code
   &code=<Authorizaton Code for Access Token>
   &client_id=<MobileAppName>
   &redirect_uri=<Mobile App URL Scheme>
   &oracle_device_profile=<Optional Base 64 Encoding of Device Profile>
   &client_assertion=<Mobile Client Assertion>
   &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer'
```

Response

```
{
 "expires_in":3600,
 "token_type":"Bearer",
 "access_token":"ey...3JhY2xlLm9hdXRoLn",
 "social_payload":
      "{
        "UserProfile":
             {
              "mail":"exampleuser@yahoo.com",
              "lastname":"",
              "commonname":"Scott",
              "firstname":"Scott",
              "loginid":"exampleuser@yahoo.com",
              "password":"",
              "displayname":"Scott"
             },
        "IdentityProvider":"Facebook",
        "Protocol":"OAuth",
        "oauth_access_token":
            "{
              "access_token":"CAAUh80zHfPQBA....lP4kmNRyg",
              "expiry":5113635,
              "consumer":"OAuthMobileApplication",
              "provider":"Facebook"
             }"
      }"
}
```

Logging Out

This request provides mobile single sign-out as follows:

- Removes the JWT user assertion from the server-side device key chain
- Terminates and removes OAM user tokens and OAM user session data from the server-side device keystore

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/oammsui/oauthservice/logout
-d 'client_id=MobileApp1
    &redirect_uri=mobileapp://
    &oracle_device_profile=<Base 64 Encoding Device Profile>
    &client_assertion=<Mobile Client Assertion>
    &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer'
```

Response

```
HTTP/1.1 200 OK

Date: Mon, 02 Dec 2013 22:55:37 GMT

Content-Length: 0

Set-Cookie:
JSESSIONID=z17tSdPLd7TG11dw7wNtTlJnzGXty3y3B8TqwW1GNvHjmzv6FqGv!535445357; path=/;
HttpOnly

X-ORACLE-DMS-ECID: 09edd9b26949554d:f4833c6:142b4da1082:-8000-000000000000277f

X-Powered-By: Servlet/2.5 JSP/2.1
```

Using REST in OAuth Services Mobile Client 2-Legged Flows

This section documents the REST calls for 2-legged mobile client flows. For more information, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

> **Note:** All attribute names and values are case-sensitive.

The following topics are covered in this section:

- Getting an Application Profile
- Requesting a Mobile Device Client Verification Code
- Registering a Mobile App and Creating Assertions
- Answer the Knowledge-Based Authentication (KBA) Challenge Request
- Logging Out
- Logging In
- Creating OAM User and Master Tokens with Valid JWT
- Creating OAM Access and Master Tokens with Valid OAM User Token
- Creating an OAuth Services Access Token Using an OAM Credential Grant Type
- Creating an OAuth Services Access Token Using a Standard JWT User Assertion Grant
- Mobile Flows When the Server-Side SSO Feature is Disabled

Getting an Application Profile

Beginning with this 11.1.2.3.0 release, the OAM Server returns the allowed grant types in response to a Get Application Profile request. The response is returned whether server side SSO is enabled or not to inform the client of how it is configured so that the client can make correct calls to the server. Following is the request and sample responses.

```
curl -i
--request GET 'http://host:port/ms_oauth/oauth2/endpoints
/oauthservice/appprofiles/MobileApp1?device_os=iPhone%20OS&os_ver=7.000000'
```

HTTP Response

```
{"allowedGrantTypes":["oracle-idm:/oauth/grant-type/oam_credentials",
"urn:ietf:params:oauth:grant-type:jwt-bearer","refresh_token","code","client_
credentials","authorization_code","password",
"oracle-idm:/oauth/grant-type/challenge-answer"],
"client_id":"mobileClient",
"mobileAppConfig":{"claimAttributes":
["oracle:idm:claims:client:sdkversion","oracle:idm:claims:client:networktype",
"oracle:idm:claims:client:fingerprint","oracle:idm:claims:client:phonenumber",
"oracle:idm:claims:client:iosidforad","oracle:idm:claims:client:ostype",
"oracle:idm:claims:client:imei","oracle:idm:claims:client:phonecarriername",
"oracle:idm:claims:client:iosidforvendor","oracle:idm:claims:client:jailbroken",
"oracle:idm:claims:client:udid","oracle:idm:claims:client:geolocation",
"oracle:idm:claims:client:vpnenabled","oracle:idm:claims:client:locale",
"oracle:idm:claims:client:osversion"]},
"oauthAuthZService":"/ms_oauth/oauth2/endpoints/oauthservice/authorize",
"oauthNotificationService":"/ms_oauth/oauth2/endpoints/oauthservice/push",
"oauthTokenService":"/ms_oauth/oauth2/endpoints/oauthservice/tokens",
"oracleConsentServiceProtection":"OAM","oracleMobileSecurityLevel":"LOW",
"server_side_sso":true,"sharedKeyAttributeName":"secret_key",
"userConsentService":["/ms_oauth/resources/consentmanagement"],
"userProfileService":["/ms_oauth/resources/userprofile"]}
```

HTTP Response Without Jail-Breaking Detection Policies

```
{
 "allowedGrantTypes": [
 "urn:ietf:params:oauth:grant-type:jwt-bearer",
 "client_credentials",
 "oracle-idm:/oauth/grant-type/mobile-client-registration-key",
 "password"
 ],
 "client_id":"MobileApp1",
 "mobileAppConfig":{
  "claimAttributes":[
    "oracle:idm:claims:client:geolocation",
    "oracle:idm:claims:client:imei",
    "oracle:idm:claims:client:jailbroken",
    "oracle:idm:claims:client:locale",
    "oracle:idm:claims:client:networktype",
    "oracle:idm:claims:client:ostype",
    "oracle:idm:claims:client:osversion",
    "oracle:idm:claims:client:phonecarriername",
    "oracle:idm:claims:client:phonenumber",
    "oracle:idm:claims:client:sdkversion",
    "oracle:idm:claims:client:udid",
```

```
    "oracle:idm:claims:client:vpnenabled",
    "oracle:idm:claims:client:fingerprint",
    "oracle:idm:claims:client:iosidforvendor",
    "oracle:idm:claims:client:iosidforad"
  ]
 },
 "oauthAuthZService":"/ms_oauth/oauth2/endpoints/oauthservice/authorize",
 "oauthNotificationService":"/ms_oauth/oauth2/endpoints/oauthservice/push",
 "oauthTokenService":"/ms_oauth/oauth2/endpoints/oauthservice/tokens",
 "oracleMobileSecurityLevel":"LOW",
 "userConsentService":["/ms_oauth/resources/consentmanagement"],
 "userProfileService":["/ms_oauth/resources/userprofile"],
 "oracleConsentServiceProtection":"OAM"
}
```

HTTP Response With Jail-Braking Detection Policies

```
{
 "allowedGrantTypes": [
 "urn:ietf:params:oauth:grant-type:jwt-bearer",
 "client_credentials",
 "oracle-idm:/oauth/grant-type/mobile-client-registration-key",
 "password"
 ],
 "client_id":"ACMEStock",
 "jailBreakingDetectionPolicy":
 {
  "autoCheckPeriodInMin":60,
  "detectionLocation":
  [
   {"action":"exists",
    "filePath":"/bin/bash",
    "success":true
   },
   {"action":"exists",
    "filePath":"/Applications/Cydia.app",
    "success":true
   },
   {"action":"exists",
    "filePath":"/Applications/limera1n.app",
    "success":true
   },
   {"action":"exists",
    "filePath":"/Applications/greenpois0n.app",
    "success":true
   },
   {"action":"exists",
    "filePath":"/Applications/blackra1n.app",
    "success":true
   },
   {"action":"exists",
    "filePath":"/Applications/blacksn0w.app",
    "success":true
   },
   {"action":"exists",
    "filePath":"/Applications/redsn0w.app",
    "success":true
   },
   {"action":"exists",
    "filePath":"/Applications/sn0wbreeze.app",
```

```
    "success":true
   }
  ],
  "device_os":"iPhone OS",
  "os_ver":"7.000000",
  "policyExpirationInSec":3600
 },
 "mobileAppConfig":
 {
  "claimAttributes":[
   "oracle:idm:claims:client:geolocation",
   "oracle:idm:claims:client:imei",
   "oracle:idm:claims:client:jailbroken",
   "oracle:idm:claims:client:locale",
   "oracle:idm:claims:client:networktype",
   "oracle:idm:claims:client:ostype",
   "oracle:idm:claims:client:osversion",
   "oracle:idm:claims:client:phonecarriername",
   "oracle:idm:claims:client:phonenumber",
   "oracle:idm:claims:client:sdkversion",
   "oracle:idm:claims:client:udid",
   "oracle:idm:claims:client:vpnenabled",
   "oracle:idm:claims:client:fingerprint",
   "oracle:idm:claims:client:iosidforvendor",
   "oracle:idm:claims:client:iosidforad"
  ]
 },
 "oauthAuthZService":"/ms_oauth/oauth2/endpoints/oauthservice/authorize",
 "oauthNotificationService":"/ms_oauth/oauth2/endpoints/oauthservice/push",
 "oauthTokenService":"/ms_oauth/oauth2/endpoints/oauthservice/tokens",
 "oracleMobileSecurityLevel":"LOW",
 "userConsentService":["/ms_oauth/resources/consentmanagement"],
 "userProfileService":["/ms_oauth/resources/userprofile"],
 "oracleConsentServiceProtection":"OAM"
}
```

Requesting a Mobile Device Client Verification Code

This section shows the REST request for a mobile client verification code for device registration.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2
/endpoints/oauthservice/tokens
-d 'grant_type=client_credentials
    &oracle_device_profile=<Base 64 Encoding Device Profile>
    &client_id=<MobileApp1>
    &oracle_requested_assertions=oracle-idm:/oauth/assertion-type/client-identity/
mobile-client-pre-authz-code-client'
```

Response

```
{
 "expires_in":300,
 "token_type":"Bearer",
 "oracle_tk_context":"pre_azc",
 "access_token":"eyJhbGciOiJg0LzJL...6LnHpAhcZA-EfJU9jQYH4GPINQXXd5_LsQy-D8TW_0Q"
}
```

Registering a Mobile App and Creating Assertions

This request creates a mobile client assertion and a JWT user assertion. The JWT user assertion is stored in the server-side device store. In addition, if Oracle Adaptive Access Manager and the adaptive-access security plug-in are active, an OAAM device handle and OAAM session handle are created and also stored in the server-side device store.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2
/endpoints/oauthservice/tokens
-d 'grant_type=password
    &username=userAbc123
    &password=passwordAbc123
    &client_id=<MobileApp1>
    &oracle_pre_authz_code=<Mobile Device Verification Code>
    &oracle_device_profile=<Base 64 Encoding Device Profile>
    &oracle_requested_assertions=urn:ietf:params:oauth:
client-assertion-type:jwt-bearer'
```

Response

This is the response if Oracle Adaptive Access Manager and the adaptive-access security plug-in are *not* active.

```
{
 "expires_in":3600,
 "token_type":"Bearer",
 "access_token":"eyJhbcOiJSzUxMIsInR5cCI6IkpX...OQN5mrZrl5pGyEJOMm4BSLQVVZhLsS5g"
}
```

Response if OAAM and the Adaptive-Access Security Plug-in are Enabled

```
{
  "oracle_client_assertion_type":"urn:ietf:params:oauth:client-assertion-type:jwt-bearer",
  "oracle_aux_tokens":
   {
    "user_assertion":
     {
      "oracle_token_in_server_device_store":true,
      "expires_in":28800,
      "token_type":"Bearer",
      "oracle_tk_context":"user_assertion",
      "oracle_grant_type":"urn:ietf:params:oauth:grant-type:jwt-bearer",
      "access_token":"eyJhbGciOiJSUzUxM...6Ik5BRVNyanZha0dUlBVGlGQSJ9"
     }
   },
  "expires_in":604800,
  "token_type":"Bearer",
  "oracle_tk_context":"client_assertion",
  "access_token":"eyJhbGciOiJSUzUxM...6Ik5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9"
}
```

Response if the Security Plug-in Responds With "Denied"

This response only occurs if Oracle Adaptive Access Manager and the adaptive-access security plug-in are active. If the security plug-in responds with "denied," nothing is created or stored in the server-side device store.

```
HTTP/1.1 401 Unauthorized
{
  "error":"DENIED",
  "error_description":"Denied Action is triggered",
}
```

Response if the Challenge Action is Triggered

This response only occurs if Oracle Adaptive Access Manager and the adaptive-access security plug-in are active. If the security plug-in responds with "challenge," a challenge question is returned. User information associated with `mobile.multi_step_authn_session_handle` is stored in memory with a time-out value. The user must answer the challenge question before the time-out value expires. To send the user's response, see "Answer the Knowledge-Based Authentication (KBA) Challenge Request."

```
HTTP/1.1 401 Unauthorized
{
   "error": "REQUIRE_MULTI_STEP_AUTHN",
   "error_description": "The Challenge Action is triggered",
   "multi-step-challenge-question":
   {
      "challengeType": "KBA",
      "locale": "en-us",
      "questionRefId": "80",
      "questionStr": "What model was your first car?",
      "mobile.multiStepAuthnSessionHandle": "eyJvcmlnU2VjdXJpdHlFdmVudHMiOlsiUkVHX1…..."
   }
}
```

Answer the Knowledge-Based Authentication (KBA) Challenge Request

Applies if Oracle Adaptive Access Manager and the adaptive-access security plug-in are active, and if the plug-in responds with "challenge."

```
curl - i
- H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm:/mobile/grant-type/mobile-client-challenge-answer
   &oracle_device_profile=<Base 64 Encoded Device Profile>
   &challenge_response=<Base 64 Encoded Response>
   &oracle_requested_assertions=urn:ietf:params:oauth:client-assertion-type:jwt-bearer'
```

In the `challenge_response` request parameter, supply the base-64 encoded version of the following JSON:

```
{
   "challenge":"KBA",
   "locale": "en - us",
   "question_ref_id": "80",
   "mobile.multi_step_authn_session_handle": "eyJvcmlnU2VjdXJpdHlFdmVudHMiOlsiUkVHX1…..."
}
```

"Allowed" Response

If the security plug-in verifies the answer and responds with "allowed," the OAAM device handle and OAAM session handle will be created and saved to the server-side keystore.

```
{
   "oracle_client_assertion_type":"urn:ietf:params:oauth:client-assertion-type:jwt-bearer",
   "oracle_aux_tokens":
   {
      "user_assertion":
      {
         "oracle_token_in_server_device_store":true,
         "expires_in":28800,
         "token_type":"Bearer",
         "oracle_tk_context":"user_assertion",
         "oracle_grant_type":"urn:ietf:params:oauth:grant-type:jwt-bearer",
         "access_token":"eyJhbGciOiJSUzUxM...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9"
      }
   },
   "expires_in":604800,
   "token_type":"Bearer",
   "oracle_tk_context":"client_assertion",
   "access_token":"eyJhbGciOiJSUzUxM...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9"
}
```

"Denied" Response

If the security plug-in responds with denied, nothing is created or stored in the server-side keystore.

```
HTTP/1.1 401 Unauthorized
{
   "error":"DENIED",
   "error_description":"Denied Action is triggered"
}
```

"Timeout" Response

If the user does not answer the challenge question before the time-out value expires, the security plug-in does not verify the answer and nothing is created or stored in the server-side keystore.

```
HTTP/1.1 401 Unauthorized
{
   "error":"TIMEOUT",
   "error_description":"Timeout Action is triggered"
}
```

Logging Out

This request cleans the JWT user assertion from the server-side device key chain.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/oammsui/oauthservice/logout
-d 'client_id=MobileApp1
    &redirect_uri=mobileapp://
    &oracle_device_profile=<Base 64 Encoding Device Profile>
    &client_assertion=<Mobile Client Assertion>
    &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer'
```

Response

```
HTTP/1.1 200 OK

Date: Mon, 02 Dec 2013 22:55:37 GMT

Content-Length: 0

Set-Cookie:
JSESSIONID=z17tSdPLd7TG11dw7wNtTlJnzGXty3y3B8TqwW1GNvHjmzv6FqGv!535445357; path=/;
HttpOnly

X-ORACLE-DMS-ECID: 09edd9b26949554d:f4833c6:142b4da1082:-8000-000000000000277f

X-Powered-By: Servlet/2.5 JSP/2.1
```

Logging In

This request creates a JWT user assertion in the server-side key chain.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2
/endpoints/oauthservice/tokens
-d 'grant_type=password
    &username=user123
    &password=pwd456xyz
    &client_assertion=<MOBILE CLIENT ASSERTION>
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type
%3Ajwt-bearer
    &oracle_device_profile=<BASE 64 ENCODING DEVICE PROFILE>
    &oracle_requested_assertions=oracle-idm%3A%2Foauth%2Fassertion-type
%2Fuser-identity%2Fjwt&oracle_use_server_device_store=true'
```

Response

```
{"oracle_token_in_server_device_store":true,
 "expires_in":28800,
 "token_type":"Bearer",
 "oracle_tk_context":"user_assertion",
 "oracle_grant_type":"urn:ietf:params:oauth:grant-type:jwt-bearer",
 "access_token":""}
```

Creating OAM User and Master Tokens with Valid JWT

This request creates an OAM user token and an OAM master token if the JWT user assertion is valid in the server-side device store.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
-request http://host:port/ms_oauth/oauth2/
endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &user_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Auser-assertion-type
%3Ajwt-bearer
    &client_assertion=<MOBILE CLIENT ASSERTION>
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type
%3Ajwt-bearer
    &oracle_device_profile=<BASE 64 ENCODING DEVICE PROFILE>
    &oracle_use_server_device_store=true'
```

Response

```
{"oracle_token_in_server_device_store":true,
 "oracle_aux_tokens":
 {"oam_mt":
   {"oracle_tk_context":"oam_mt",
    "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/oam\/master-token",
    "access_token":"VERSION_4%7EDj10z62v9CQbnuX...Stid6XMhamU%2B"
   }
 },
 "oracle_tk_context":"oam_ut",
 "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/user-token\/oam",
 "access_token":""
}
```

Creating OAM Access and Master Tokens with Valid OAM User Token

This request creates an OAM access token and an OAM master token if the OAM user token is valid in the server-side device store. Note that in the following request `oracle_oam_application_resource` is a WebGate protected resource, and `oracle_oam_application_context` is a WebGate generated value.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
-request http://host:port/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &oracle_use_server_device_store=true
    &user_assertion_type=oracle-idm:/oauth/assertion-type/user-identity/oam
    &client_assertion=<MOBILE CLIENT ASSERTION>
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type
%3Ajwt-bearer
    &oracle_device_profile=<BASE 64 ENCODING DEVICE PROFILE>
    &scope=oracle.security.oauth.oam.resource_access
    &oracle_oam_application_context=<WebGate generated value>
    &oracle_oam_application_resource=http%3A%2F%2Fhost.example.com
%3A12884%2Findex.html'
```

Response

```
{
 "oracle_aux_tokens":
   {"oam_ut":
     {"oracle_token_in_server_device_store":true,
      "oracle_tk_context":"oam_ut",
      "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/user-token\/oam",
      "access_token":""
     }
    },
 "oracle_tk_context":"oam_at",
 "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/resource-access-token\/oam",
 "access_token":"3F62m7EDq%2FRMIwA16gUjg40DT43xDEik...xAViyc7XmzGIFBoBsNbbuN6SO1"
}
```

Creating an OAuth Services Access Token Using an OAM Credential Grant Type

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &username=alice
    &password=welcome
    &client_assertion=<MOBILE CLIENT ASSERTION>
    &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
    &oracle_device_profile=<BASE 64 ENCODING DEVICE PROFILE>
    &oracle_use_server_device_store=true
    &scope=UserProfile.users'
```

Response

```
{
 "expires_in":3600,
 "token_type":"Bearer",
 "access_token":"eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCciO...iJSfkhhXLHhonktvigMCeI"
}
```

Creating an OAuth Services Access Token Using a Standard JWT User Assertion Grant

The following request creates an OAuth Services Access Token if the JWT User Assertion is valid in the server-side device store.

```
curl -i
- H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST
http: //host:port/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer
    &client_id=App2
    &oracle_pre_authz_code=<Mobile DeviceVerification Code >
    &oracle_device_profile = < Base 64 Encoding DeviceProfile >
    &oracle_requested_assertions = urn: ietf: params: oauth: client- assertion-
type: jwtbearer
    &oracle_use_server_device_store = true'
```

Response

```
HTTP / 1.1 200 OK
{
  "oracle_client_assertion_type":
"un:ietf:params:oauth:client-assertion-type:jwt-bearer",
  "expires_in": 604800,
  "token_type": "Bearer",
  "oracle_tk_context": "client_assertion",
  "refresh_token": "eyJhbGciOiJSUzUxMiIsInR5cCI6Ikp...mbmU5cDl2WjhtdUlBVGlGQSJ9.",
  "access_token":  "eyJhbGciOiJSUzUxMiIsInR5cCI6Ikp...mbmU5cDl2WjhtdUlBVGlGQSJ9."
}
```

Response if the Server-Side JWT User Token is Expired or Invalid

```
HTTP/1.1 401 Unauthorized
{
  "error":"invalid_grant",
  "error_description":"Invalid Grant: grant_
type=urn:ietf:params:oauth:grant?]type:jwt?]bearer"}
```

Mobile Flows When the Server-Side SSO Feature is Disabled

The advantage of using server-side SSO is that the server will maintain the session and associated artifacts and the client can focus on the business aspects of the application rather than maintaining sessions. Only when the client needs to control SSO, should server-side SSO be disabled. If server-side SSO is turned off, two-legged mobile OAuth Services scenarios will return tokens to the application instead of storing tokens in the server-side device store.

> **Note:** For more information, see Understanding Mobile OAuth Services Server-Side Single Sign-on in the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

An administrator can disable the server-side SSO option at the OAuth Services Service Profile level by setting the `serverside.sso.enabled` configuration parameter to `false`. The following sections contain details on mobile requests and responses when server-side SSO is disabled.

- Register Mobile App1 Using a User Name and Password
- Register Mobile App2 Using a JWT User Assertion Grant
- Create an Access Token Using a Standard JWT User Assertion Grant With a JWT Client Assertion and a User Assertion
- Answer the Knowledge-Based Authentication (KBA) Challenge Request
- Create an Access Token Using a Refresh Token
- Terminate the JWT User Assertion
- Login (Create JWT User Assertion)
- Create an OAM User Token and OAM Master Token using a JWT User Assertion (Token Exchange)
- Create an OAM User Token and OAM Master Token Using JWT User Assertion + User PIN Credential (Token Exchange)
- Create an OAM Access Token using the OAM User Token

Register Mobile App1 Using a User Name and Password

Create the client and user assertion.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST https://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=password
   &username=userAbc123
   &password=passwordAbc123
   &client_id=App1
   &oracle_pre_authz_code=<Mobile Device Verification Code>
   &oracle_device_profile=<Base 64 Encoding Device Profile>
   &oracle_requested_assertions=urn:ietf:params:oauth:client-assertion-type:jwt-bearer'
```

Response

```
{
  "oracle_client_assertion_type":"urn:ietf:params:oauth:client-assertion-type:jwt-bearer",
```

```
  "oracle_aux_tokens":
   {
    "user_assertion":
     {
      "expires_in":28800,
      "token_type":"Bearer",
      "oracle_tk_context":"user_assertion",
      "oracle_grant_type":"urn:ietf:params:oauth:grant-type:jwt-bearer",
      "access_token":"eyJhbGciOiJSUzUxM...6Ik5BRVNyanZha0dUlBVGlGQSJ9"
     }
   },
  "expires_in":604800,
  "token_type":"Bearer",
  "oracle_tk_context":"client_assertion",
  "refresh_token":"eyJhbGciOiJSUzUxM...6Ik5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9",
  "access_token":"eyJhbGciOiJSUzUxM...6Ik5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9"
}
```

Register Mobile App2 Using a JWT User Assertion Grant

Create the client assertion.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST https://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer
   &client_id=App2
   &oracle_pre_authz_code=<Mobile Device Verification Code>
   &oracle_device_profile=<Base 64 Encoding Device Profile>
   &oracle_requested_assertions=urn:ietf:params:oauth:client-assertion-type:jwt-bearer
   &assertion=<JWT User Assertion>'
```

Positive Response

```
HTTP/1.1 200 OK
{
  "oracle_client_assertion_type":"urn:ietf:params:oauth:client-assertion-type:jwt-bearer",
  "expires_in":604800,
  "token_type":"Bearer",
  "oracle_tk_context":"client_assertion",
"refresh_token":"eyJhbGciOiJSUzUxM...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9",
"access_token":"eyJhbGciOiJSUzUxM...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9"
}
```

Negative Response

```
HTTP/1.1 401 Unauthorized

{"error":"invalid_grant",
 "error_description":"Invalid Grant: grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer"
}
```

Response if the Challenge Action is Triggered

Only applies if the adaptive-access security plug-in for Oracle Adaptive Access Manager is active and if knowledge-based authentication (KBA) is enabled.

```
HTTP/1.1 401 Unauthorized
```

```
{"error":"require_multi_step_authn",
 "oracle_challenge_questions":
    {"questionList":
      [
       {"challengeType":"KBA",
        "questionStr":"What color was your first dog?",
        "questionRefId":"98"
       }
      ],
    "mobile.multiStepAuthnSessionHandle":"eyJ……MkE",
    "locale":"en"
},
"error_description":"The Challenge Action is triggered "
}
```

Create an Access Token Using a Standard JWT User Assertion Grant With a JWT Client Assertion and a User Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST https://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer
   &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer
   &assertion=<JWT User Assertion>
   &client_id=App1
   &client_assertion=<Mobile Client Assertion>
   &scope=UserProfile.users'
```

In this request, send the <JWT User Assertion> and <Mobile Client Assertion> response values that were returned during the sample request Register Mobile App1 Using a User Name and Password.

Positive Response

```
{
  "expires_in":3600,
  "token_type":"Bearer",
  "refresh_token":"eyJhbGciOiJSUzUx...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9",
  "access_token":"eyJhbGciOiJSUzUxMi...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9" "
}
```

Negative Response

```
HTTP/1.1 401 Unauthorized
{
  "error":"invalid_grant",
  "error_description":"Invalid Grant: grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer"
}
```

If the JWT User Assertion value is expired, then the mobile application can create a JWT User Assertion using the Login (Create JWT User Assertion) step.

Answer the Knowledge-Based Authentication (KBA) Challenge Request

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST https://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type= oracle-idm:/oauth/grant-type/challenge -answer&
   &client_id=App1
   &oracle_requested_assertions=urn:ietf:params:oauth:client-assertion-type:jwt-bearer
   &oracle_device_profile=<Base 64 Encoded Device Profile>
```

```
   &oracle_challenge_response=<Base 64 Encoded Response>'
```

In the `oracle_challenge_response` request parameter, supply the base-64 encoded version of the following JSON:

```
{
  "mobile.multi_step_authn_session_handle":"eyJ......MkE",
  "locale":"en",
  "answer_list":
  [
    {
      "question_ref_id":"98",
      "challenge_type":"KBA",
      "question_ans":"dog"
    }
  ]
}
```

Positive HTTP Response

```
HTTP/1.1 200 OK
{
  "oracle_client_assertion_type":"urn:ietf:params:oauth:client-assertion-type:jwt-bearer",
  "oracle_aux_tokens":{
    "user_assertion":{
      "expires_in":28800,
      "token_type":"Bearer",
      "oracle_tk_context":"user_assertion",
      "oracle_grant_type":"urn:ietf:params:oauth:grant-type:jwt-bearer",
      "access_token":"eyJhbGciOiJSUzUxM...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9"
    }
  },
  "expires_in":604800,
  "token_type":"Bearer",
  "oracle_tk_context":"client_assertion",
  "refresh_token":"eyJhbGciOiJSUzUxM...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9",
  "access_token":"eyJhbGciOiJSUzUxM...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9"
}
```

Create an Access Token Using a Refresh Token

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST https://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=refresh_token
    &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer
    &client_id=App1
    &client_assertion=<Mobile Client Assertion>
    &scope=UserProfile.users
    &refresh_token=<Refresh Token>'
```

Positive Response

```
{
  "expires_in":3600,
  "token_type":"Bearer",
  "refresh_token":"eyJhbGciOiJSUzUxM...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9",
  "access_token":"eyJhbGciOiJSUzUxM...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9"
}
```

Terminate the JWT User Assertion

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST https://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'client_id=App1
    &grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Fuser-token%2Fjwt
    &assertion=<JWT User Assertion>
    &oracle_token_action=delete
    &oracle_device_profile=<Base 64 Device Profile>
    &client_assertion=<Mobile Client Assertion>
    &client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer'
```

In this request, send the <JWT User Assertion> and <Mobile Client Assertion> response values that were returned during the sample request Register Mobile App1 Using a User Name and Password.

Positive Response

```
{"successful":true}
```

Login (Create JWT User Assertion)

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST  https://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=password
   &username=weblogic
   &password=welcome1
   &client_assertion=<MOBILE CLIENT ASSERTION>
   &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
   &oracle_device_profile=<Base 64 Device Profile>
   &oracle_requested_assertions=oracle-idm%3A%2Foauth%2Fassertion-type%2Fuser-identity%2Fjwt'
```

Positive Response

```
{
  "expires_in":28800,
  "token_type":"Bearer",
  "oracle_tk_context":"user_assertion",
  "oracle_grant_type":"urn:ietf:params:oauth:grant-type:jwt-bearer",
  "access_token":"eyJhbGciOiJSUzUxM...k5BRVNyanZha0RmbmU5cDl2WjhtdUlBVGlGQSJ9"
}
```

Negative HTTP Response if the User Name and Password are Invalid

```
HTTP/1.1 401 Unauthorized

{
 "error":"invalid_grant",
 "error_description":"Invalid resource owner user name or password "
}
```

Create an OAM User Token and OAM Master Token using a JWT User Assertion (Token Exchange)

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
```

```
-request https://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
   &user_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Auser-assertion-type%3Ajwt-bearer
   &user_assertion=<JWT User Assertion>
   &client_assertion=<MOBILE CLIENT ASSERTION>
   &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
   &oracle_device_profile=<Base 64 Device Profile>'
```

Positive Response

```
{
  "oracle_aux_tokens":
   {
    "oam_mt":
    {
      "oracle_tk_context":"oam_mt",
      "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/oam\/master-token",
      "access_token":"VERSION_4%...."
    }
   },
  "oracle_tk_context":"oam_ut",
  "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/user-token\/oam",
  "access_token":"fEmB0nPdgGfyNjshws8z….. "
}
```

Create an OAM User Token and OAM Master Token Using JWT User Assertion + User PIN Credential (Token Exchange)

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
-request https://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &user_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Auser-assertion-type%3Ajwt-bearer
    &user_assertion=<JWT User Assertion>
    &client_assertion=<MOBILE CLIENT ASSERTION>
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
    &oracle_device_profile=<Base 64 Device Profile>
    &oracle_user_credentials=<Base 64 encoding of user credential>'
```

In the `oracle_user_credentials` request parameter, supply the base-64 encoded version of the user credential payload JSON. For example, if this is the PIN:

```
{"pin":"123"}
```

The Base 64 encoded value is this:

```
eyJwaW4iOiIxMjMifQ==
```

Positive Response

```
{
  "oracle_aux_tokens":
  {
    "oam_mt":
    {
      "oracle_tk_context":"oam_mt",
      "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/oam\/master-token",
      "access_token":"VERSION_4%...."
    }
  },
```

```
  "oracle_tk_context":"oam_ut",
  "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/user-token\/oam",
  "access_token":"fEmB0nPdgGfyNjshws8z….. "
}
```

Create an OAM Access Token using the OAM User Token

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
-request https://host.example.com:14100/ms_oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &user_assertion_type=oracle-idm:/oauth/assertion-type/user-identity/oam
    &client_assertion=<MOBILE CLIENT ASSERTION>
    &user_assertion=<JWT User Assertion>
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
    &oracle_device_profile=<Base 64 Device Profile>
    &scope=oracle.security.oauth.oam.resource_access&oracle_oam_application_context=dfsdfsdfsdfsdf
    &oracle_oam_application_resource=http%3A%2F%2Fhost.example.com%3A12884%2Findex.html'
```

Positive Response

```
{
  "oracle_tk_context":"oam_at",
  "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/resource-access-token\/oam",
  "access_token":"3F62m7EDq%....."
}
```

Using Credentials, PIN and Assertions to Get Tokens

This section documents the REST calls for procuring tokens from OAuth Services.

> **Note:** All attribute names and values are case-sensitive.

The following topics are covered in this section:

- Using a Client Credential + User Name and Password Combination
- Using a Client Credential + oracle_user_credentials Combination
- Using JWT Assertion
- Using JWT Assertion + PIN
- Using SAML2 Assertion
- Getting OAM Tokens on Mobile Devices

Using a Client Credential + User Name and Password Combination

This section documents how to use a client credential together with a user name and password to get the following token types: a JWT user token, a JWT access token, an OAM user token and master token, or an OAM access token.

The following topics are covered in this section:

Overview

Requests in this section use the following basic template.

```
curl -i
-H 'Authorization: Basic <sample client ID and password>'
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/
endpoints/oauthservice
 /tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
   &username=<username>
   &password=<password>
   &oracle_requested_assertions=<Oracle_Requested_Assertion_Type>
   &oam_authen_resource=<oam_authen_resource>'
```

Note the following:

- The sample client ID and password takes the following form:

  ```
  userID123:password123

  --> base 64 encoding -->

  NTQzMjFpZDp3ZWxjb21lMQ==
  ```

  The actual client ID will be a machine generated GUID.

- You can specify the following assertion types:
  - `oracle-idm%3A%2Foauth%2Fassertion-type%2Fuser-identity%2Foam`
  - `oracle-idm%3A%2Foauth%2Fassertion-type%2Fuser-identity%2Fjwt`
- Use the `oam_authen_resource` optional parameter to specify the authentication resource name configured on the OAM server side.

How to Get a JWT User Token

```
$ curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ==='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &username=user123
    &password=passwordAbc12323
    &oracle_requested_assertions=
   oracle-idm%3A%2Foauth%2Fassertion-type%2Fuser-identity%2Fjwt'
```

How to Get a JWT Access Token

```
$ curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &username=user123
    &password=passwordAbc123
    &scope=ConsentManagement.retrieve ConsentManagement.grant
ConsentManagement.revoke'
```

How to Get an OAM User Token and Master Token

```
$ curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &username=user123
    &password=passwordAbc123'
```

Using a Client Credential + oracle_user_credentials Combination

This section documents how to use a client credential together with the `oracle_user_credentials` value to get the following token types: a JWT user token, a JWT access token, an OAM user token and master token, or an OAM access token.

The following topics are covered in this section:

Overview

Requests in this section use the following basic template.

```
curl -i
-H 'Authorization: Basic <sample client ID and password>'
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &oracle_user_credentials=<ORACLE_USER_CREDENTIALS>
    &oracle_requested_assertions=<Oracle_Requested_Assertion_Type>
    &oam_authen_resource=<oam_authen_resource>'
```

Note the following:

- The `oracle_user_credentials` take the following form:

  ```
  {"userid":"user123","password":"password123"}
  ```

  >> Base64 encoded value of JSON data >>

  ```
  eyJ1c2VyaWQiOiJ3ZWJsb2dpYyIsInBhc3N3b3JkIjoid2VsY29tZTEifQ==
  ```

  The actual client ID will be a machine generated GUID.

- You can specify the following assertion types:
  - `oracle-idm%3A%2Foauth%2Fassertion-type%2Fuser-identity%2Foam`
  - `oracle-idm%3A%2Foauth%2Fassertion-type%2Fuser-identity%2Fjwt`
- Use the `oam_authen_resource` optional parameter to specify the authentication resource name configured on the OAM server side.

How to Get a JWT User Token

```
$ curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb211MQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
   &oracle_user_credentials=eyJ1c2VyaWQiOiJ3ZWJsb2dpYyIsInBhc3N3b3JkIjoid2VsY29tZT
EifQ==
   &oracle_requested_assertions=oracle-idm%3A%2Foauth%2Fassertion-type%2F
user-identity%2Fjwt'
```

How to Get a JWT Access Token

```
$ curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
   &oracle_user_credentials=eyJ1c2VyaWQiOiJ3ZWJsb2dpYyIsInBhc3N3b3JkIjoid2VsY29t
ZTEifQ==
   &scope=ConsentManagement.retrieve ConsentManagement.grant
ConsentManagement.revoke'
```

How to Get an OAM User Token and Master Token

```
$ curl -i
-H 'Authorization: Basic NTQzMjFpZDp3ZWxjb21lMQ=='
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
   &oracle_user_credentials=<base64_encoded_credential>
   &client_assertion=<client_jwt_assertion or client_saml2_assertion>
   &client_assertion_type=<client_assertion_type>
   &oracle_requested_assertions=<Oracle_Requested_Assertion_Type>'
```

Using JWT Assertion

This section documents how to use a JWT assertion to get the following token types: a JWT user token, a JWT access token, an OAM user token and master token, or an OAM access token.

The following topics are covered in this section:

Overview

Requests in this section use the following basic template.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host:port/ms_oauth/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
```

How to Get a JWT User Token

```
$ curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &user_oracle_credentials=<base64_encoded_credentials>
    &client_assertion=eyJhbGciOiJSUzUxMiIsjiRZ1_3edKknPTCEtQS79h_44H_8VbGvnA6Dr3M0
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type
%3Ajwt-bearer
    &oracle_requested_assertions=oracle-idm%3A%2Foauth%2Fassertion-type%2F
user-identity%2Fjwt'
```

How to Get a JWT Access Token

```
$ curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials&
    &user_assertion=<JWT User assertion Value>
    &user_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Auser-assertion-type%3Ajwt-bearer
    &client_assertion=eyJhbGciOiJSUzUxMiIsInR5cCI6Ik...j5mZJrfrwxgXxzwVcNbjRgi7uM8
    &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer'
```

How to Get an OAM User Token and Master Token

```
$ curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
```

```
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &user_assertion=<JWT User assertion Value>
    &user_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Auser-assertion-type
%3Ajwt-bearer
   &client_assertion=eyJhbGciOiJSUzUxMiIsInR5cCI6Ik...j5mZJrfrwxgXxzwVcNbjRgi7uM8
   &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
client-assertion-type%3Ajwt-bearer
   &oracle_device_
profile=eyJvcmFjbGU6aWRtOmNsYWltczpjbGllbnQ6c2Rrd...1zOmNvc3ZlcnNpb24iOiI0LjAifQ==
   &oracle_use_server_device_store=true'
```

How to Get an OAM Access Token With an OAM User Token Located in the Server-Side Key Store

```
$ curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_
oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
   &oracle_use_server_device_store=true
   &user_assertion_
type=oracle-idm%3A%2Foauth%2Fassertion-type%2Fuser-identity%2oam
   &client_assertion=eyJhbGciOiJSR5cCI6IkpXVCIsIm...UBaJkagXsLbqb_fNJHqNfwe3QCr7Uk
   &client_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
   &oracle_device_profile=eyJvcmFjbtczpjbGllbnQ6c2Rrdm...pc3ZlcnNpb24iOiI0LjAifQ==
   &scope=oracle.security.oauth.oam.resource_access
   &oracle_oam_application_context=dfsdfsdfsdfsdf
   &oracle_oam_application_
resource=http%3A%2F%2Fhost123.example.com%3A12884%2Findex.html'
```

Using JWT Assertion + PIN

This section documents how to use a JWT user assertion and a PIN (or PIN-like user credential) to get an OAM user token and OAM master token. The client can specify the PIN or passcode value (as an additional credential) together with a JWT user assertion in the request.

The following topics are covered in this section:

Overview

Requests in this section use the following basic template:

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
-request http://host.example.com:14100/ms_oauth/oauth2/
endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &oracle_user_credentials=<Base64 encoded PIN Value>
    &client_assertion=<JWT Client Assertion>
    &client_assertion_type=
urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
    &oracle_user_credentials=<BASE64 ENCODED USER CREDENTIALS>
    &user_assertion_type=
urn%3Aietf%3Aparams%3Aoauth%3Auser-assertion-type%3Ajwt-bearer
    &oracle_device_profile=<BASE64 ENCODING DEVICE PROFILE>'
```

The `oracle_user_credentials` parameter is optional. It is a Base64-encoded value of JSON data that can contain any pair of name and value. For example:

`{"pin":"pinvalue123"}` encodes to `eyJwaW4iOiJwaW52YWx1ZTEyMyJ9`

Response

```
{
  "oracle_aux_tokens":{
    "oam_mt":{
      "oracle_tk_context":"oam_mt",
      "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/oam\/master-token",
      "access_token":""
    }
  },
  "oracle_tk_context":"oam_ut",
  "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/user-token\/oam",
  "access_token":""
}
```

How to Get an OAM User Token and Master Token

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
request http://host.us.example.com:14100/ms_
oauth/oauth2/endpoints/oauthservice/tokens
-d '
     grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &user_assertion_type=
urn%3Aietf%3Aparams%3Aoauth%3Auser-assertion-type%3Ajwt-bearer
```

```
    &oracle_user_credentials=eyJwaW4iOiJwaW52YWx1ZTEyMyJ9
    &client_assertion=eyJhbGciOiJSUzI1NiIs...jOGVj0GXMCA
    &client_assertion_type=
urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
    &oracle_device_profile=ew0KICAgIm9yYWNsZTppZG0...fQ0K
    &user_assertion=eyJhbGciOiJSUzI1NiIsInR5...UyFT7Y9eeo5af4OA
   '
```

Response

```
{
  "oracle_aux_tokens":
   {
    "oam_mt":
     {
      "oracle_tk_context":"oam_mt",
      "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/oam\/master-token",
      "access_token":"VERSION_4%7ELw3jGjxe...F6wouV7ow"
     }
   },
  "oracle_tk_context":"oam_ut",
  "oracle_grant_type":"oracle-idm:\/oauth\/grant-type\/user-token\/oam",
  "access_token":"E6Fyeco+F0GgucHJuLmlkX3R5c...DC0dsLVdJYyJ3Su2xpZWB3"
}
```

Using SAML2 Assertion

This section documents how to use a SAML2 assertion to get the following token types: a JWT user token, a JWT access token, an OAM user token and master token, or an OAM access token.

The following topics are covered in this section:

Overview

Requests in this section use the following basic template.

```
curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host123.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &oracle_user_credentials=<base64_encoded_value>
    &client_assertion=<client_jwt_assertion or client_saml2_assertion>
    &client_assertion_type=<client_assertion_type>
    &oracle_requested_assertions=<Oracle_Requested_Assertion_Type>'
```

How to Get a JWT User Token

```
$ curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_
oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &oracle_user_credentials=<base64_encoded_value>
    &client_assertion=PHNhbWw6QXNzZXJ0aW9uI...2ln%0AbmF0dXJltbDpBc3NlcnRpb24%2B%0A
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type
%3Asaml2-bearer
    &oracle_requested_assertions=oracle-idm%3A%2Foauth%2Fassertion-type
%2Fuser-identity%2Fjwt'
```

How to Get a JWT Access Token

```
$ curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &oracle_user_credentials=<base64_encoded_value>
    &client_assertion=PHNhbWw6QXNzZXJ0aW9...uIHhtbG5zOnNhbWwc3NlcnRpb24%2B%0A
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type
%3Asaml2-bearer&scope=ConsentManagement.retrieve'
```

How to Get an OAM User Token and Master Token

```
$ curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
```

```
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &oracle_user_credentials=<base64_encoded_value>
    &client_assertion=PHNhbWw6QXNzZXJ0aW9uIHhtb9InVyb...2BPC9zYW1sOkF0dHJpYnV0ZT48
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type
%3Asaml2-bearer'
```

Getting OAM Tokens on Mobile Devices

This section documents how to get an OAM user token and master token, or an OAM access token on mobile devices.

The following topics are covered in this section:

How to Request a Verification Code

```
$ curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_
oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=client_credentials
    &oracle_device_
profile=eyJvcmFjbGU6aWRtOmNsYWltczpjbGllbnQ6c2RrdmVyc2l...OmNsaWVudDpvc3ZlcnNpb24i
OiI0LjAifQ==
    &client_id=<MobileAgent1>
    &oracle_requested_assertions=oracle-idm%3A%2Foauth%2Fassertion-type
%2Fclient-identity%2Fmobile-client-pre-authz-code-client'
```

How to Register the Client

```
$ curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_
oauth/oauth2/endpoints/oauthservice/tokens
-d 'grant_type=password&username=userAbc123
    &password=passwordAbc123
    &client_id=<MobileAgent1>
    &oracle_pre_authz_code=eyJhbGci...SsLRxbAt8Yl473vBACuH2Ms2fR_HwhQGVu_zgI3W3a_c
    &oracle_device_profile=eyJvcmFjbGU6aWRtOmNsYWl...G06Y2xhaW1zOmNsaWViI0LjAifQ==
    &oracle_requested_assertions=urn%3Aietf%3Aparams%3Aoauth
%3Aclient-assertion-type%3Ajwt-bearer'
```

How to Get an OAM User Token and Master Token

```
$ curl -i
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth9%2Fgrant-type%2Foam_credentials
    &user_assertion_
type=urn%3Aietf%3Aparams%3Aoauth%3Auser-assertion-type%3Ajwt-bearer
    &client_assertion=eyJhbGciOiJSUzUxMiIsInR5cCI...qwzcgoh5t7sfZInGkbprlA5UswMzqk
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type
%3Ajwt-bearer
    &oracle_device_profile=eyJvcmFjbGU6aWRtOmNsYWltczpjbG...udDnNpb24iOiI0LjAifQ==
    &oracle_use_server_device_store=true'
```

How to Get an OAM Access Token

```
curl -i
```

```
-H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
--request POST http://host.example.com:18001/ms_oauth/oauth2/endpoints/
oauthservice/tokens
-d 'grant_type=oracle-idm%3A%2Foauth%2Fgrant-type%2Foam_credentials
    &client_assertion=eyJhbGciOiJSUzUxMiIs...6NxPv0x_Ng2pEcjVJf42p-tiBFClavI56ycCg
    &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type
%3Ajwt-bearer
    &oracle_device_profile=eyJvcmFjbGU64czpjbGllbnQ6c...ivc3ZlcnNpb24iOiI0LjAifQ==
    &user_assertion_type=oracle-idm%3A%2Foauth%2Fassertion-type%2Fuser-identity
%2Foam
    &scope=oracle.security.oauth.oam.resource_access
    &oracle_oam_application_context=fdsfsdfsdfsdf
    &oracle_oam_application_resource=http%3A%2F%2Fhost123.example.com
%3A12884%2Findex.html
    &oracle_use_server_device_store=true'
```

16

Customizing the OAuth Services

This chapter discusses how to customize Oracle Access Management OAuth Services using the plug-in framework. This chapter includes the following topics:

16.1 Introduction

The Oracle Access Management OAuth Services utilizes a plug-in framework that lets you customize and extend functionality in the following areas using Java interfaces.

- Client Management - This plug-in delegates client authentication, authorization, and profile management to an external module.
- Resource-Server Profile Management - This plug-in delegates resource server authentication, authorization, and profile management to an external module.
- Token Attributes - This plug-in allows administrators to add custom claims to generated access tokens.
- Authorization and Consent Service - This plug-in handles authorization duties during user consent-based authorization.
- Adaptive Access Security - This plug-in integrates adaptive access security applications, such as Oracle Adaptive Access Manager, with OAuth Services.

16.2 Creating a Custom Client Management Plug-in

The client management plug-in delegates the following client functions to an external module:

- client authentication
- client authorization (the evaluation of privileges)
- client profile management (both registration and reading)

An administrator must configure the OAuth client profile before OAuth Services can process an authorization request from the OAuth client. When the request is sent, the client plug-in evaluates which grant types, scopes, and so on the client can access.

You or another OAuth developer can write a custom plug-in implementation to address custom requirements, for example creating a custom client authentication mechanism, or storing OAuth client profiles in an external repository instead of the default client MBean repository, and so on. The Client Management Plug-in consists of the following three Java interfaces:

- Client Authenticator Interface - Verifies an OAuth client credential and redirect URI.
- Client Privilege Interface - Evaluates if an OAuth client has the privileges necessary to use certain OAuth grant types and scopes, as well as the user consent required for the requested scopes. The OAuth Services framework sends the requested parameters along with the client IP address to the plug-in during client privilege checking.
- Dynamic Client Registration Interface - Writes OAuth client profiles to the client repository and retrieves the profiles as needed. This interface has two parts, *the client profile reader* and *the client profile writer*. This interface is consumed by the OAuth service runtime.

16.2.1 The Default Client Management Plug-in Implementation

The default implementation consists of the Client Profile Reader, the Client Authenticator, and the Client Privilege interfaces. OAuth Services invokes the default plug-in implementation during runtime, and the plug-in utilizes the OAuth MBean repository server to read, authenticate, and evaluate client privileges. The OAuth MBean Console and the WLST command-line read and write OAuth client profiles from the OAuth MBean repository using the MBean API.

16.2.2 The Client Runtime Flow

- The client application sends a `client_id` parameter as part of an authorization request and a `client_id+client_secret` Base64 value as an authorization header in its token endpoint requests.
- The plug-in validates the client ID and secret values with the client repository.
- During validation, the plug-in compares the client ID with the client definitions stored in `oauth.xml` (or in LDAP and potentially in other repositories). The secret is validated with CSF.
- If the `client_id` value in the authorization request is *invalid*, the authorization endpoint responds with an `invalid_client` error. The token endpoint validates the Base64 authorization header value.
- If the client credentials *are valid*, then they are embedded in the response along with the authorization code and access tokens. For example, an issued JWT authorization code and access token includes this claim:

  ```
  "oracle.oauth.client_origin_id":"2a180cbc780742698cf51d9a19f80ff6"
  ```

- The plug-in checks if the client is requesting its configured scopes. If not, the plug-in rejects the request and sends an error response.
- The plug-in checks if the client is requesting its configured grant flow. If not, the plug-in rejects the request and sends an error response.

The following example shows the `client_id` and `client_secret` attributes used in an authorization code request:

```
GET /authorize?response_type=code
```

```
&client_id=s6BhdRkqt3&state=xyz
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Freturn
&scope=user_read
```

The following example shows the client ID and secret sent in a Basic authorization header:

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

16.2.3 Deployment Notes

Refer to the following notes when deploying a custom plug-in.

- To deploy the plug-in, copy the JAR file to the following location:

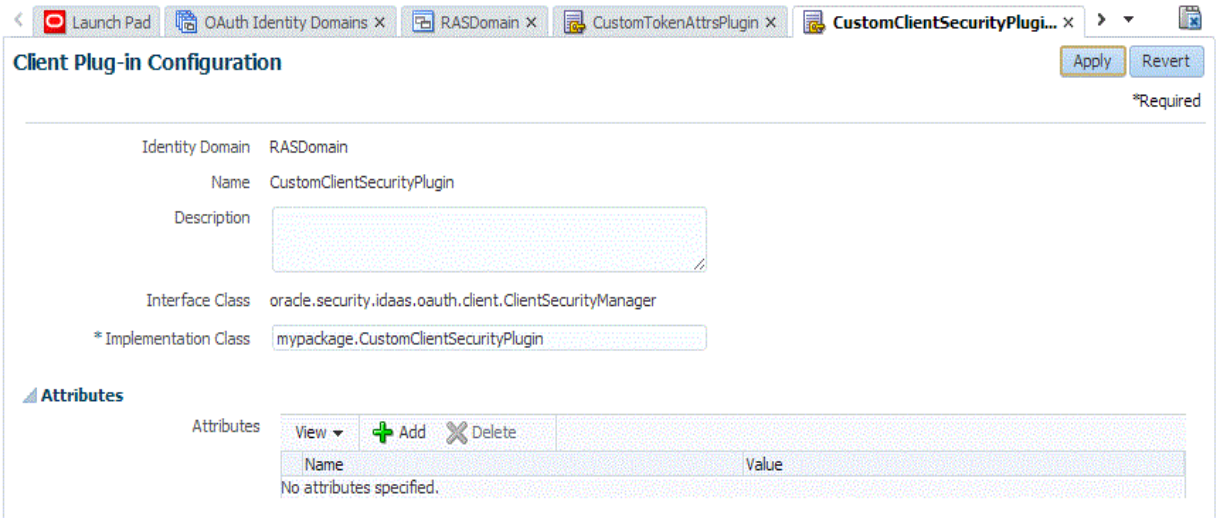
  ```
  $ORACLE_HOME/user_projects/domains/base_domain/config/fmwconfig/oic/plugins/
  ```

- If any third-party libraries were used to develop the custom plug-in, they need to be available in the container (WebLogic or WebSphere) classpath.
- Restart the managed server after deploying the JAR files.

Use the Oracle Access Management Console to create your new client security plug-in.

1. Log in to the OAM console.
2. From the **Launch Pad**, choose **OAuth Services** > *Specific Domain* > **OAuth Plug-ins** > **Client Plug-ins**.
3. Create the new plug-in. Refer to the following screen capture for details.
4. From the **Launch Pad**, choose **OAuth Services** > *Specific Domain* > **OAuth Services Profiles** > *OAuth Service Profile*.

   In the Plug-ins section, assign the client plug-in to the service profile by choosing it from the menu.



16.2.4 Sample Code

```
package mypackage;

import java.util.ArrayList;
import oracle.security.idaas.oauth.client.ClientAuthenticationResponse;
import oracle.security.idaas.oauth.client.ClientAuthorizationResponse;
import oracle.security.idaas.oauth.client.ClientProfile;
import oracle.security.idaas.oauth.client.ClientRequest;
import oracle.security.idaas.oauth.client.ClientScopeProfile;
import oracle.security.idaas.oauth.client.ClientSecurityManager;
import oracle.security.idaas.oauth.client.ClientSecurityManagerException;
import oracle.security.idaas.oauth.client.ClientWritableProfile;
import oracle.security.idaas.oauth.common.appinfra.AppAuthnRequest;
import oracle.security.idaas.oauth.common.provider.exception.OAuthMisconfigurationException;
import java.util.Collection;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.concurrent.locks.ReentrantReadWriteLock;
import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.atomic.AtomicLong;

import oracle.security.idaas.oauth.client.ClientScopeWritableProfile;
import oracle.security.idaas.oauth.client.impl.ClientAuthenticationResponseImpl;
import oracle.security.idaas.oauth.client.impl.ClientAuthorizationResponseImpl;
import oracle.security.idaas.oauth.client.impl.ClientScopeWritableProfileImpl;
import oracle.security.idaas.oauth.client.impl.ClientWritableProfileImpl;
import oracle.security.idaas.oauth.common.Validator;
import oracle.security.idaas.oauth.common.appinfra.AppWritableProfile;
import oracle.security.idaas.oauth.common.appinfra.impl.AppWritableProfileImpl;

public class CustomClientSecurityPlugin implements ClientSecurityManager{


    protected static AtomicLong sequenceNumberGenerator = new AtomicLong(0);
```

```
    protected ReadWriteLock lock = new ReentrantReadWriteLock();

    public CustomClientSecurityPlugin() {
        super();
    }

    public void init(Map<String, Object> config)
            throws OAuthMisconfigurationException {
        System.out.println("CustomClientSecurityPlugin::init()");
    }


    @Override
    public ClientProfile readClientProfile(ClientRequest clientRequest) throws
ClientSecurityManagerException {
        System.out.println("CustomClientSecurityPlugin::readClientProfile()");
        ClientProfile clientProfile = getClientProfile( clientRequest.getAppId() );
        if ( clientProfile != null && clientRequest.getAppId().equals( clientProfile.getAppId()) )
          return clientProfile;
        else {
          System.out.println("CustomClientSecurityPlugin::readClientProfile():  No Client Profile
found.");
          return null;
        }
    }


    public boolean isValidConfidentialSecret(AppAuthnRequest appAuthnRequest) {
        System.out.println("CustomClientSecurityPlugin::isValidConfidentialSecret()");
        boolean result = false;
        ClientProfile clientProfile = getClientProfile(appAuthnRequest.getAppId());
        if ( clientProfile != null && compareChars(appAuthnRequest.getSecret(),
clientProfile.getAppSecret().getSecret()) )
          result = true;

        return result;
    }

    public boolean isValidRedirectURI(ClientRequest clientRequest) {
        System.out.println("CustomClientSecurityPlugin::isValidRedirectURI()");
        boolean result = false;
        return result;

    }


    @Override
    public boolean isUserConsentRequired(ClientRequest clientRequest) {
        System.out.println("CustomClientSecurityPlugin::isValidRedirectURI()");
        boolean result = false;
        ClientProfile clientProfile = getClientProfile(clientRequest.getAppId());
        if( clientProfile != null )
          result = clientProfile.getClientScopeProfile().isUserConsentRequired();
        return result;
    }

    @Override
    public void destroy() {
    }
```

```
    @Override
    public Collection<ClientProfile> readClientProfiles(ClientRequest clientRequest) throws
ClientSecurityManagerException {
        System.out.println("CustomClientSecurityPlugin::readClientProfiles()");
        Collection<ClientProfile> clientProfiles = new HashSet<ClientProfile>();
        return clientProfiles;
    }

    @Override
    public ClientProfile write(ClientWritableProfile clientWritableProfile) throws
ClientSecurityManagerException {
        throw new UnsupportedOperationException();
    }

    @Override
    public ClientProfile update(ClientWritableProfile clientWritableProfile) throws
ClientSecurityManagerException {
        throw new UnsupportedOperationException();
    }

    @Override
    public void delete(ClientWritableProfile clientWritableProfile) throws
ClientSecurityManagerException {
        throw new UnsupportedOperationException();
    }

    /**
     * This methods does perform client authentication against XML repository
     * This method returns only authentication result (true/false) to the IDM OAuth framework in R2
PS2.
     * In future, this method may send error message if authentication is failed and other
additional attributes
     * which may be used for client authorization and authorization plugin
     * @param clientRequest contains requested client id, secret, ip address and requested map
     * @return ClientAuthenticationResponse contains authentication result, error message and
additional attributes
     * @throws ClientSecurityManagerException
     */
    @Override
    public ClientAuthenticationResponse authenticate(ClientRequest clientRequest) throws
ClientSecurityManagerException {
        ClientAuthenticationResponse clientAuthenticationResponse = new
ClientAuthenticationResponseImpl();
        clientAuthenticationResponse.setResult(isValidConfidentialSecret(clientRequest));
        return clientAuthenticationResponse;
    }

    @Override
    public ClientAuthorizationResponse isAuthorized(ClientRequest clientRequest) throws
ClientSecurityManagerException {
        ClientAuthorizationResponse clientAuthorizationResponse = new
ClientAuthorizationResponseImpl();

        if (allowToUseScopes(clientRequest) &&  allowToUseGrantTypes(clientRequest)) {
            clientAuthorizationResponse.setResult(true);
        } else {
            clientAuthorizationResponse.setResult(false);
        }

        return clientAuthorizationResponse;
```

```
    }

    private ClientProfile getClientProfile(String appid) {
      List<ClientProfile> CProfiles = getClientProfiles();
      for (ClientProfile profile: CProfiles) {
        if (appid.equals( profile.getAppId()))
          return profile;
      }
      return null;
    }

     private List<ClientProfile> getClientProfiles() {
       List<ClientProfile> CProfiles = new ArrayList<ClientProfile>();

       ClientWritableProfile clientWritableProfile = new ClientWritableProfileImpl();
       List<String> grantTypes = new ArrayList<String>();
       grantTypes.add( Validator.OAUTH_STD_GRANT_TYPE_CLIENT_CRED );
       grantTypes.add( Validator.OAUTH_STD_GRANT_TYPE_RESOURCE_OWNER_PW_CRED );
       grantTypes.add( Validator.OAUTH_GRANT_TYPE_JWT_BEARER );
       grantTypes.add( Validator.OAUTH_GRANT_TYPE_SAML2_BEARER );

       AppWritableProfileImpl appWritableProfileImpl = new AppWritableProfileImpl();
       AppWritableProfile.AppWritableSecret appWritableSecret = appWritableProfileImpl.new
AppWritableSecretImpl();
       appWritableSecret.setSecret("welcome1".toCharArray());

       ClientScopeWritableProfile clientScopeWritableProfile = new
ClientScopeWritableProfileImpl();
       clientScopeWritableProfile.setAnyScopeAllowed(false);
       clientScopeWritableProfile.setUserConsentRequired(false);
       clientScopeWritableProfile.setAllowedResourceServers(Collections.<String>emptySet());
       clientScopeWritableProfile.addAllowedScope("DocResourceServiceProfile.ALL");
       clientScopeWritableProfile.addAllowedScope("MessagingResourceServiceProfile.ALL");

       clientWritableProfile.setAllowedGrantTypes(grantTypes);
       clientWritableProfile.setAppId( "f35eed9e0cb3471bbd5a6a19919c7a78");
       clientWritableProfile.setAppSecret(appWritableSecret);
       clientWritableProfile.setClientScopeProfile(clientScopeWritableProfile);
       clientWritableProfile.setClientType(ClientProfile.ClientType.CONFIDENTIAL_CLIENT);
       clientWritableProfile.setSequenceNumber( (new
Long(sequenceNumberGenerator.get()).longValue() ));

       CProfiles.add(  clientWritableProfile );

       clientWritableProfile = new ClientWritableProfileImpl();
       clientScopeWritableProfile = new ClientScopeWritableProfileImpl();
       clientScopeWritableProfile.setAnyScopeAllowed(false);
       clientScopeWritableProfile.setUserConsentRequired(false);
       clientScopeWritableProfile.setAllowedResourceServers(Collections.<String>emptySet());
       clientScopeWritableProfile.addAllowedScope("DocResourceServer.ALL");

       clientWritableProfile.setAllowedGrantTypes(grantTypes);
       clientWritableProfile.setAppId( "a0709401479645c2923142e04dbd483f");
       clientWritableProfile.setAppSecret(appWritableSecret);
       clientWritableProfile.setClientScopeProfile(clientScopeWritableProfile);
       clientWritableProfile.setClientType(ClientProfile.ClientType.CONFIDENTIAL_CLIENT);
       clientWritableProfile.setSequenceNumber( (new
Long(sequenceNumberGenerator.get()).longValue() ));
```

```
       CProfiles.add(  clientWritableProfile );
       return CProfiles;

     }

    private  boolean compareChars(char source[], char destion[]) {
        return new String(source).equals(new String(destion));
    }

    private boolean allowToUseScopes(ClientRequest clientRequest)
    throws ClientSecurityManagerException {
        final String sourceMethod = "allowToUseScopes";
        boolean result = false;

        ClientProfile clientProfile = getClientProfile( clientRequest.getAppId() );
        if (clientProfile == null)
          return result;

        ClientScopeProfile clientScopeProfile = clientProfile.getClientScopeProfile();
        if (clientScopeProfile != null && !clientRequest.getScopes().isEmpty()) {
            if (clientScopeProfile.isAnyScopeAllowed()) {
                result = true;
            } else if
(clientScopeProfile.getAllowedScopes().containsAll(clientRequest.getScopes())) {
                result = true;
            } else {
                //TODO:  verify resource server level scopes
            }
        } else {

            //some case scope is not in a part of request
            //for example create UT and CT creation (Identity domain)
            return true;
        }
        return result;
    }

    private boolean allowToUseGrantTypes(ClientRequest clientRequest)
    throws ClientSecurityManagerException {
        final String sourceMethod = "allowToUseGrantTypes";
        boolean result = false;

        ClientProfile clientProfile = getClientProfile( clientRequest.getAppId() );
        if( clientProfile == null )
          return result;

        Collection<String> allowedGrantTypes = clientProfile.getAllowedGrantTypes();
        if (allowedGrantTypes != null) {
            if (!clientRequest.getGrantTypes().isEmpty() &&
allowedGrantTypes.containsAll(clientRequest.getGrantTypes())) {
                result = true;
            }
        }
        return result;

    }
}
```

16.3 Creating a Custom Resource Server Profile-Management Plug-in

The resource server profile-management plug-in delegates the following functions to an external module:

- resource server profile management

The plug-in reads and writes OAuth resource server profiles from the resource server repository. This plug-in is used to read and write OAuth resource server profiles from the resource server repository. This plug-in consists of two Java interfaces. One interface is a resource profile reader, and the other is a resource profile-writer interface. This plug-in is consumed by the OAuth runtime server.

16.3.1 The Default Resource Server Profile-Management Plug-in Implementation

The default plug-in implements the Resource Server Profile Reader interface. The plug-in reads profiles from the OAuth MBean repository. An OAuth administrator can use either the OAM console or the WLST command-line to read and write OAuth Resource Server profiles. (Both the console and the command line use the MBean API to interact with the OAuth MBean repository.)

16.3.2 Resource Server Usage and Validation

- Resource servers are identified through the usage of scopes in the authorization request. The client application sends the `scope` parameter as part of an authorization request.
- The scope parameter value is compared with the resource and scope value stored in the repository. The definitions are stored in `oauth.xml`.
- If the scope parameter value is *invalid*, an `invalid_scope` error response is sent to the client application.
- If the scope parameter value is *valid*, then it gets embedded in the response along with the authorization code and access tokens. For example, an issued JWT authorization code and access token includes this claim:

  ```
  "oracle.oauth.scope":"resumes.position.pmts"
  ```

- When the client application accesses the resource with an access token, the resource server can either validate locally, or it can check with the OAuth Service whether an access token for a given scope is valid before allowing access. The resource server sends a resource server ID and a resource server secret as an authorization header Base64 value. It also sends these parameters as POST body of the validate access token call:
  - `grant_type`. For example:

    ```
    grant_type=oracle-idm:/oauth/grant-type/resource-access-token/jwt
    ```

  - `oracle_token_action`. For example:

    ```
    oracle_token_action=validate
    ```

  - `scope`. For example:

    ```
    scope=resumes.position.pmts
    ```

  - `assertion`. For example:

    ```
    assertion=accessToken-value
    ```

- Finally, the OAuth service validates the access token and sends a JSON response that indicates successful or invalid scope usage.

16.3.3 Development and Deployment Notes

Refer to the following notes when deploying a custom plug-in.

- To deploy the plug-in, copy the JAR file to the following location:

```
$ORACLE_HOME/user_projects/domains/base_domain/config/fmwconfig/oic/plugins/
```

- If any third-party libraries were used to develop the custom plug-in, they need to be available in the container (WebLogic or WebSphere) classpath.
- Restart the managed server after deploying the JAR files.

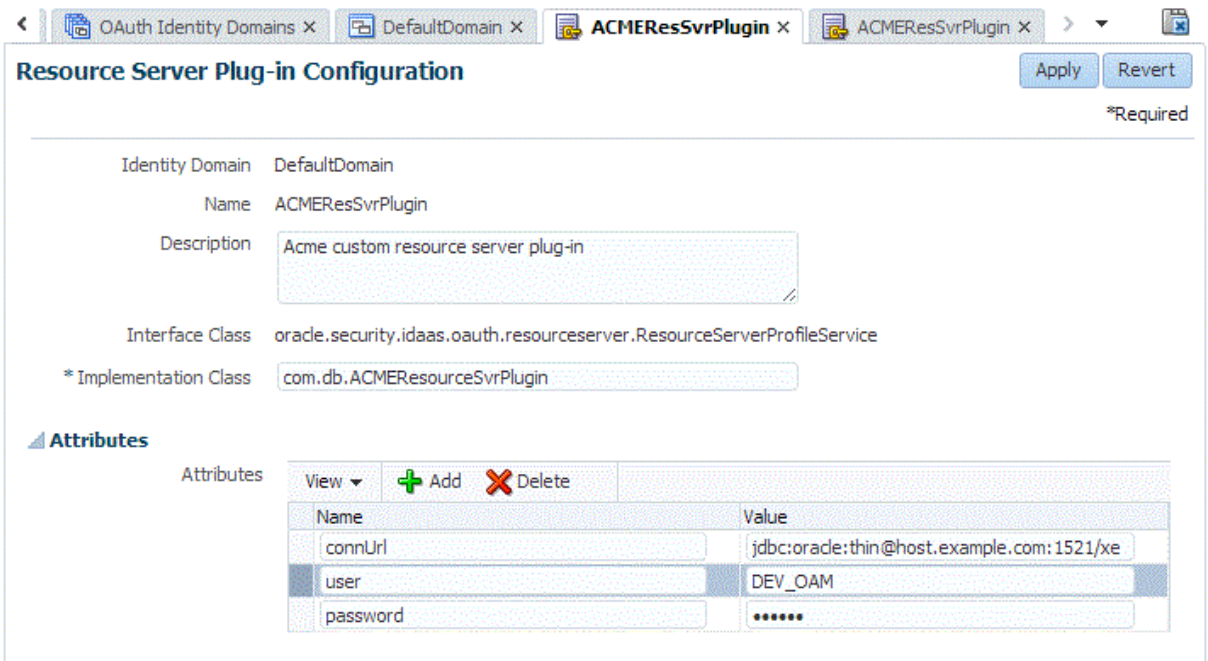
Use the OAM console to create your new client security plug-in.

1. Log in to the OAM console.
2. From the **Launch Pad**, choose **OAuth Service** > *Specific Domain* > **OAuth Plug-ins** > **Resource Server Profile Plug-ins**.
3. Create the new plug-in.

   Refer to the screen capture for details.
4. From the **Launch Pad**, choose **OAuth Service** > *Specific Domain* > **OAuth Service Profiles** > *OAuth Service Profile*.

   In the Plug-ins section, assign the resource server profile plug-in to the service profile by choosing it from the menu.

   Make sure the **Allow clients access to all resource servers** option (under **Custom Resource Servers**) is enabled.



Note: The password field is shown for demonstration purposes. Use other literals instead. If you use password or secret literals, the system stores their value in the deployment's credential storage where third-party plug-in code cannot retrieve them.

Refer to the following notes when developing a custom plug-in.

- The following JAR files are needed for plug-in development and compilation:
  - `oauth_common.jar`
  - `oic_common.jar`
  - `ms_oauth.jar`
- Pay attention to collection usage with regards to scopes. The framework uses List and HashSet types, so the implementation should not do any casting. Otherwise get ClassCast exceptions.
- The scope description retrieval expects the locale to be set in the plug-in implementation, otherwise a null pointer exception occurs. You can modify this by changing the `if` condition as follows.

```
@Override
public ScopeDescriptionProfile getScopeDescription(Locale locale) {
   if (this.scopeDescriptionProfiles != null) {
      for (ScopeDescriptionProfile scopeDescriptionProfile : scopeDescriptionProfiles) {
         if (scopeDescriptionProfile.getLocale().equals(locale)) {
             return scopeDescriptionProfile;
}
```

16.3.4 Sample Code

```
package com.db;

import oracle.security.idaas.oauth.common.provider.exception.OAuthMisconfigurationException;
import oracle.security.idaas.oauth.resourceserver.*;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

/**
 * Custom resource plug-in sample. This plug-in demonstrates the development of a custom plug-in
 * for resource server profile service. In this specific sample, resource server profiles are
 * defined in a database. So the logic typically revolves around CRUD operations, and in this
 * specific sample various retrieval methods are implemented, which get used by the OAuth server
 * runtime.
 */

public class ACMEResourceSvrPlugin implements ResourceServerProfileService {
    //plugin config
    Map<String, Object> pluginConfig = new HashMap<String, Object>();

    // database util gets used for CRUD operations
    DBUtil dbUtil = new DBUtil();

    @Override
    public void init(Map<String, Object> pluginAttrs) throws OAuthMisconfigurationException {
        //initialize plugin config as set
```

```
        pluginConfig = pluginAttrs; //

    }

    @Override
    public void destroy() {
        // destroy plugin config
        pluginConfig.clear();
    }

    @Override
    public ResourceServerProfile readResourceServerProfile(ResourceServerProfileRequest
resourceServerProfileRequest)
            throws ResourceServerProfileNotFoundException {
        // get resource server profile based on resource server name
        return dbUtil.getResSvrByName(resourceServerProfileRequest.getAppName(),
                pluginConfig);
    }

    @Override
    public Collection<ResourceServerProfile>
readResourceServerProfiles(ResourceServerProfileRequest resourceServerProfileRequest)
            throws ResourceServerProfileNotFoundException {
        // get all resource server profiles
        return dbUtil.getAllResSvrs(pluginConfig);
    }

    @Override
    public Collection<ResourceServerProfile>
readResourceServerProfileByRequestedScope(ResourceServerProfileRequest
resourceServerProfileRequest)
            throws ResourceServerProfileServiceException {
        // get all resource server profiles based on requested scopes
        return dbUtil.searchWithScopesList(resourceServerProfileRequest.getRequestedScopes(),
                pluginConfig);
    }

    @Override
    public ResourceServerProfile write(ResourceServerWritableProfile resourceServerWritableProfile)
            throws ResourceServerProfileNameAlreadyUsedException,
ResourceServerProfileIdAlreadyUsedException, ResourceServerProfileServiceException {
        //not implemented
        throw new UnsupportedOperationException();
    }

    @Override
    public ResourceServerProfile update(ResourceServerWritableProfile
resourceServerWritableProfile)
            throws ResourceServerProfileNotFoundException, ResourceServerProfileServiceException {
        // not implemented
        throw new UnsupportedOperationException();
    }

    @Override
    public void delete(ResourceServerProfileRequest resourceServerProfileRequest)
            throws ResourceServerProfileNotFoundException, ResourceServerProfileServiceException {
        // not implemented
        throw new UnsupportedOperationException();
    }
}
```

DBUtil.java Code Sample

```
package com.db;

import oracle.security.idaas.oauth.common.appinfra.AppWritableProfile;
import oracle.security.idaas.oauth.common.appinfra.impl.AppWritableProfileImpl;
import oracle.security.idaas.oauth.resourceserver.*;
import oracle.security.idaas.oauth.resourceserver.impl.*;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.*;

/**
 * Custom resource plug-in sample. This is a utility class which interacts with the database
 * for doing CRUD operations. This specific sample shows how various retrieval methods are
 * implemented.
 */
public class DBUtil {
    private final static String CONN_URL = "connUrl";
    private final static String CONN_USER = "user";
    private final static String CONN_USER_PWD = "password";

    /**
     * Get resource profile by resource server profile name
     * @param resName
     * @param pluginConfig
     * @return
     * @throws ResourceServerProfileNotFoundException
     */
    public ResourceServerProfile getResSvrByName(String resName,
                                                 Map<String, Object> pluginConfig)
            throws ResourceServerProfileNotFoundException {
        Connection dbcon;
        ResourceServerWritableProfile resProfile = null;
        try {
            dbcon = DriverManager.getConnection((String) pluginConfig.get(CONN_URL),
                    (String) pluginConfig.get(CONN_USER),
                    (String) pluginConfig.get(CONN_USER_PWD));
            Statement stmt = dbcon.createStatement();
            System.out.println("searchWithScopesList: BEFORE LIST");
            String resQuery = "select NAME ," +
                    "      AUTHZ_PLUGIN_REF ," +
                    "      SCOPE_PREFIX ," +
                    "      REFRESH_TOKEN_EXPIRY_TIME ," +
                    "      ACCESS_TOKEN_OVERRIDDEN ," +
                    "      DOMAIN_UUID ," +
                    "      AUD_CLAIM ," +
                    "      ALLOW_TOKEN_ATTR_RETRIEVAL ," +
                    "      LAST_UPDATE_TIME ," +
                    "      ID ," +
                    "      DESCR ," +
                    "      ACCESS_TOKEN_EXPIRY_TIME ," +
                    "      OFFLINE_SCOPE_NAME," +
                    "      OVERRIDE_TOKEN_SETTINGS" +
                    " from oauth_resource_svr where name = '" + resName + "'" ;
            ResultSet rslt = stmt.executeQuery(resQuery);

            if (rslt.next()) {
```

```
                resProfile = new ResourceServerWritableProfileImpl();
                System.out.println("searchWithScopesList: RES: " + rslt.getString("id") + "\t" +
rslt.getString("name"));
                String resId = rslt.getString("id");
                populateResSvrPrimaryData(rslt, resProfile);
                //get scopes data  and populate resource profile
                populateScopesData(resId, stmt, resProfile);

                //get static and dynamic attributes data and populate them
                populateTokenCustomAttributes(resId, stmt, resProfile);

                //get resource attributes data
                populateResSvrAttributes(resId, stmt, resProfile);

            }
            dbcon.close();


        } catch (Exception ex) {
            System.out.println(ex);
        }

        if (resProfile == null) {
            throw new ResourceServerProfileNotFoundException("Invalid resource name: " + resName );
        }
        return resProfile;
    }

    /**
     * Get all resource server profiles.
     * @param pluginConfig
     * @return
     * @throws ResourceServerProfileNotFoundException
     */
    public Collection<ResourceServerProfile> getAllResSvrs(Map<String, Object> pluginConfig)
            throws ResourceServerProfileNotFoundException {
        Connection dbcon;
        List<ResourceServerProfile> listResources =
                new ArrayList<ResourceServerProfile>();
        try {
            dbcon = DriverManager.getConnection((String) pluginConfig.get(CONN_URL),
                    (String) pluginConfig.get(CONN_USER),
                    (String) pluginConfig.get(CONN_USER_PWD));
            Statement stmt = dbcon.createStatement();
            System.out.println("searchWithScopesList: BEFORE LIST");
            String resQuery = "select NAME ," +
                    "      AUTHZ_PLUGIN_REF ," +
                    "      SCOPE_PREFIX ," +
                    "      REFRESH_TOKEN_EXPIRY_TIME ," +
                    "      ACCESS_TOKEN_OVERRIDDEN ," +
                    "      DOMAIN_UUID ," +
                    "      AUD_CLAIM ," +
                    "      ALLOW_TOKEN_ATTR_RETRIEVAL ," +
                    "      LAST_UPDATE_TIME ," +
                    "      ID ," +
                    "      DESCR ," +
                    "      ACCESS_TOKEN_EXPIRY_TIME ," +
                    "      OFFLINE_SCOPE_NAME," +
                    "      OVERRIDE_TOKEN_SETTINGS" +
                    " from oauth_resource_svr " ;
```

```
            ResultSet rslt = stmt.executeQuery(resQuery);
            while (rslt.next()) {
                ResourceServerWritableProfile testProfile = new
ResourceServerWritableProfileImpl();
                System.out.println("searchWithScopesList: RES: " + rslt.getString("id") + "\t" +
rslt.getString("name"));
                String resId = rslt.getString("id");
                populateResSvrPrimaryData(rslt, testProfile);
                //get scopes data  and populate resource profile
                populateScopesData(resId, stmt, testProfile);

                //get static and dynamic attributes data  and populate them
                populateTokenCustomAttributes(resId, stmt, testProfile);

                //get resource attributes data
                populateResSvrAttributes(resId, stmt, testProfile);
                listResources.add(testProfile);
            }
            dbcon.close();


        } catch (Exception ex) {
            System.out.println(ex);
        }

        if (listResources.isEmpty()) {
            throw new ResourceServerProfileNotFoundException("Not able to retrieve any resources,
may be resource table has no data");
        }
        return listResources;
    }

    /**
     * Get resource server profiles based on requested scopes.
     * @param scopeList
     * @param pluginConfig
     * @return
     * @throws ResourceServerProfileNotFoundException
     */
    public Collection<ResourceServerProfile> searchWithScopesList(Collection<String> scopeList,
                                    Map<String, Object> pluginConfig) throws
ResourceServerProfileNotFoundException {
        Connection dbcon;
        List<ResourceServerProfile> listResources =
                new ArrayList<ResourceServerProfile>();
        try {
            dbcon = DriverManager.getConnection((String) pluginConfig.get(CONN_URL),
                    (String) pluginConfig.get(CONN_USER),
                    (String) pluginConfig.get(CONN_USER_PWD));
            Statement stmt = dbcon.createStatement();
            System.out.println("searchWithScopesList: BEFORE LIST");
            String scopeQueryPat = "select id from oauth_resource_svr_scope where name in (%s)";
            String scopeQuery = String.format(scopeQueryPat, prepareInQueryPart(scopeList));
            System.out.println("searchWithScopesList: scopeQuery :" + scopeQuery);
            String resQuery = "select NAME ," +
                    "      AUTHZ_PLUGIN_REF ," +
                    "      SCOPE_PREFIX ," +
                    "      REFRESH_TOKEN_EXPIRY_TIME ," +
                    "      ACCESS_TOKEN_OVERRIDDEN ," +
                    "      DOMAIN_UUID ," +
```

```
                    "      AUD_CLAIM ," +
                    "      ALLOW_TOKEN_ATTR_RETRIEVAL ," +
                    "      LAST_UPDATE_TIME ," +
                    "      ID ," +
                    "      DESCR ," +
                    "      ACCESS_TOKEN_EXPIRY_TIME ," +
                    "      OFFLINE_SCOPE_NAME," +
                    "      OVERRIDE_TOKEN_SETTINGS" +
                    " from oauth_resource_svr where id in (" + scopeQuery + ")" ;
            ResultSet rslt = stmt.executeQuery(resQuery);
            while (rslt.next()) {
                ResourceServerWritableProfile testProfile = new
ResourceServerWritableProfileImpl();
                System.out.println("searchWithScopesList: RES: " + rslt.getString("id") + "\t" +
rslt.getString("name"));
                String resId = rslt.getString("id");
                populateResSvrPrimaryData(rslt, testProfile);
                //get scopes data  and populate resource profile
                populateScopesData(resId, stmt, testProfile);

                //get static and dynamic attributes data  and populate them
                populateTokenCustomAttributes(resId, stmt, testProfile);

                //get resource attributes data
                populateResSvrAttributes(resId, stmt, testProfile);
                listResources.add(testProfile);
            }
            dbcon.close();


        } catch (Exception ex) {
            System.out.println(ex);
        }
        if (listResources.isEmpty()) {
            throw new ResourceServerProfileNotFoundException("Invalid scopes: " + scopeList );
        }
        return listResources;
    }

    /**
     * Populates the resource server profile writable instance with data retrieved from the
     * database
     * @param rslt
     * @param resourceServerWritableProfile
     */
    private static void populateResSvrPrimaryData(ResultSet rslt,
                                                  ResourceServerWritableProfile
resourceServerWritableProfile) {
        try {
            resourceServerWritableProfile.setAppId(rslt.getString("ID"));
            resourceServerWritableProfile.setAppName(rslt.getString("NAME"));
            resourceServerWritableProfile.setIdentityDomainUUID(rslt.getString("DOMAIN_UUID"));
            resourceServerWritableProfile.setAudienceClaimValue(rslt.getString("AUD_CLAIM"));
            resourceServerWritableProfile.setAuthzUserConsentPluginRef(rslt.getString("AUTHZ_
PLUGIN_REF"));
            resourceServerWritableProfile.setOfflineScopeName(rslt.getString("OFFLINE_SCOPE_
NAME"));
            resourceServerWritableProfile.setScopeNamespacePrefix(rslt.getString("SCOPE_PREFIX"));

            if ("Y".equalsIgnoreCase(rslt.getString("OVERRIDE_TOKEN_SETTINGS"))) {
```

```
                Long rtExp =  rslt.getLong("REFRESH_TOKEN_EXPIRY_TIME");
                if (rtExp != null ) {
                    System.out.println("RES: REFRESH_TOKEN_EXPIRY_TIME " + rslt.getLong("REFRESH_
TOKEN_EXPIRY_TIME"));

                    OAuthRefreshableTokenWritableProfile oAuthRefreshableTokenWritableProfile
                        = new OAuthRefreshableTokenWritableProfileImpl();
                    oAuthRefreshableTokenWritableProfile.setRefreshTokenExpiresIn(rtExp);

resourceServerWritableProfile.setOverriddenAccessTokenProfile(oAuthRefreshableTokenWritableProfile)
;
                }

                Long atExp = rslt.getLong("ACCESS_TOKEN_EXPIRY_TIME");
                if ( atExp != null ) {
                    OAuthTokenWritableProfile oAuthTokenWritableProfile = new
OAuthTokenWritableProfileImpl();
                    oAuthTokenWritableProfile.setExpiresIn(atExp);

resourceServerWritableProfile.setOverriddenAuthzCodeTokenProfile(oAuthTokenWritableProfile);
                }
            }

            AppWritableProfile.AllowedTokenAttributesRetrievalWritableProfile
allowedTokenAttributesRetrievalWritableProfile =
                new AppWritableProfileImpl().new
AllowedTokenAttributesRetrievalWritableProfileImpl();
            if ("Y".equalsIgnoreCase(rslt.getString("ALLOW_TOKEN_ATTR_RETRIEVAL"))) {

allowedTokenAttributesRetrievalWritableProfile.setAllTokenAttributesRetrievalAllowed(true);
            } else {

allowedTokenAttributesRetrievalWritableProfile.setAllTokenAttributesRetrievalAllowed(false);
            }

resourceServerWritableProfile.setAllowedTokenAttributesRetrieval(allowedTokenAttributesRetrievalWri
tableProfile);
        } catch (Exception e) {
            System.out.println("ACMEResourceSvrPlugin: DBUtil: populateResSvrPrimaryData: " + e) ;
        }
    }

    /**
     * Populates resource server profile writable instance with scopes data.
     * @param resId
     * @param stmt
     * @param resourceServerWritableProfile
     */
    private static void populateScopesData(String resId,
                                           Statement stmt,
                                           ResourceServerWritableProfile
resourceServerWritableProfile)   {
        try {
            String scopeQuery1 = "select name, " +
                "descr, " +
                "user_consent_required " +
                "from OAUTH_RESOURCE_SVR_SCOPE where id='" + resId + "'";
            ResultSet scopeRslt = stmt.executeQuery(scopeQuery1);
            while (scopeRslt.next()) {
```

```
                System.out.println("RES SCOPES: " +scopeRslt.getString("name") + "\t" +
scopeRslt.getString("descr"));
                ScopeWritableProfile scopeProfile = new ScopeWritableProfileImpl();
                //set scope value
                scopeProfile.setName(scopeRslt.getString("name"));
               //set user consent required flag
               if ("Y".equalsIgnoreCase(scopeRslt.getString("user_consent_required"))) {
                   scopeProfile.setUserConsentRequired(true);
               } else {
                   scopeProfile.setUserConsentRequired(false);
               }

               // set scope description
               if (scopeRslt.getString("descr") != null) {
                   ScopeWritableProfile.ScopeDescriptionWritableProfile
scopeDescriptionWritableProfile = new
ScopeWritableProfileImpl.ScopeDescriptionWritableProfileImpl();
                   scopeDescriptionWritableProfile.setDescription(scopeRslt.getString("descr"));
                   scopeDescriptionWritableProfile.setLocale(Locale.ENGLISH);
                   scopeProfile.addScopeDescription(scopeDescriptionWritableProfile);
               }
               resourceServerWritableProfile.addScopeProfile(scopeProfile);
           }
       } catch (Exception e) {

           System.out.println("ACMEResourceSvrPlugin: DBUtil: populateScopesData: " + e) ;
        }
    }

    /**
     * Populates resource server profile writable instance with access token's custom attributes
     * data.
     * @param resId
     * @param stmt
     * @param resourceServerWritableProfile
     */
    private static void populateTokenCustomAttributes(String resId,
                                                      Statement stmt,
                                                      ResourceServerWritableProfile
resourceServerWritableProfile) {
        try {
            TokenAttributeWritableProfile tokenAttributeWritableProfile
                    = new TokenAttributeWritableProfileImpl();
            String stQuery = "select name, " +
                    "value " +
                    "from OAUTH_RESOURCE_SVR_AT_STATTRS where id='" + resId + "'";
            ResultSet stRslt = stmt.executeQuery(stQuery);
            while (stRslt.next()) {
                System.out.println("RES STATIC ATTRS: " +  stRslt.getString("name") + "\t" +
stRslt.getString("value"));
                tokenAttributeWritableProfile.addTokenStaticAttribute(stRslt.getString("name"),
                        stRslt.getString("value"));

            }

            //get dynamic attributes data
            String dyQuery = "select name " +
                    "from OAUTH_RESOURCE_SVR_AT_DYNATTRS where id='" + resId + "'";
            ResultSet dyRslt = stmt.executeQuery(dyQuery);
            while (dyRslt.next()) {
```

```
                System.out.println("RES DYN ATTRS: " +dyRslt.getString("name"));
                tokenAttributeWritableProfile.addTokenDynamicAttribute(dyRslt.getString("name"));
            }
            resourceServerWritableProfile.setTokenAttributeProfile(tokenAttributeWritableProfile);
        } catch (Exception e) {

            System.out.println("ACMEResourceSvrPlugin: DBUtil: populateTokenCustomAttributes: " +
e) ;
        }
    }

    /**
     * Populates resource server profile writable instance with additional attributes data.
     * @param resId
     * @param stmt
     * @param resourceServerWritableProfile
     */
    private static void populateResSvrAttributes(String resId,
                                                 Statement stmt,
                                                 ResourceServerWritableProfile
resourceServerWritableProfile) {
        try {
            String attrsQuery = "select name, " +
                    "value " +
                    "from OAUTH_RESOURCE_SVR_ATTRS where id='" + resId + "'";
            ResultSet attrsRslt = stmt.executeQuery(attrsQuery);
            Map<String, String> attrs = new HashMap<String, String>();
            while (attrsRslt.next()) {

                System.out.println("RES ATTRS: " +attrsRslt.getString("name") + "\t" +
attrsRslt.getString("value"));
                attrs.put(attrsRslt.getString("name"), attrsRslt.getString("value"));
            }

            if (!attrs.isEmpty()) {
                resourceServerWritableProfile.setAttributes(attrs);
            }
        } catch (Exception e) {

            System.out.println("ACMEResourceSvrPlugin: DBUtil: populateResSvromAttributes: " + e) ;
        }

    }

    private static String prepareInQueryPart(Collection<String> listStr) {
        StringBuilder builder = new StringBuilder();
        Iterator itr = listStr.iterator();
        for (int i = 0; i < listStr.size();) {
            builder.append("'");
            //builder.append(listStr.get(i));
            builder.append((String)itr.next()) ;
            builder.append("'") ;
            if (++i < listStr.size()) {
                builder.append(",");
            }
        }
        System.out.println("ACMEResourceSvrPlugin: prepareInQueryPart: " + builder.toString()) ;
        return builder.toString();
    }
}
```

16.4 Creating a Custom Token Attributes Plug-in

OAuth Services provides ways for adding custom attributes to access tokens based on server configuration. There are two types of attributes:

- Static attributes - These are defined as attribute name and value pairs. The value is fixed at the time of attribute definition. For example: `name1=value1`.
- Dynamic (user) attributes - These attributes are limited to user profile specific attributes. OAuth Services needs to designate the source of the user profile attributes. The user profile service may be used for attribute name and attribute value retrieval.

Static and dynamic attributes can be defined for two entities in the IDM OAuth token server: (1) OAuth Service Profile (2) Resource Server

- Static attribute names should not conflict with dynamic ones. Because dynamic attribute names are known to the system, name collision will be prevented. However in a corner case where a static attribute is defined first and a related dynamic attribute with the same name is later introduced into the system, the static attribute should be removed first.
- If the same attributes are defined in both of these entities, then the Resource Server based attributes will be taken into consideration for population into an access token.
- Because dynamic attributes are related to users, a user consent page will show (if configured) to request the user's consent. The user acknowledges that the configured attributes will be shared with clients/resources for information purposes.

Custom attributes appear as-is as claims in the access token. OAuth Services issues a JWT-based access token that contains standard JWT claims along with OAuth server-specific ones. Here is a sample set:

- Standard

```
"exp":1357596398000,
"iat":1357589198000,
"aud":"oam_server1",
"iss":"OAuthServiceProfile",
"prn":null,
"jti":"340c8324-e49f-43cb-ba95-837eb419e068",
```

- OAuth server specific

> **Note:** The following may vary slightly based on 2-legged/3-legged and web/mobile scenarios.

```
"oracle.oauth.user_origin_id":"john101",
"oracle.oauth.user_origin_id_type":"LDAP_UID",
"oracle:idm:claims:client:macaddress":"1C:AB:A7:A5:F0:DC",
"oracle.oauth.scope":"brokerage",
"oracle.oauth.client_origin_id":"oauthssoapp1id",
"oracle.oauth.grant_
type":"oracle-idm:/oauth/grant-type/resource-access-token/jwt"
```

The above claims are inherently available as part of an access token generated by the OAuth Services. Because the custom attributes appear as claims in JWT-based access tokens, the following naming restrictions should be followed:

- Avoid JWT standard claim names
- Avoid the Oracle prefix (as shown above)
- Avoid reserved words

When defining attributes, keep in mind that:

- The OAuth Services profile may have 0..n static and dynamic attributes.
- The resource server may have 0..n static and dynamic attributes.
- Each scope may have 0..n static and dynamic attributes.
- The static and dynamic type indicator may be needed for runtime usage.

16.4.1 Deployment Notes

Refer to the following notes when deploying a custom plug-in.

- To deploy the plug-in, copy the JAR file to the following location:

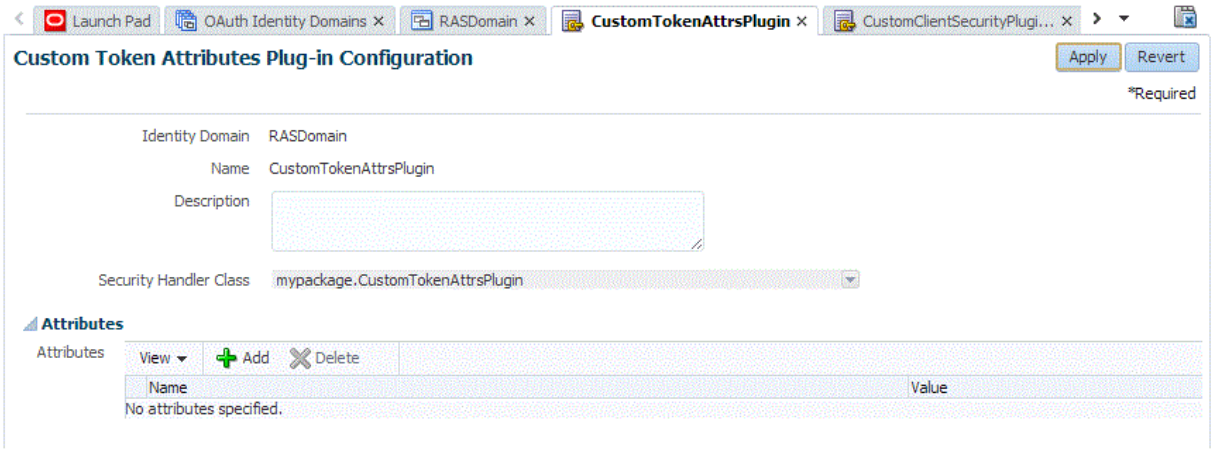
  ```
  $ORACLE_HOME/user_projects/domains/base_domain/config/fmwconfig/oic/plugins/
  ```

- If any third-party libraries were used to develop the custom plug-in, they need to be available in the container (WebLogic or WebSphere) classpath.
- Restart the managed server after deploying the JAR files.

Use the OAM console to create your new client security plug-in.

1. Log in to the OAM console.
2. From the **Launch Pad**, choose **OAuth Service** > *Specific Domain* > **OAuth Plug-ins** > **Custom Token Attributes Plug-ins**.
3. Create the new plug-in. Refer to the following screen capture for details.
4. From the **Launch Pad**, choose **OAuth Service** > *Specific Domain* > **OAuth Service Profiles** > *OAuth Service Profile*.

   In the Plug-ins section, assign the custom token attributes plug-in to the service profile by choosing it from the menu.



16.4.2 Sample Code

```
package mypackage;

import java.util.Map;
import oracle.security.idaas.rest.provider.plugin.HandlerRequest;
import oracle.security.idaas.rest.provider.plugin.HandlerResult;
import oracle.security.idaas.rest.provider.plugin.SecurityHandler;

public class CustomTokenAttrsPlugin implements SecurityHandler {
    public CustomTokenAttrsPlugin() {
        super();
        System.out.println("CustomTokenAttrsPlugin: in the constructor");

    }

    public void init(Map<String, String> config) {
        System.out.println("CustomTokenAttrsPlugin: in the constructor init: " +
config.toString());
    }

    // get token claims from the config and add them to handler result
    public void processSecurityEvent(HandlerRequest req,
                                     HandlerResult result) {

        String svcDynAttr1 = "thirdparty-svc-username";
        String svcStaticAttr1 = "thirdparty-svcs-st1";
        String rsrcDynAttr1 = "thridparty-rs-empnumber";
        String rsrcStaticAttr1 = "thridparty-rs-st1";

        result.addTokenClaim(svcStaticAttr1, "my-svc-st1-val");
        result.addTokenClaim(svcDynAttr1, "my-svc-margchou");
        result.addTokenClaim(rsrcStaticAttr1, "my-rs-st1-val");
        result.addTokenClaim(rsrcDynAttr1, "my-rs-123456");

        result.setResultStatus(HandlerResult.ResultStatus.ALLOW);

    }

    public void destroy() {
        final String METHOD = "destroy()";
    }
```

```
}
```

16.5 Creating a Custom Authorization and Consent Service Plug-in

The Authorization and Consent Service plug-in handles general authorization during user consent-based authorization. This plug-in can also influence claims in a generated token. This plug-in has two parts: *the resource authorization service*, and *the user consent service*.

The Resource Authorization Service

The IDM OAuth Service framework invokes the Resource Authorization Security Handler plug-in prior to processing OAuth access token requests, authorization code requests, and OAuth access token validation requests. The service then returns an authorization decision (that is, *allowed* or *denied*).

When invoking the Resource Authorization Server plug-in, OAuth Services sends the plug-in the following information:

- Requested scopes
- Grant type
- Client IP address
- User and client authentication status
- Client profile and resource server profile

The Resource Server Authorization plug-in uses the requested data and evaluates an authorization decision. In addition, the Resource Authorization Service plug-in may add a permission payload claim into the access token if an authorization decision is allowed.

User Consent Service

The User Consent Service stores, retrieves, and revokes user consent based on user ID, client ID, scope name, OAuth service profile, and identity domain. The User Consent Service is invoked by the OAuth Services framework when processing the authorization code and access token requests for scopes that require consent.

> **Note:** The OAuth administrator can define the resource authorization and user consent plug-in when configuring the OAuth Service Profile. The administrator can also override this plug-in reference by configuring the Resource Server Profile.

This plug-in is invoked during access token creation and authorization code creation for requested scopes. This plug-in cannot be invoked for identity client creation and user token creation (JWT User/Client Assertion).

16.5.1 The Default Resource Authorization and User Consent Services Implementations

There are two different implementations for the Authorization and Consent plug-in: the default authorization with OAuth database consent repository, and the REST callback authorization plug-in (for RAS).

The Default Authorization and Use Consent Plug-in Implementation

The default plug-in implements both the Authorization and user Consent services. User consent is managed using the OAuth database repository. When processing requests for access token creation and validation, the authorization implementation checks if the user has already given consent for the required scopes.

The REST Callback Resource Authorization Plug-in Implementation

This REST callback plug-in implements the Authorization Service interface only. It is used when an external authorization service is deployed for making the authorization decisions. Because user consent is not implemented in this plug-in, 2-legged OAuth is the intended use case. The invocation sequence is as follows:

OAuth Service Framework > REST Callback Resource Authorization Plug-in > (remote) RAS REST Services

This plug-in enables OAuth developers to write a REST implementation for Custom Resource Authorization Service and deploy it on a remote server. An OAuth administrator can define a REST authorization endpoint in the REST Callback Authorization Plug-in configuration using the OAM Console or WLST commands.

Part V

Developing with Identity Federation

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

It contains the following chapters:

17

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:

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

> **See Also:** For more information about using the default user provisioning plug-in, see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

17.1 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.

17.2 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*.

17.3 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 17–1, "Sample UserProvisioning.java"
- Example 17–2, "Sample UserPlugin.xml"
- Example 17–3, "Sample MANIFEST.MF"

Example 17–1 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.example.com:389";
        String ldapUserBaseDN = "dc=iplanet,dc=com";

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

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

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

    }
}
```

Example 17–2 Sample UserPlugin.xml

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

Example 17–3 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
```

17.4 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

17.4.1 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 17.4.2, "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*.

17.4.2 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.

18

Using the REST API for Identity Federation

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

Notes About Using cURL

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

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

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

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

Available Java API References

In addition to this *Oracle Fusion Middleware Developer's Guide for Oracle Access Management*, the *Oracle Fusion Middleware User Provisioning Plug-in Java API Reference for Oracle Access Management Identity Federation* is available.

18.1 Resource URLs

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

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

where:

- **host** is the host where the OAM Admin Service is running.

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

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

The Access Manager identity federation REST Web Application Description Language (WADL) file lists the supported policy resources and methods. The Policy Administration REST WADL document is available at `http://`*`host:port`*`/oam/services/rest/11.1.2.0.0/fed/admin/application.wadl`.

18.2 URL Resources and Supported HTTP Methods

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

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

18.3 Resources Summary

Table 18–1 provides detail about each identity federation resource, the supported HTTP methods, and the results of each action.

Table 18–1 Access Manager Identity Federation Resources Summary

Resource	Method	Description
`/oam/services/rest/11.1.2.0.0/fed/admin/sso`	POST	Enable Federation SSO service on the server and configure the logout done URL.
	PUT	Same as POST operation.
	GET	Get the enable status of the Federation SSO service and the logout done URL
`/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners`	POST	A service provider (SP) or identity provider (IdP) partner resource is created. The request is performed on the resource that is the parent of the object. A partner resource matching the request is created.
`/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/idp`	GET	A list of IdP partners is retrieved by this method.
`/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp`	GET	A list of SP partners is retrieved by this method. The resource that represents the Resource Types object is returned. This representation contains the matching Resource Type resource attributes and their values.
`/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/idp/partnerName`	POST	A specific IdP partner resource is created by this method, where partnerName is the name of the partner to be created.

Table 18–1 (Cont.) Access Manager Identity Federation Resources Summary

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

Table 18–1 (Cont.) Access Manager Identity Federation Resources Summary

Resource	Method	Description
/fedrest/configuresso	POST	Re-directs the respective fedrest url to /oam/services/rest/11.1.2.0.0/fed/admin/ssoservice. This is used to create a local server for local authentication or federation SSO. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services.The input data type is FORM POST data.
/fedrest/createsp	POST	Re-directs the respective fedrest url to /oam/services/rest/11.1.2.0.0/fed/admin/trustedsppartners. This is used to create a specific SP partner resource. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.
/fedrest/createidp	POST	Re-directs the respective fedrest url to /oam/services/rest/11.1.2.0.0/fed/admin/trustedidppartners. This is used to create a specific IdP partner resource. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.
/fedrest/orchestrator	POST	Re-directs the respective fedrest url to /oam/services/rest/11.1.2.0.0/fed/admin/orchestratorservice. This service is used by a client to connect two federation servers to remote REST services. This REST service is published on the Access Management Admin Server for backward compatibility where the OIF 11gR1 server or existing OIF REST clients will connect to those services. The input data type is FORM POST data.

18.4 cURL Command Examples

The following examples are provided as reference.

- Configuring SSO Service using POST cURL Command
- Retrieving SSO Service using GET cURL Command
- Configuring SSO Service using PUT cURL Command
- Creating an SP Partner cURL Command
- Listing all SP Partners cURL Command
- Retrieving SP Partner Data cURL Command
- Updating SP Partner Details cURL Command
- Deleting SP Partner Details cURL Command
- Enabling Test SP using POST cURL Command
- Retrieving Test SP Enablement using GET cURL Command
- Disabling Test SP using PUT cURL Command
- Configuring SSO Service using POST cURL Command using /fedrest/configuresso
- Creating an SP Partner cURL Command using /fedrest/createsp

  Creating an IdP Partner cURL Command using /fedrest/createidp

- Connecting Federation Servers to remote REST services using /fedrest/orchestrator

Configuring SSO Service using POST cURL Command

The REST endpoint`/oam/services/rest/11.1.2.0.0/fed/admin/sso` request is used to configure the SSO service when the customer is the identity provider using the POST method.

This API is used for wiring with Fusion Applications and it configures the FAAuthScheme.

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

- Enable SAML 2.0 only.
- Enable SSO POST, SSO Artifact, SLO Redirect profiles only.
- NameID
  - Email Address with mail as the attribute of the user.
  - Unspecified with uid as the attribute of the user (default).
- One set of keys/certificates for SAML operations.

OAM/Fed will be able to have specific SP Partner configuration:

- SSO binding to be used.
- NameID format and value to be used.
- NameID format and value to be used.
- Extra attributes to be sent
  - NameID value sent as an attribute: SP Partner will indicate the SAML Attribute name, and whether to send user's ID or email address.
  - Static attribute value used by the SP during Assertion mapping operations: SP Partner will indicate the SAML Attribute name and its value.

When IdP needs to authenticate the user, it will redirect the user to an URL protected by WebGate OAM with the FAAuthScheme:

- If OAM is configured for local authentication, FAAuthScheme will instruct OAM to display a login page for the user to enter its credentials.
- If OAM is configured for Federation SSO, FAAuthScheme will instruct OAM to start a Federation SSO flow by redirecting the user to SaaS OIF/SP.
- If OAM is configured to let the user decide how to authenticate, FAAuthScheme will instruct OAM to display a chooser page and to then perform a local authentication or Federation SSO operation, depending on the user's choice.

The following is a sample file for this cURL command

Where

- **ssoFederation**

  is the setting in FAAuthScheme that enables federated SSO for the protected resource.

- ssoFederation = ,

  = ,

  =

- **ssoChooser**

  is the setting that enables the login page to show both federated SSO link and local login with username and password.

- **oamLogoutDoneURL**

  is the URL to redirect after user has been logged out through single logout.

```
curl -X
POST -H "Content-Type: application/json" -d '{"ssoFederation": "true",
"ssoChooser": "false", "oamLogoutDoneURL": "http://test.com/customLogout"}'
http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/sso --user
USER:PASSWORD
```

Return result:

```
{
"status":"1",
"statusMessage":""}
```

Retrieving SSO Service using GET cURL Command

The REST endpoint `/oam/services/rest/11.1.2.0.0/fed/admin/sso` request is used to retrieve the SSO service information when the customer is the identity provider using the GET method. The following is a sample file for this cURL command

```
curl -u USER:PASSWORD --request GET
'http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/sso'
```

Return result:

```
{
"ssoFederation":"true",
"ssoChooser":"false",
"oamLogoutDoneURL":"http://test.com/customLogout"
}
```

Configuring SSO Service using PUT cURL Command

The REST endpoint `/oam/services/rest/11.1.2.0.0/fed/admin/sso` request is used to configure the SSO service when the customer is the identity provider using the PUT method. The following is a sample file for this cURL command

```
curl -X PUT -H "Content-Type: application/json" -d '{"ssoFederation": "false",
"ssoChooser": "false", "oamLogoutDoneURL": ""}'
http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/sso --user
USER:PASSWORD
```

Return result:

```
{
"status":"1",
"statusMessage":""
}
```

Creating an SP Partner cURL Command

The REST endpoint `/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/partnerName` request creates a Trusted Partners Service. The service used is the /trustedpartners/sp/acmeSP service, containing the name of the SP Partner.

The following is a sample file for this cURL command.

Where

- **metadataB64**

  is the hexadecimal string corresponding to base 64 encoding of the peer partner's metadata XML. When using curl, you will have to escape the + signs in the base 64 encoded metadata string.

- **ssoProfile**

  the SAML 2.0 SSO profile to use (artifact or httppost).

- **nameIDFormat**

  the NameID format used during Federation SSO. Possible values are emailaddress or unspecified. If emailaddress, then the NameID value of an Assertion created by the IdP will contain the user's email address; if unspecified, then the NameID value of an Assertion created by the IdP will contain the user's ID.

```
curl -X POST
-H "Content-Type: application/json" -d
'{ "metadataB64": "...", "partnerType": "sp", "partnerName": "acmeSP",
"nameIDFormat": "unspecified", "ssoProfile": "httppost" }'
http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/a
cmeSP --user <USER>:<PASSWORD>
```

Return result:

```
{
"status":"1",
"statusMessage":""
}
```

Listing all SP Partners cURL Command

The REST endpoint `/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp` request retrieves a list of Trusted Partners Services. The following is a sample file for this cURL command.

```
curl -u USER:PASSWORD --request GET
'http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp'
```

Return result:

```
{
"partnerInfoList":
  [
    {
     "metadataB64":"...",
     "partnerName":"acmeSP",
     "nameIDFormat":"unspecified",
     "ssoProfile":"httppost",
     "providerID":"http://acme:7499/fed/sp",
"assertionConsumerURL":"http://acme:7777/fed/sp/sso",
"logoutRequestURL":"http://acme:7777/fed/idp/samlv20",
"logoutResponseURL":"http://acme:7777/fed/idp/samlv20",
     "adminManualCreation":"false",
"displaySigningCertDN":"CN=acme OIF Signing Certificate",
"displaySigningCertIssuerDN":"CN=OIFCert",
"displaySigningCertStart":"2014-10-07T06:32:16-07:00",
"displaySigningCertExpiration":"2024-10-11T06:32:17-07:00",
"displayEncryptionCertDN":" CN=acme OIF Enc Certificate",
"displayEncryptionCertIssuerDN":"CN=OIFCert",
"displayEncryptionCertStart":"2014-10-07T06:32:16-07:00",
"displayEncryptionCertExpiration":"2024-10-11T06:32:17-07:00"
    },
{
     "metadataB64":"...",
     "partnerName":"ciscoSP",
     "nameIDFormat":"emailaddress",
     "ssoProfile":"httppost",
     "providerID":"http://cisco:7499/fed/sp",
"assertionConsumerURL":"http://cisco:7777/fed/sp/sso",
"logoutRequestURL":"http://cisco:7777/fed/idp/samlv20",
"logoutResponseURL":"http://cisco:7777/fed/idp/samlv20",
     "lastNameAttrName":"lastname",
     "firstNameAttrName":"firstname",
     "userNameAttrName":"username",
     "emailAttrName":"email"
     "adminManualCreation":"false",
"displaySigningCertDN":"CN=cisco OIF Signing Certificate",
"displaySigningCertIssuerDN":"CN=OIFCert",
"displaySigningCertStart":"2014-10-07T06:32:16-07:00",
"displaySigningCertExpiration":"2024-10-11T06:32:17-07:00",
"displayEncryptionCertDN":" CN=cisco OIF Enc Certificate",
"displayEncryptionCertIssuerDN":"CN=OIFCert",
"displayEncryptionCertStart":"2014-10-07T06:32:16-07:00",
"displayEncryptionCertExpiration":"2024-10-11T06:32:17-07:00"
     }
  ]
}
```

Retrieving SP Partner Data cURL Command

The REST endpoint /oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/partnerName request retrieves information for a specific Trusted Partners Service. The following is a sample file for this cURL command.

```
curl -u USER:PASSWORD --request GET
'http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/ac
meSP'
```

Return result:

```
{
"metadataB64":"...",
"partnerName":"acmeSP",
"nameIDFormat":"unspecified",
"ssoProfile":"httppost",
"providerID":"http://acme:7499/fed/sp",
"assertionConsumerURL":"http://acme:7777/fed/sp/sso",
"logoutRequestURL":"http://acme:7777/fed/idp/samlv20",
"logoutResponseURL":"http://acme:7777/fed/idp/samlv20",
"adminManualCreation":"false",
"displaySigningCertDN":"CN=acme OIF Signing Certificate",
"displaySigningCertIssuerDN":"CN=OIFCert",
"displaySigningCertStart":"2014-10-07T06:32:16-07:00",
"displaySigningCertExpiration":"2024-10-11T06:32:17-07:00",
"displayEncryptionCertDN":" CN=acme OIF Enc Certificate",
"displayEncryptionCertIssuerDN":"CN=OIFCert",
"displayEncryptionCertStart":"2014-10-07T06:32:16-07:00",
"displayEncryptionCertExpiration":"2024-10-11T06:32:17-07:00"

}
```

Updating SP Partner Details cURL Command

The REST endpoint /oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/partnerName request is used to modify information for a specific Trusted Partners Service. The following is a sample file for this cURL command.

```
curl -X PUT
-H "Content-Type: application/json" -d
'{ "metadataB64": "..." }'
http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/acm
eSP --user USER:PASSWORD
```

Return result:

```
{
"status":"1",
"statusMessage":""
}
```

Deleting SP Partner Details cURL Command

The REST endpoint /oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/partnerName request is used to delete information for a specific Trusted Partners Service. The following is a sample file for this cURL command.

```
curl -u
USER:PASSWORD --request DELETE
'http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/trustedpartners/sp/ac
meSP'
```

Return result:

```
{
"status":"1",
"statusMessage":""
}
```

Enabling Test SP using POST cURL Command

The REST endpoint /oam/services/rest/11.1.2.0.0/fed/admin/testsp service request is used to enable a test SP using the POST method. The following is a sample file for this cURL command.

```
curl -X POST
-H "Content-Type: application/json" -d '{"enabled": "true"}'
http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/testsp --user
USER:PASSWORD
```

Return result:

```
{
"status":"1",
"statusMessage":""
}
```

Retrieving Test SP Enablement using GET cURL Command

The REST endpoint /oam/services/rest/11.1.2.0.0/fed/admin/testsp service request is used to retrieve test SP enablement details using the GET method. The following is a sample file for this cURL command.

```
curl -u USER:PASSWORD --request GET
'http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/testsp'
```

Return result:

```
{
"enabled":"true"
}
```

Disabling Test SP using PUT cURL Command

The REST endpoint /oam/services/rest/11.1.2.0.0/fed/admin/testsp service request is used to disable a test SP using the PUT method. The following is a sample file for this cURL command.

```
curl -X PUT
-H "Content-Type: application/json" -d '{"enabled": "false"}'
http://hostname:7001/oam/services/rest/11.1.2.0.0/fed/admin/testsp --user
USER:PASSWORD
```

Return result:

```
{
"status":"1",
"statusMessage":""
}
```

Configuring SSO Service using POST cURL Command using /fedrest/configuresso

The /fedrest/configuresso request redirects the request url to the actual required url /oam/services/rest/11.1.2.0.0/fed/admin/ssoservice, which is used to configure the SSO service when the customer is the identity provider using the POST method.

The following is a sample file for the cURL command

```
curl -v -i -u <USER>:<PASSWORD>
-X POST -d @ssoConfigureData.in http://<SERVER>:<PORT>/fedrest/configuresso
```

File ssoConfigureData.in:

```
spTenantName=&idpProviderID=&
preverify=false&ssoFederation=true&
ssoChooser=true&oamadminuser=<USER>&
oamadminpassword=<PASSWORD>&
oamadminhost=<SERVER>&
oamadminport=<PORT>

curl -u <USER>:<PASSWORD> --data "spTenantName=""&idpProviderID=""&
preverify="false"&ssoFederation="true"&ssoChooser="true"&
oamadminuser="<USER>"&oamadminpassword="<PASSWORD>"&oamadminhost="<SERVER>"&
oamadminport="<PORT>"&oamLogoutDoneURL=""" --request
POST 'http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/fed/admin/ssoservice';
-X POST -d @ssoConfigureData.in http://<SERVER>:<PORT>/fedrest/configuresso
```

Creating an SP Partner cURL Command using /fedrest/createsp

The /fedrest/createsp request redirects the request url to the actual required url /oam/services/rest/11.1.2.0.0/fed/admin/trustedsppartners, which creates a Trusted Partners Service.

The following is a sample file for this cURL command.

Where

- **ssoProfile**

  the SAML 2.0 SSO profile to use (artifact or httppost).

- **nameIDFormat**

  the NameID format used during Federation SSO. Possible values are emailaddress or unspecified. If emailaddress, then the NameID value of an Assertion created by the IdP will contain the user's email address; if unspecified, then the NameID value of an Assertion created by the IdP will contain the user's ID.

```
curl -v -i -u <USER>:<PASSWORD>
-X POST -d @spCurlData.in http://<HOST>:<PORT>/fedrest/createsp
```

File spCurlData.in:

```
idpTenantName=&idpTenantURL=&
spPartnerName=spPartner-sample&
spProviderID=&metadata=&metadataURL=&
assertionConsumerURL=&logoutRequestURL=&
logoutResponseURL=&signingCert=&encryptionCert=&
nameIDFormat=unspecified&ssoProfile=artifact&
generateNewKeys=&validityNewKeys=&preverify=false&
lastNameAttrName=&firstNameAttrName=&userNameAttrName=&
emailAttrName=&staticAttrName=&staticAttrValue=&customAttrs= ...
```

```
curl -v -i -u <USER>:<PASSWORD>
-X POST -d @spCurlData.in
https://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/fed/admin/trustedsppartners
```

File spCurlData.in:

```
idpTenantName=&idpTenantURL=&
spPartnerName=spPartner-sample&
spProviderID=&metadata=&metadataURL=&
assertionConsumerURL=&logoutRequestURL=&
logoutResponseURL=&signingCert=&encryptionCert=&
nameIDFormat=unspecified&ssoProfile=artifact&
generateNewKeys=&validityNewKeys=&preverify=false&
lastNameAttrName=&firstNameAttrName=&userNameAttrName=&
emailAttrName=&staticAttrName=&staticAttrValue=&customAttrs=...
```

Creating an IdP Partner cURL Command using /fedrest/createidp

The /fedrest/createidp request redirects the request url to the actual required url /oam/services/rest/11.1.2.0.0/fed/admin/trustedidppartners, which creates a Trusted IdP Partner Service.

The following is a sample file for this cURL command.

Where

- **ssoProfile**

  the SAML 2.0 SSO profile to use (artifact or httppost).

- **nameIDFormat**

  the NameID format used during Federation SSO. Possible values are emailaddress or unspecified. If emailaddress, then the NameID value of an Assertion created by the IdP will contain the user's email address; if unspecified, then the NameID value of an Assertion created by the IdP will contain the user's ID.

```
curl -v -i -u <USER>:<PASSWORD>
-X POST -d @idpCurlData.in http://<SERVER>:<PORT>/fedrest/createidp
```

File idpCurlData.in:

```
spTenantName=&spTenantURL=&
idpPartnerName=idpPartner-sample&
idpProviderID=&metadata=&metadataURL=&
ssoURL=&ssoSOAPURL=&logoutRequestURL=
&logoutResponseURL=&signingCert=&
encryptionCert=&succinctID=&
nameIDFormat=emailaddress&attributeLDAP=&
attributeSAML=&ssoProfile=artifact&faWelcomePage=
&tenantKeyName=&tenantKeyValue=&generateNewKeys=&
validityNewKeys=&preverify=false
```

```
curl -v -i -u <USER>:<PASSWORD>
-X POST -d @idpCurlData.in
https://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/fed/admin/trustedidppartners
```

File idpCurlData.in:

```
spTenantName=&spTenantURL=&
idpPartnerName=idpPartner-sample&
idpProviderID=&metadata=&metadataURL=&
ssoURL=&ssoSOAPURL=&logoutRequestURL=&
logoutResponseURL=&signingCert=&
encryptionCert=&succinctID=&
nameIDFormat=emailaddress&attributeLDAP=&
attributeSAML=&ssoProfile=artifact&faWelcomePage=&
tenantKeyName=&tenantKeyValue=&generateNewKeys=&
validityNewKeys=&preverify=false
```

Connecting Federation Servers to remote REST services using /fedrest/orchestrator

The /fedrest/orchestrator request redirects the request url to the actual required url /oam/services/rest/11.1.2.0.0/fed/admin/orchestrator, which connect two federation servers to remote REST services.

The following are sample files for the cURL commands.

Where

- **ssoProfile**

  the SAML 2.0 SSO profile to use (artifact or httppost).

- **nameIDFormat**

  the NameID format used during Federation SSO. Possible values are emailaddress or unspecified. If emailaddress, then the NameID value of an Assertion created by the IdP will contain the user's email address; if unspecified, then the NameID value of an Assertion created by the IdP will contain the user's ID.

```
curl -v -i -u <USER>:<PASSWORD>
-X POST -d @orch.in http://<SERVER>:<PORT>/fedrest/orchestrator
```

File orch.in:

```
command=setupSPAndIdPTrust&
spresturl=https://<SERVER>:<PORT>/fedrest/createidp&
spadminuser=<USER>&
spadminpassword=<PASSWORD>&
spmetadataurl=&
idpPartnerName=sample-idp&
sptype=oif&
idpresturl=http://<SERVER>:<PORT>/fedrest/createsp&
idpadminuser=<USER>&
idpadminpassword=<PASSWORD>&
idpmetadataurl=&
spPartnerName=sample-sp&
idptype=oif&
nameIDFormat=emailaddress&
ssoProfile=httppost
```

Note: idpmetadataurl and spmetadataurl should be url encoded.

```
curl -v -i -u <USER>:<PASSWORD>
-X POST -d @orch.in
https://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/fed/admin/orchestratorservice
```

File orch.in:

```
command=setupSPAndIdPTrust&
spresturl=http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/fed/admin/trustedidp
partners&
spadminuser=<USER>&
spadminpassword=<PASSWORD>&
```

```
spmetadataurl=&
idpPartnerName=sample-idp&
sptype=oif&
idpresturl=http://<SERVER>:<PORT>/oam/services/rest/11.1.2.0.0/fed/admin/trustedsp
partners&
idpadminuser=<USER>&
idpadminpassword=<PASSWORD>&
idpmetadataurl=&
spPartnerName=sample-sp&
idptype=oif&
nameIDFormat=emailaddress&
ssoProfile=httppost
```

19

Developing a Message Processing Plug-in

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

This chapter contains the following sections.

19.1 Understanding Custom SAML Elements

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

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

19.2 Extending the OIFMessageProcessingPlugin

Follow this procedure to extend the `OIFMessageProcessingPlugin` code.

1. Create a directory.

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

2. Create the following subdirectories.

   `plugindev/src/msgprocplugin`

3. Create `SampleMsgProcPlugin.java` using the content in Example 19–1.

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

Example 19–1 SampleMsgProcPlugin.java

```
package msgprocplugin;

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

public class SampleMsgProcPlugin extends OIFMessageProcessingPlugin {

        private boolean monitoringStatus;

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

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

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

                        msg += "=================ENDS==================\n";
                        err.println(msg);
                        return ExecutionStatus.SUCCESS;
```

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

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

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

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

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

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

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

4. Place `SampleMsgProcPlugin.java` in the `plugindev/src/msgprocplugin` directory.
5. Create the `SampleMsgProcPlugin.xml` plug-in manifest using the content in Example 19–2.

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

Example 19–2 SampleMsgProcPlugin.xml

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

6. Put `SampleMsgProcPlugin.xml` in the `plugindev/` directory
7. Create the MANIFEST.MF file using the content in Example 19–3.

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

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

Example 19–3 MANIFEST.MF

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

8. Put `MANIFEST.MF` in the `plugindev/` directory
9. Create the `compile.sh` shell script using the content in Example 19–4.

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

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

Example 19–4 compile.sh

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

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

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

19.3 Deploying the Message Processing Plug-in

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

1. Login to Oracle Access Management Console as administrator.
2. Click Application Security at the top of the Console.

3. Click Authentication Plug-ins under the Plug-ins section.

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

4. Click Import Plug-in.

   An Import Plug-in screen is displayed.

5. Click Browse and search for the `SampleMsgProcPlugin.jar` that was built in "Extending the OIFMessageProcessingPlugin" on page 19-1.

6. Click Import to upload the JAR.

7. Refresh the table and search for the plug-in you just imported.

8. Click Distribute Selected.

9. Click the Refresh icon to confirm that the status has changed to Distributed.

10. Click Activate Selected.

11. Click the Refresh icon to confirm that the status has changed to Activated.

The plug-in has now been installed and activated.

19.4 Enabling the Message Processing Plug-in

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

1. Open the `$DOMAIN_HOME/config/fmwconfig/oam-config.xml` file in a text editor.

2. Find the Setting with name `messageprocessingeplugin` tag defined under the Setting with name `fedserverconfig` tag.

3. Change the value of `messageprocessingeplugin` to the name of the plug-in.

4. Find the Setting with name `messageprocessingenabled` tag defined under the Setting with name `fedserverconfig` tag.

5. Change the value of `messageprocessingenabled` from false to true.

6. Find the Setting with name `Version` (near the top of the file) and increment the version number.

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

7. Save the file.

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

20

Implementing Custom Authentication Actions

Custom authentication actions enable site-specific operations to be executed during a Federation single sign-on flow with Oracle Access Management Identity Federation acting as an Identity Provider. These actions can be used to authenticate the user or check the validity of the user's session if the user is already authenticated.

The following sections explain how to implement custom authentication actions.

20.1 Understanding Custom Authentication Actions

The Oracle Access Management Identity Federation server (Identity Federation) implements custom actions using the pre- and post-processing action plug-ins. The pre- and post-processing plug-ins are implemented as JSP or JavaEE servlets which are invoked during a Federation single sign-on (SSO) flow either before or after invoking Oracle Access Manager. The following sections explain how the actions work and how they interact with Identity Federation.

20.1.1 Using Pre and Post Processing Custom Authentication Actions

Identity Federation acting as an Identity Provider (IdP) always invokes Oracle Access Manager during a Federation SSO operation. This is done either to identify the user or to check the user's session to see if the user is already authenticated. If Identity Federation determines the user must be identified, the user is forwarded to Oracle Access Manager, specifying the root context and the relative path of the OAM endpoint. At this point, Oracle Access Manager will:

- Perform an authentication operation if necessary.
- Check the validity of the user's session
- (Optionally) perform an Authorization verification to ensure that the user can perform a Federation SSO operation with the SP Partner.

If these operations are successful, Oracle Access Manager forwards the user back to Identity Federation with the authentication information (a user identifier and the time at which the identity was established). Identity Federation analyzes the information and creates or updates the user session. Custom actions can be used during this interaction to:

- Manipulate the data exchanged between Identity Federation and the Oracle Access Manager Authentication Engine. For example, you can construct an email address from a user name: `johndoe` becomes `johndoe@mycompany.com`.
- Perform additional steps during authentication. For example, you can contact an external data source or system to obtain more information about the user.

20.1.2 Setting Up a Custom Authentication Action Plug-in

The following overview illustrates how to set up a custom action plug-in.

1. Implement one or more custom action plug-ins as explained.
   - Implement a pre-processing action plug-in to be performed before invoking Oracle Access Manager. (See "Using Pre-Processing Custom Actions" on page 20-3.)
   - Implement a post-processing plug-in for any actions or changes to be performed after authentication. This would occur when the user is returned from Oracle Access Manager to Identity Federation. (See "Using Post-Processing Custom Actions" on page 20-6.)
2. Deploy the plug-in(s) to the WebLogic Managed Server on which Oracle Access Manager is running.
3. Configure Identity Federation based on the plug-in task you are implementing.
   - Configure Identity Federation to invoke the plug-in (rather than Oracle Access Manager) if the plug-in is to perform pre-processing tasks.
   - Configure Identity Federation to have Oracle Access Manager invoke the plug-in (rather than redirecting to Identity Federation) if the plug-in is to perform post-processing tasks.

20.1.3 Understanding the Custom Action Flow

When Identity Federation needs to authenticate a user, the flow is as follows.

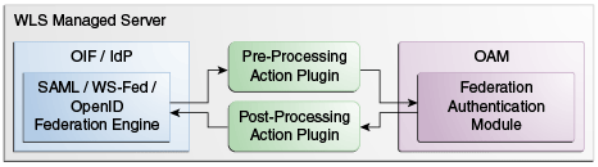
1. Identity Federation, as part of a runtime IdP SSO flow, determines whether the:
   - User needs to be locally authenticated by Oracle Access Manager.
   - User has an existing session and needs to be forwarded to Oracle Access Manager to check the validity of the session.
2. Identity Federation (acting as the IdP) invokes the pre-processing plug-in to perform the applicable custom tasks.
3. The pre-processing plug-in invokes Oracle Access Manager.
4. Oracle Access Manager challenges and authenticates the user, or checks the validity of the user's session.
5. Oracle Access Manager bundles the authentication data and invokes the post-processing plug-in to perform applicable custom tasks.
6. The post-processing plug-in invokes Identity Federation, providing the authentication data.

7. Identity Federation (acting as an IdP) resumes operations.

Figure 20–1 illustrates this flow when Identity Federation is customized and configured to invoke plug-ins:

- Before Identity Federation invokes Oracle Access Manager for authentication or session validation. (See "Using Pre-Processing Custom Actions" on page 20-3.)
- Before Oracle Access Manager invokes Identity Federation after authentication or session validation. (See "Using Post-Processing Custom Actions" on page 20-6.)

Figure 20–1 Custom Actions Plug-in Flow



20.2 Using Pre-Processing Custom Actions

The pre-processing plug-in is a module to which the user is directed, as part of an authentication operation, before invoking Oracle Access Manager. The plug-in enables custom actions to be taken before authentication. When the plug-in is in use, Identity Federation does not redirect the user to the Authentication Engine; rather, it forwards the user internally to the plug-in, passing to it certain data for use during authentication. After performing its custom actions, the plug-in forwards the user to Oracle Access Manager, along with the data originally provided by Identity Federation, to resume the authentication flow. The following sections contain details.

- Passing Data to the Pre-Processing Plug-in
- Configuring Identity Federation for the Pre-Processing Action

20.2.1 Passing Data to the Pre-Processing Plug-in

The pre-processing custom action interacts with Identity Federation. When Identity Federation redirects a user to Oracle Access Manager, it passes certain data as attributes in the `HttpServletRequest` object. The same data must be made available to pre-processing plug-ins. The data includes:

- The default scheme identifier to be used to challenge the user. This String is identified by `oracle.security.fed.authn.defaultschemeid`.
- A list of scheme identifiers requested by the service provider (SP) to be used to challenge the user. This String list is identified by `oracle.security.fed.authn.schemeidlevels`.
- The comparison rule requested by the SP to determine the scheme with which to challenge the user when a list of scheme identifiers is provided. This is a String identified by `oracle.security.fed.authn.schemeidcomp`.
- The Force Authentication flag indicating whether Oracle Access Manager should challenge the user - even if the user is already authenticated. This is a Boolean identified by `oracle.security.fed.authn.forceauthn`. If missing, false is assumed.
- The Is Passive flag indicating whether Oracle Access Manager is allowed to visually interact with the user. This is a Boolean identified by `oracle.security.fed.authn.passive`. If missing, false is assumed.

- An identifier referencing the action being performed. This String is identified by `oracle.security.fed.authn.refid`.
- The identifier (userID) of the user if set. This String is identified by `oracle.security.fed.authn.userid`.
- The canonical user identifier of the user (userID + Identity Store Name + LDAP DN) if set. This String is identified by `oracle.security.fed.authn.canonicaluserid`.
- The Identity Federation SessionID if set. This String is identified by `oracle.security.fed.sessionid`.
- The identifier referencing the Oracle Access Manager server used to authenticate the user. This String is identified by `oracle.security.fed.authn.engineid`.
- The partner name and the description of the remote SP for which this local authentication is requested, if a federated SSO operation is performed. This String is identified by `oracle.security.fed.authn.providerid` and `oracle.security.fed.authn.providerdescription` respectively.
- The web context where the user should be redirected after authentication by Oracle Access Manager. This String is identified by `oracle.security.fed.return.webcontext`. The root web context is /oam.
- The web relative path where the user should be redirected after authentication by Oracle Access Manager. This String is identified by `oracle.security.fed.return.webpath`. The relative path is /server/fed/authn.

Note: The pre-processing plug-in can modify all attributes that were set in the `HttpServletRequest` object except the following:

- oracle.security.fed.authn.defaultschemeid
- oracle.security.fed.authn.schemeidlevels
- oracle.security.fed.authn.schemeidcomp
- oracle.security.fed.authn.refid
- oracle.security.fed.authn.engineid
- oracle.security.fed.return.webcontext
- oracle.security.fed.return.webpath

20.2.2 Configuring Identity Federation for the Pre-Processing Action

Configure Identity Federation to forward the user to a pre-processing plug-in by performing these tasks.

1. Enter the WLST environment.

   ```
   $IAM_HOME/common/bin/wlst.sh
   ```

2. Connect to the WLS Admin Server.

   ```
   connect()
   ```

3. Navigate to the Domain Runtime folder.

   ```
   domainRuntime()
   ```

4. Execute the `putStringProperty()` WLST command to set the following properties:

- The preauthnenginewebcontext property references the web context where the custom JSP Page or servlet of the pre-processing plug-in resides. Replace CUSTOM_WEB_CONTEXT with the value specific to your plug-in.

  putStringProperty("/authnengines/preauthnenginewebcontext", "CUSTOM_WEB_CONTEXT ")

- The preauthnenginewebpath property references the path in the web context where the pre-processing plug-in resides. Replace CUSTOM_WEB_PATH with the value specific to your plug-in.

  putStringProperty("/authnengines/preauthnenginewebpath", "CUSTOM_WEB_PATH")

5. Execute the `putBooleanProperty()` WLST command to enable or disable the pre-processing custom plug-in.
   - putBooleanProperty("/authnengines/preauthnengineenabled", "true") to configure Identity Federation to invoke the pre-processing plug-in.
   - putBooleanProperty("/authnengines/preauthnengineenabled", "false") to configure Identity Federation not to invoke the pre-processing plug-in.

20.3 Example: Custom Action Pre-processing

This section illustrates a simple pre-processing plug-in that is invoked by Identity Federation before the user is redirected to Oracle Access Manager. This pre-processing plug-in retrieves the name of the SP partner with which the Federation SSO operation is performed and will save it in a custom cookie that can be used by custom pages; for example, a custom error page. In this example:

- Identity Federation acts as an IdP
- A custom pre-authentication plug-in is used in this example to set a cookie containing the SP partner name. The cookie is called fed-sppartner-cookie.

The pre-processing plug-in consists of a Web application with a root context set to /plugin. It contains one JSP page (named `cookiepartnerset.jsp`) which sets the SP partner name in a cookie. Example 20–1 is an example implementation of `cookiepartnerset.jsp`.

Example 20–1 cookiepartnerset.jsp

```
<%@page buffer="5" autoFlush="true" session="false"%>
<%@page language="java" import="java.util.*, javax.naming.*,
  javax.naming.directory.*, java.net.*"%>
<%
response.setHeader("Cache-Control", "no-cache");
response.setHeader("Pragma", "no-cache");
response.setHeader("Expires", "Thu, 29 Oct 1969 17:04:19 GMT");

String partnerName = (String)request.getAttribute
  ("oracle.security.fed.authn.providerid");

Cookie cookie = new Cookie("fed-sppartner-cookie", partnerName);
response.addCookie(cookie);

// forward to the OAM server to resume the flow
request.getSession().getServletContext().getContext("/oam").getRequestDispatcher
 ("/server/fed/authn").forward(request, response);
%>
```

Note: The WAR file of this Web application will need to be deployed to the WLS Managed Server on which Oracle Access Manager is running.

To resume the flow, the plug-in redirects the user to Oracle Access Manager by means of an internal forward. Take the following steps to configure Identity Federation to invoke the pre-processing plug-in.

1. Enter the WLST environment:

   `$IAM_HOME/common/bin/wlst.sh`

2. Connect to the WLS Admin Server:

   `connect()`

3. Navigate to the Domain Runtime folder:

   `domainRuntime()`

4. Execute the `putStringProperty()` WLST command to set the preauthnenginewebcontext property.

   putStringProperty("/authnengines/preauthnenginewebcontext", "/plugin")

5. Execute the `putStringProperty()` WLST command to set the preauthnenginewebpath property.

   putStringProperty("/authnengines/preauthnenginewebpath", "/cookiepartnerset.jsp")

6. Execute the `putBooleanProperty()` WLST command to enable the pre-processing plug-in to be invoked by Identity Federation.

   putBooleanProperty("/authnengines/preauthnengineenabled", "true")

20.4 Using Post-Processing Custom Actions

The user is directed to the post-processing plug-in module, as part of an authentication operation, after the Oracle Access Manager Authentication Engine has completed processing and before the user is directed to Identity Federation. The plug-in enables custom actions to be taken after authentication.

When the post-processing plug-in is in use, Oracle Access Manager forwards the user and authentication data internally to it. After performing its custom actions, the plug-in returns the user to Identity Federation, supplying the authentication data. The plug-in must provide Identity Federation with the data that was passed to it as part of the authentication flow; this consists of attributes that were set in the `HttpServletRequest` object. The following sections have details.

- Passing Data to the Post-Processing Plug-in
- Configuring Identity Federation for the Post-Processing Action

20.4.1 Passing Data to the Post-Processing Plug-in

The post-processing plug-in interacts with Identity Federation. When Oracle Access Manager redirects a user to Identity Federation, it passes certain data to the plug-in as attributes in the `HttpServletRequest` object. The data includes:

- The identifier referencing the requested action that was performed. The String is identified by `oracle.security.fed.authn.refid`.
- The schemeID and level of the authentication performed by Oracle Access Manager. The String is identified by `oracle.security.fed.authn.result.schemeidlevel`.
- The result of the authentication operation. The String is identified by `oracle.security.fed.authn.result.statuscode` as SUCCESS, if the operation was successful.
- The partner name of the remote SP for which this local authentication is requested if a federated SSO operation is performed. The String is identified by `oracle.security.fed.authn.providerid`.
- The Oracle Access Manager module identifier used to authenticate the user. The String is identified by `oracle.security.fed.authn.engineid`.
- The canonical identifier of the user (userID + Identity Store Name + LDAP DN). The String is identified by `oracle.security.fed.authn.userid`.
- The authentication time. The Date is identified by `oracle.security.fed.authn.authntime`.
- The expiration time of the authenticated session. The Date is identified by `oracle.security.fed.authn.expirationtime`.
- The user's Oracle Access Manager SessionID. The String is identified by `oracle.security.fed.authn.oamsessionid`.
- The user's Oracle Access Manager Session type. The String is identified by `oracle.security.fed.authn.oamsessiontype`.
- The Identity Federation SessionID if set. The String is identified by `oracle.security.fed.sessionid`.

Note: The plug-in can modify all attributes that were set on the `HttpServletRequest` object except:

- oracle.security.fed.authn.result.schemeidlevel
- oracle.security.fed.authn.engineid
- oracle.security.fed.authn.oamsessionid
- oracle.security.fed.authn.oamsessiontype
- oracle.security.fed.sessionid

The custom post-processing plug-in can add these optional elements.

- A map of attributes to be included in the outgoing SAML Assertion as SAML Attributes. This map has String objects as keys and a Collection (Set/List) of String objects or a String object as values (identified by `oracle.security.fed.authn.fedattributes`).

  The attributes will be included in the outgoing SSO response as is, and will only be sent for this current Federation SSO operation. They will be discarded afterwards.

- A string to be used as the SAML NameID value instead of the configured NameID expression in the SP Partner entry. For SAML 2.0 SP Partners, this string will only be used if the NameID format for the SP Partner is not Persistent or Transient. The String is identified by `oracle.security.fed.authn.feduserid`.

This string will only be used as the NameID for this current Federation SSO operation and will be discarded afterwards.

After processing, the post-processing plug-in must forward the user to Identity Federation. (Oracle Access Manager would invoke in the absence of the plug-in).

- The root web context is /oamfed
- The relative path is /user/loginsso

20.4.2 Configuring Identity Federation for the Post-Processing Action

Configure Identity Federation to forward the user to a post-processing plug-in by performing the following tasks.

1. Enter the WLST environment:

   ```
   $IAM_HOME/common/bin/wlst.sh
   ```

2. Connect to the WLS Admin Server:

   ```
   connect()
   ```

3. Navigate to the Domain Runtime folder:

   ```
   domainRuntime()
   ```

4. Execute the `putStringProperty()` WLST command to set the two following properties:

   - The `postauthnenginewebcontext` property references the web context where the custom JSP Page or servlet of the post-processing plug-in resides. Replace CUSTOM_WEB_CONTEXT by the value specific to your plug-in.

     putStringProperty("/authnengines/postauthnenginewebcontext", "CUSTOM_WEB_CONTEXT")

   - The `postauthnenginewebpath` property references the path in the web context where the post-processing plug-in resides. Replace CUSTOM_WEB_PATH by the value specific to your plug-in.

     putStringProperty("/authnengines/postauthnenginewebpath", "CUSTOM_WEB_PATH")

5. Execute the `putBooleanProperty()` WLST command to enable or disable the post-processing plug-in.

   - putBooleanProperty("/authnengines/postauthnengineenabled", "true") to configure Identity Federation to invoke the post-processing plug-in.
   - putBooleanProperty("/authnengines/postauthnengineenabled", "false") to configure Identity Federation not to invoke the post-processing plug-in.

20.5 Example: Custom Action Post-Processing

This section illustrates a simple post-processing plug-in that is invoked by Oracle Access Manager before the user is redirected to Identity Federation at the end of a local authentication operation. This plug-in accesses a custom cookie presented by the browser, extracts data from it, and sets the data as attributes. Identity Federation will then include it in the outgoing SAML assertion. In this example:

- Identity Federation acts as an IdP.

- The custom post authentication plug-in sets some attributes as session attributes called attr1 and attr2.
- Identity Federation (as the IdP) will send the attr1 and attr2 attributes to the SP Partner with which the Federation SSO operation is being performed when creating an assertion.
- A custom component sets the cookie used in this example.

In this sample, the plug-in adds the following attributes, extracted from a custom cookie that is previously set by another component, after a successful authentication:

- cookie-language contains the preferred language of the user.
- cookie-homepage contains the preferred home page of the user.

The post-processing plug-in consists of a Web application with a root context set to /plugin. It contains one JSP page (named cookieextract.jsp) which extracts the data from the custom cookie and sets it as session attributes. Example 20–2 is an example implementation of cookieextract.jsp.

Example 20–2 cookieextract.jsp

```
<%@page buffer="5" autoFlush="true" session="false"%>
<%@page language="java" import="java.util.*, javax.naming.*,
  javax.naming.directory.*, java.net.*"%>
<%
response.setHeader("Cache-Control", "no-cache");
response.setHeader("Pragma", "no-cache");
response.setHeader("Expires", "Thu, 29 Oct 1969 17:04:19 GMT");

// check if authentication was successful
if
("SUCCESS".equals(request.getAttribute("oracle.security.fed.authn.result.statuscod
e")))
{
        // authentication was successful. Attributes will be added
        Map attributes = new HashMap();

        // get the cookie
        Cookie[] cookies = request.getCookies();
        String cookieValue = null;
        for(int i = 0; i < cookies.length; i++)
        {
                Cookie cookie = cookies[i];
                if (cookie.getName().equals("customcookie"))
                        cookieValue = cookie.getValue();
        }
        if (cookieValue != null && cookieValue.length() > 0)
        {
                StringTokenizer st = new StringTokenizer(cookieValue, "+");
                String language = st.nextToken();
                String homepage = st.nextToken();

                attributes.put("cookie-language", language);
                attributes.put("cookie-homepage", homepage);

                request.setAttribute( "oracle.security.fed.authn.fedattributes",
attributes);
        }
}
```

```
// forward to the OIF server to resume the flow
request.getSession().getServletContext().getContext("/oamfed").getRequestDispatche
r("/user/loginsso").forward(request, response);
%>
```

Note: The WAR file of this Web application will need to be deployed on the WLS Managed Server where OAM is running

The plug-in redirects the user to the Identity Federation server by means of an internal forward to resume the flow. Take the following steps to configure Identity Federation to invoke the post-processing plug-in at the end of local authentication flow.

1. Enter the WLST environment:

   `$IAM_HOME/common/bin/wlst.sh`

2. Connect to the WLS Admin Server:

   `connect()`

3. Navigate to the Domain Runtime folder:

   `domainRuntime()`

4. Execute the `putStringProperty()` WLST command to set the postauthnenginewebcontext property.

   putStringProperty("/authnengines/postauthnenginewebcontext", "/plugin")

5. Execute the `putStringProperty()` WLST command to set the postauthnenginewebpath property.

   putStringProperty("/authnengines/postauthnenginewebpath", "/cookieextract.jsp")

6. Execute the `putBooleanProperty()` WLST command to enable the post-processing plug-in.

   putBooleanProperty("/authnengines/postauthnengineenabled", "true")

Part VI

Developing with Security Token Service

This part discusses developing applications using the Oracle Access Management Security Token Service APIs.

It contains the following chapters:

21

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:

- Introduction to Oracle Security Token Service Custom Token Module Classes
- Writing a TokenValidatorModule Class
- Writing a TokenIssuanceModule Class

21.1 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*.

21.2 Writing a TokenValidatorModule Class

This section provides the following topics:

- About Writing a TokenValidatorModule Class
- Writing a TokenValidatorModule Class

21.2.1 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 21–1 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")
      ```

21.2.2 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 21.2.1, "About Writing a TokenValidatorModule Class"

- *Oracle Fusion Middleware Java API Reference for Oracle Access Management Security Token Service*

2. Proceed as needed:
   - Section 21.3, "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*

21.3 Writing a TokenIssuanceModule Class

This section provides the following topics:

- About Writing a TokenIssuanceModule Class
- Writing a TokenIssuanceModule Class

21.3.1 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 21–2 provides and example of `EmailTokenIssuanceModuleImpl` class. The overview that follows outlines development highlights for this module class.

Example 21–2 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.

21.3.2 Writing a TokenIssuanceModule Class

Task overview: Writing an Issuance Module class

1. Write the issuance module class as you refer to Section 21.3.1, "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 VII

Appendices

This part 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:

21.4 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 21–1 on page A-2.
3. The user submits the login page, passing the parameters contained in Table 21–2 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.

21.4.1 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.

21.4.1.1 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.

21.4.1.2 Password Is About to Expire

A warning page is displayed where the user can either change their password, or continue without changing before continuing.

21.4.1.3 Grace Login Is in Force

Same behavior as when password is about to expire.

21.4.1.4 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.

21.5 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

21.5.1 Login Page Parameters

The URL for the login page must accept the parameters listed in Table 21–1 on page A-2.

Table 21–1 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 21–1 (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 21–2 to the authentication URL:

`http://sso_host:sso_port/sso/auth`

Table 21–2 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**.

21.5.2 Change Password Page Parameters

The URL for the change password page must accept the parameters listed in Table 21–3.

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 21–3 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 21–3 (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 21–5 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 21–4 to the change password URL:

http://*sso_host*:*sso_port*/sso/ChangePwdServlet

Table 21–4 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 21–5 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.

21.6 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.

21.6.1 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 21–5.

Table 21–5 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 21–5 (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."

21.7 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.

21.7.1 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.

21.7.1.1 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.

21.7.1.2 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 21–2) in the login form. A number of sites contain a full list of ISO-639 two-letter language codes. Here is a site that contains a full list of ISO-3166 two-letter country codes:

```
http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html
```

> **Note:** In the event that the `locale` parameter is passed to the single sign-on server (Table 21–1), 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.

21.7.2 Rendering the Page

Once it determines the end-user's locale, the deployment-specific page must use the corresponding translation strings to render the page. To learn how to store and retrieve these strings, see the chapter about locale awareness in *Oracle Application Server Globalization Guide*. You may also want to consult standard documents about Java development. Here are two links:

- Java Internationalization Guide:

  ```
  http://java.sun.com/j2se/1.4.2/docs/guide/intl/index.html
  ```

- General link for Java documentation:

  ```
  http://java.sun.com/j2se/1.4.2/docs
  ```

21.8 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.

21.9 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`

21.9.1 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.

21.10 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 21–6 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 21–7 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 21–8 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.