Oracle® Fusion Middleware

Using the Oracle Access Management OAuth Service

11*g* Release 2 (11.1.2.3) for All Platforms

E54424-04

September 2015

ORACLE®

Oracle Fusion Middleware Administrator's Guide for Oracle Access Management, 11*g* Release 2 (11.1.2.3) for All Platforms

E54424-04

Primary Author: Michael Teger

Contributing Author: Vinaye Misra, Kevin Kessler, Cathy Tenga, Serge Pomorski

Contributor: Vadim Lander, Vamsi Motokuru, Damien Carru, Peter Povinec, Weifang Xie, Satish Madawand, Neelima Jadhav, Charles Wesley, Harshal X Shaw, Jeremy Banford, Rey Ong, Ramana Turlapati, Deepak Ramakrishnan, David Goldsmith, Vishal Parashar, Carlos Subi, Patricia Fuzesy

Preface

This customer extract from the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management* and the *Oracle Fusion Middleware Developer's Guide for Oracle Access Management* provides administration and developer information for using the Oracle Access Management OAuth Service. The OAuth Service allows organizations to implement the open standard OAuth 2.0 Web authorization protocol in an Access Manager environment. This Preface covers the following topics.

Audience

This document is intended for Administrators who are familiar with:

- Oracle WebLogic Server concepts and administration
- LDAP server concepts and administration
- Database concepts and administration (for policy and session management data)
- Web server concepts and administration
- WebGate and mod_osso agents
- Auditing, logging, and monitoring concepts
- Security token concepts
- Integration of the policy store, identity store, and familiarity with Oracle Identity Management and OIS might be required

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

This Preface is for the Using the Oracle Access Management extract from the *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management* and the *Oracle Fusion Middleware Developer's Guide for Oracle Access Management*. It provides details for using the Oracle Access Management implementation of the open standard OAuth 2.0 Web authorization protocol. For more information, see the following guides in the Oracle Fusion Middleware 11*g* Release 2 (11.1.2.3) documentation.

- Oracle Access Management *11g* Release 2 (11.1.2.3) *Release Notes*
- *Oracle Fusion Middleware Installation Guide for Oracle Identity and Access Management*—Explains how to use the Oracle Universal Installer and the WebLogic Configuration Wizard for initial Access Manager 11g deployment. Installing 11g WebGates for Access Manager is also covered.
- *Oracle Fusion Middleware Developer's Guide for Oracle Access Management*—Explains how to write custom applications and plug-ins to functions programmatically, to create custom Access Clients that protect non-Web-based resources.
- *Oracle Fusion Middleware Upgrade Guide for Java EE*—For information about the types of Java EE environments available in 10g and instructions for upgrading those environments to Oracle Fusion Middleware 11g.
- *Oracle Fusion Middleware Upgrade Guide for Oracle Identity and Access Management*
- *Oracle Fusion Middleware Migration Guide for Oracle Identity and Access Management*
- *Oracle Fusion Middleware Performance and Tuning Guide*
- *Oracle Fusion Middleware Administrator's Guide*—Describes how to manage a secure Oracle Fusion Middleware environment, including how to change ports, deploy applications, and how to back up and recover Oracle Fusion Middleware. This guide also explains how to move data from a test to a production environment.
- *Oracle Fusion Middleware Enterprise Deployment Guide for Oracle Identity Management*—For a step-by-step guide to deployment.
- *Oracle Fusion Middleware High Availability Guide*—For high availability conceptual information as well as administration and configuration procedures for Administrators, developers, and others whose role is to deploy and manage Oracle Fusion Middleware with high availability requirements.
- *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference for Identity and Access Management*—Provides details on customized Identity and Access Management WLST commands.
- *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*—Describes how to administer and secure Web services.

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.

Convention	Meaning
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.

1 Understanding OAuth Services

OAuth provides a method to exchange identity credentials for an access token. This token, in return, can be used for granting access of private resources in a user's account on one service provider site to a second, consumer site without having to divulge the identity credentials to the consumer site. Oracle Access Management implements the OAuth Core 2.0 specifications to offer OAuth Services.

This chapter describes the purpose and capabilities of the Oracle Access Management OAuth Services. It includes the following topics.

1.1 Using Oracle Access Management OAuth Services

OAuth is an open standard authorization protocol that provides authentication and access control between a Client (including mobile apps and Web services) and a Resource Owner (or Service Provider) on the Web. Oracle Access Management OAuth Services is based on this standard and designed:

- To address enterprise-level extranet use cases.
- To provide secure mobile access to APIs.
- To leverage built-in Oracle Access Management features (including authentication schemes, strong authentication, fraud detection, session management and federated authentication).
- To secure confidential clients with a high level of security.

Oracle Access Management OAuth Services are available for Web clients or for mobile clients. OAuth Services for Web clients implement the standard OAuth 2.0 use cases. In this case, the clients rely on a Client ID/Client Password (or secret) to secure itself. For an example, see `http://tools.ietf.org/html/rfc6749#page-4`.

Mobile OAuth Services is an extension on top of the standard OAuth specification in which the identity of the mobile client is secured through application registration, and a credential specific to the mobile client is included with a request for access. As mobile clients store passwords on mobile devices, they can not be confidential like Web clients so the identity of the mobile client is established through device/app registration before accessing REST or Web services using the OAuth Services Access Token. Thus, the key difference between the standard Web and mobile OAuth Services use cases is that the mobile client is secure before it can request an Access Token (through device/app registration) whereas a standard OAuth Web client uses a credential like password or an assertion to self identify. Section 2.1, "Enabling OAuth Services" contains details on how OAuth Services and Mobile OAuth Services are enabled and configured separately. See the following sections for details on how OAuth Services works.

- Understanding OAuth Services Authorization for Web Clients
- Understanding OAuth Services Authorization for Mobile Clients

1.2 Understanding OAuth Services Authorization for Web Clients

In the most common OAuth scenario, the Client accessing the protected resource is issued a different set of credentials than those of the user. (In this case, the user does not disclose their credentials to the client.) Oracle Access Management OAuth Services acts as the intermediary Authorization Server, interacting directly with the Client, the service hosting the user's protected resource (Resource Owner) and the server on which the resource is located (Resource Server). It issues access tokens to a Client that has (already) successfully authenticated with the Resource Server - in effect, authorizing the client to access private resources or activities on the server. A single Authorization Server instance can issue access tokens accepted by multiple resource servers.

> **Note:** OAuth does not impose special requirements on the interaction between a Resource Server and an Authorization Server.

The following sections describe web-based scenarios in which OAuth Services works.

- Understanding 3-Legged Authorization
- Understanding 2-Legged Authorization

The scenarios introduce the concept of OAuth Services endpoints. For detailed information on these endpoints, see Understanding the Authorization and Authentication Endpoints. The scenarios also use terms documented in Understanding the OAuth Services Components including the following:

- The Resource Owner refers to the user requesting access to a protected resource.
- The Client is the mobile app or Web service through which the Resource Owner is requesting access to a protected resource.
- OAuth Services refers to the Authorization Server, Oracle Access Management.
- The Resource Server is the machine on which the protected resource is stored. It can be any website or Web service where restricted resources are located; for example, a photo sharing site, a blogging platform and an online bank service control access to private resources and activities. The Resource Server is deployed in a different location from Oracle Access Management and the Client. The Resource Server needs to be capable of accepting and responding to protected

resource requests using access tokens. The Resource Server must also validate the access token with OAuth Services as described in Enforcing Access Control.

1.2.1 Understanding 3-Legged Authorization

In 3-legged authorization, the Resource Owner grants access to an OAuth-enabled Client to request access to resources stored on an OAuth protected Resource Server. Oracle Access Management OAuth Services validates the Resource Owner's identity and presents the owner with a consent form in a Web browser when approval is required. The third leg in this authorization scheme is the step in which the user grants or denies the client access. The following text has more details and Figure 1–1 illustrates the process.

> **Note:** A WebGate proxy is required to use 3-legged authorization with an external LDAP directory server. See Section 2.5, "Configuring a WebGate to Protect OAuth Services" for details.

1. The Resource Owner (user) undertakes an action in the user-agent (a browser, for example) that requires the Client web service (or app) to access protected resources belonging to the user on a different site.
2. The Client initiates the OAuth flow by invoking the OAuth Services authorization endpoint to get a request token. The Client sends its identifier, the requested scope, and a redirection URI to which the Authorization Server will direct the user-agent once access is granted or denied.
3. OAuth Services redirects the user-agent to request the Resource Owner's password credentials.
4. Access Manager displays a login page requesting a user name and password from the Resource Owner. OAuth Services supports all authentication schemes provided by Access Manager.
5. The Resource Owner enters a user name and password.
6. Access Manager validates the credentials, returns a request token and redirects the user-agent to OAuth Services.
7. OAuth Services determines that the Resource Server requires the user's consent before the authorization code can be sent to the Client.
8. OAuth Services displays the user consent form.

   Web-based clients require the consent form to be protected by a WebGate.For details, see Section 2.5, "Configuring a WebGate to Protect OAuth Services."
9. The user approves the request.
10. OAuth Services returns an authorization code to the Client using the redirection URI.

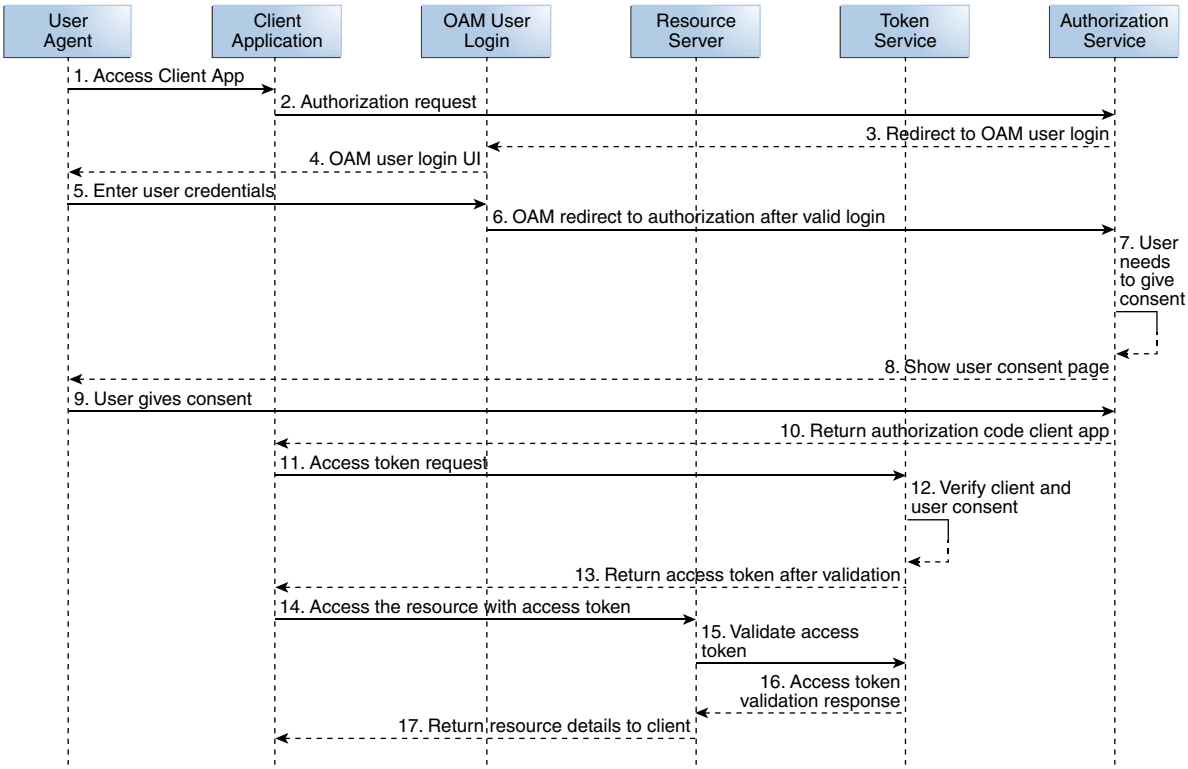
> **Note:** The Authorization Code grant type is required for 3-legged authorization. See Section 1.4.3, "Understanding Clients" for details.

11. The Client sends the authorization code in a POST request (including the redirection URI used to obtain the authorization code for verification) to the token endpoint and requests an OAuth access token. When making the request, the Client authenticates with OAuth Services.

12. If the client type requires client credentials, the OAuth Services authenticates the client credentials, validates the authorization code, and ensures that the redirection URI received matches the URI previously used to return the authorization code. OAuth Services also validates the requested scope based on the Resource Server's configuration and the user's consent details.
13. OAuth Services returns an access token to the Client.

    A *refresh token* may also be returned with the access token if the client sends a refresh token request. For more information, see Section 1.5, "Understanding OAuth Services Tokens."
14. The Client presents the access token to the Resource Server.
15. The Resource Server validates the access token by sending a request to the OAuth Services token endpoint and waits for a success or failure response.
16. OAuth Services validates and sends the token success or failure response back to the Resource Server.
17. If the token is deemed valid, the Resource Server returns the requested resource to the Client.

Figure 1–1 OAuth 3-Legged Flow Diagram



OAuth 3-legged flow diagram

****

1.2.2 Understanding 2-Legged Authorization

In 2-legged authorization, the OAuth Client is pre-approved to access resources; thus, the user consent form step (described in Understanding 3-Legged Authorization) is not required. In this scenario, Access Manager returns a request token to the Client which the client sends to OAuth Services to request an access token. Because the request token is pre-authorized, OAuth Services token service returns an access token to the Client without displaying the consent form. This arrangement fits a service-to-service model, especially when the requesting service (Client) and the Resource Server are in a close partnership and Resource Owner approval is either assumed or not required.

Note: The Client Credentials grant type or the Resource Owner Credentials grant type are required for 2-legged authorization. See Section 2.3.3, "Configuring Clients" for more information.

1.3 Understanding OAuth Services Authorization for Mobile Clients

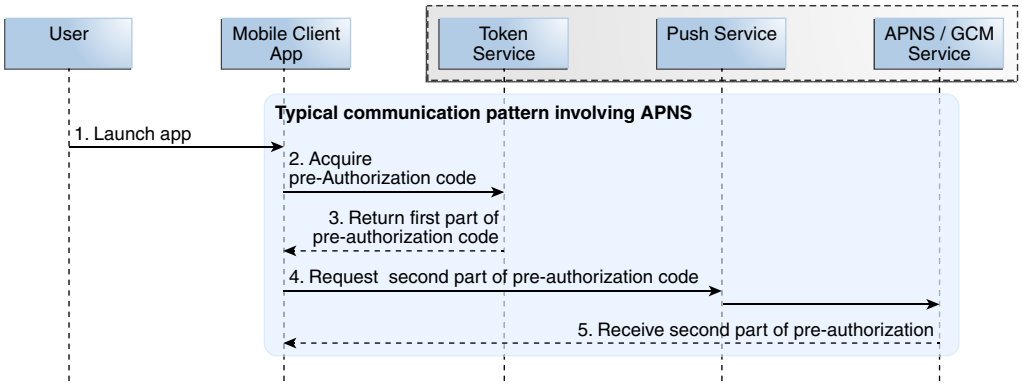
The Mobile OAuth Services authorization scenario supports mobile apps that run in a browser as well as device-native apps that do not use a browser or just use the browser during user authentication. This scenario provides enhanced security support (in addition to the baseline security measures defined in the OAuth 2.0 specification) including:

- Client apps must be registered with OAuth Services in the Identity Domain that your organization uses to manage Mobile OAuth Services clients.
- Mobile applications must register with Oracle Access Management prior to using OAuth Services and each registration is specific to one app on the device. After the application registration, the mobile app will have a client token. It uses this token as the security credential for making Access Token requests. In comparison, the OAuth web client uses either a credential-like password or an assertion to identify itself.
- This scenario supports user consent management. If consent management is enabled, the client app prompts the user to accept or decline the app's request to register with Access Manager.

Note: This consent is controlled with the "Require User Consent for Client Registration" attribute under the Service Profiles. If set, the user will be asked to confirm app registration; if off, the user will not be asked. See Section 2.3.2, "Configuring Service Profiles."

- Except for access tokens (and user tokens if server-side SSO is disabled), the server does not send security material, such as OAAM device and session handles to the client on the mobile device, but stores it in the Server-Side Device Store. Access tokens are both sent to the client and stored in the Server-Side Device Store to provide for validation and life cycle management.
- The OAM server component can restrict token delivery to a specific app installed on a specific device by sending part of a token through HTTPS, and sending the other part through push notification using either the Apple Push Notification Service (APNS) or Google Cloud Messaging (GCM). Figure 1–2 illustrates this.

Figure 1–2 Using a Split Request to get a Client Verification Code



Flow for using a split request to get Client Verification code

****

The following scenario describes the additional interactions that Oracle Access Management undertakes when authenticating with a mobile client. The process is illustrated in Figure 1–3.

1. The Resource Owner opens the Client mobile app.

   An Oracle Access Management administrator has already registered this Client app as a Mobile OAuth Services Client.

2. The Mobile Client sends the client ID and the device token to OAuth Services and requests a client verification code.

3. OAuth Services returns half of the client verification code over HTTPS or HTTP.

   See Figure 1–2. This behavior can be configured in the Mobile Service Settings section of the OAuth Services Profile configuration page.

   - If the security level is set to **Advanced**, all codes and tokens are returned using both HTTP and push notification.
   - If the security level is set to **Standard** mode, all codes and tokens are sent over HTTP only.

   The rest of this scenario (beginning with step 4) contains details for when the security level is set to Advanced.

4. The Mobile client requests the second half of the client verification code from the OAuth Services push endpoint.

   The push endpoint forwards the request to the APNS or the GCM service depending on the mobile device's operating system.

5. The APNS or GCM service sends the second half of the client verification code to the Client app.

6. The Mobile client requests an authorization code from OAuth Services by sending the client verification code and the device token.

7. OAuth Services redirects the request to Access Manager.

8. Access Manager sends a login page to the user-agent so that the user can log in.

9. The Resource Owner (user) enters a user ID and password.

10. Access Manager validates the login and redirects to OAuth Services.
11. OAuth Services is configured to obtain the user's approval to register the device. (It will not ask for the user's consent to register if **Require User Consent for Client Registration** is disabled on the OAuth Services Profile Configuration page.)
12. The consent page is sent to the Resource Owner.
13. The Resource Owner provides (or denies) consent.
14. OAuth Services checks the Oracle Adaptive Access Manager (OAAM) plug-in to determine if additional authentication steps are required.
15. The plug-in determines that an additional challenge question is required.
16. The OAAM challenge question is sent to the Resource Owner.
17. The Resource Owner provides the challenge answer which is forwarded to the OAAM plug-in.
18. The OAAM plug-in validates the challenge answer.
19. OAuth Services uses the mobile redirect URI to return half of the authorization code that the mobile app will need to request a client token.
20. The Mobile OAuth Services client requests the second half of the authorization code from OAuth Services push endpoint.

    The push endpoint forwards the request to the APNS or GCM service.
21. The mobile client app receives the second half of the authorization code from the APNS or GCM service.

    The mobile client app assembles the authorization code in preparation for requesting a client token.
22. After validating the authorization code, the Mobile OAuth client uses the code to request the first half of the client token from the OAuth Services token endpoint.
23. The token endpoint returns the first half of the client token to the mobile client.
24. The mobile client requests the second half of the client token from the OAuth Services push endpoint.
25. The APNS or GCM service sends the second half of the client token to the mobile client app.

    The mobile client assembles the client token as well as a refresh token. The client can use the refresh token to request a new client token.
26. The mobile client prepares to request an access token by completing the following steps:
    - The Client requests and receives a client verification code from OAuth Services.
    - The Client requests and receives the first part of the authorization code from OAuth Services.
    - The resource owner does not need to log in if the user session is still valid.
    - User consent may be required based on the Resource Server scope to which the Client is requesting access.
    - The OAAM plug-in does not repeat its challenge.
    - The client requests the second part of the authorization code.

27. The APNS or GCM service returns the second half of the authorization code for the access token.

    The Client assembles the authorization code in preparation for the access token request.

28. The mobile client requests an access token by sending the client token and the access token authorization code.

29. The token endpoint sends the access token to the client. This behavior depends on whether the Security Level setting in the Mobile Service Settings section of the OAuth Services Profile configuration page is set to Advanced or Standard.

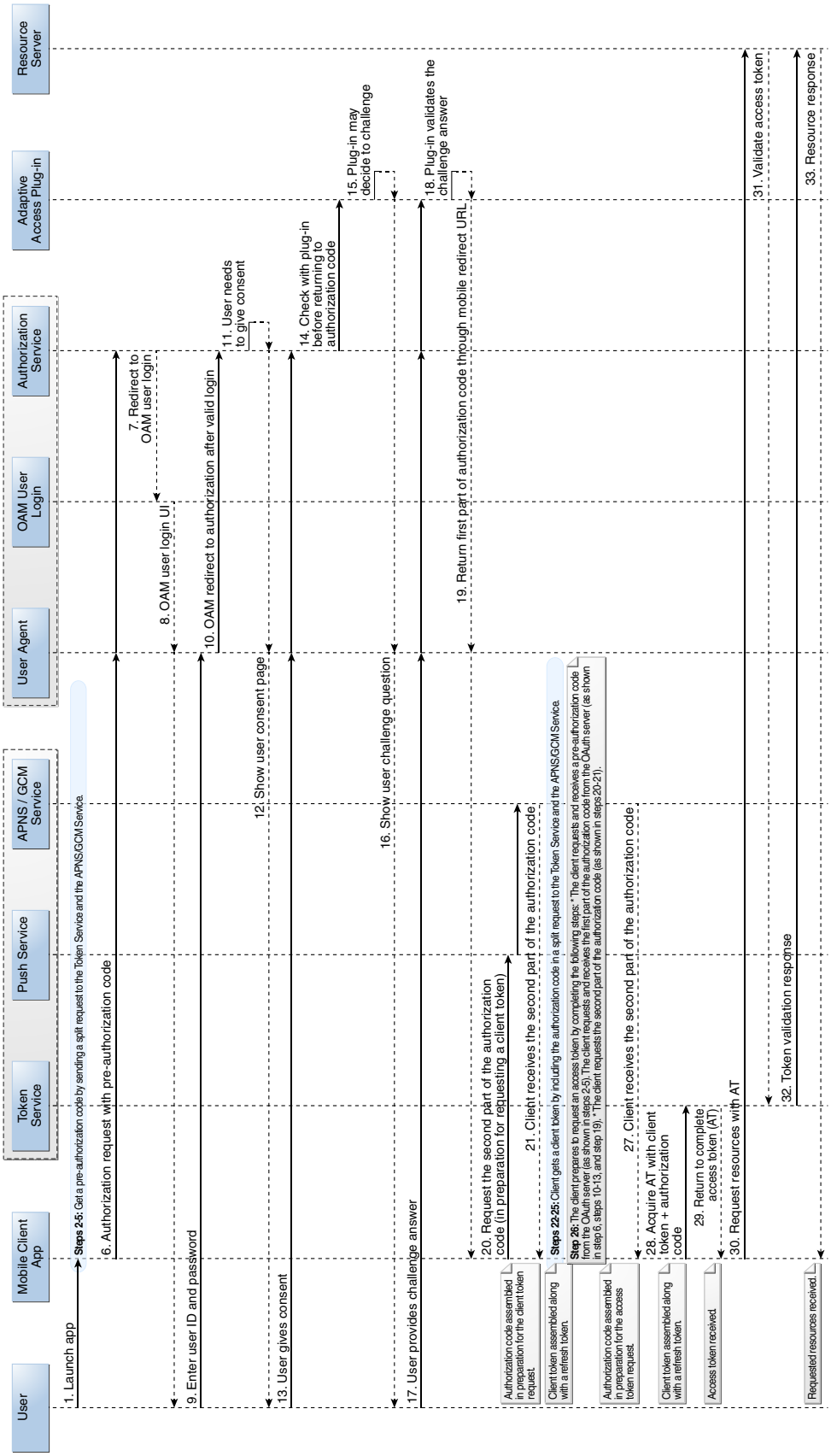
30. The Mobile OAuth Services client requests access to the protected resources by sending the access token to the Resource Server.

31. The Resource Server validates the access token with the OAuth Services token service. The Resource Server can also validate the token locally. If the certificates are configured correctly, JWT token signing is verified at the Resource Server.

32. The OAuth Services token service sends a response to the Resource Server.

33. The Resource Server sends the requested resources to the mobile client.

Figure 1–3 The Complete Mobile App Authorization Request Flow



Flow diagram for Mobile App Authorization Request

****

1.4 Understanding the OAuth Services Components

The following sections contain information about the Identity Domains configuration options. Information in the following sections applies to both Identity Federation and Mobile Security OAuth Services except for the Jailbreak Detection Policy which is specific to Mobile Security OAuth Services. See Chapter 2, "Configuring OAuth Services" for details on configuring these components.

- Understanding Identity Domains
- Understanding Service Profiles
- Understanding Clients
- Understanding Service Providers
- Understanding Resource Servers
- Understanding Plug-Ins
- Understanding Server Settings
- Understanding Jailbreak Detection Policy
- Understanding Token Life Cycle Management

1.4.1 Understanding Identity Domains

Identity Domains are entities that contain all artifacts required to provide standard OAuth Services or Mobile OAuth Services. Each Identity Domain is an independent entity. One of the primary use cases of the Identity Domain is for multi tenants deployments. Each Identity Domain will correspond to a tenant. This can apply to different departments in an organization if there is a need for independence. This will also be useful for cloud deployments where each Identity Domain can correspond to a separate tenant or entity. The following artifacts are just some of the components configured within an OAuth Services Identity Domain.

- One or more Service Profiles
- One or more Clients
- A Service Provider
- One or more Resource Servers
- Plug-ins
- Server Settings
- Token Life Cycle Management (search for and revoke tokens across an Identity Domain)

For information on configuring Identity Domains, see Section 2.3.1, "Configuring Identity Domains."

1.4.2 Understanding Service Profiles

A Service Profile defines the following settings.

- The clients with whom OAuth Services can interact

- The Custom and System Resource Servers that OAuth Services protects and to which it provides access
- Refresh token settings, token expiration settings, and the option to enable the token life-cycle management
- The User Profile Service and Consent Management Service profiles
- The enabled security profile plug-ins
- The mobile service settings, including security settings for the supported mobile platform(s)
- The root URL for the OAuth Services endpoints

If necessary, you can create multiple Service Profiles. Different Service Profiles may be needed if different clients or resources need to be grouped, or different token settings are required, or there are different service endpoints with different configuration settings. Being able to create multiple Service Profiles gives flexibility to configuration options although in most cases it may not be needed. For information on configuring Service Profiles, see Section 2.3.2, "Configuring Service Profiles."

1.4.3 Understanding Clients

The Client initiates the OAuth protocol by invoking the OAuth Services. Client profiles must be created using the OAuth Services interface (in the Oracle Access Management Console) before the protocol can be initiated. At a minimum, client profiles include the application name, a client ID, and one or more URIs to which OAuth Services will redirect the user-agent once access is granted or denied. An OAuth Services Client can be defined as Web, Public or Mobile.

- Web clients are assigned with a client ID and secret. These clients can interact with the OAuth Services server by sending the client ID and secret as part of an authorization header. It is up to each individual client to determine how the secret issued to them is securely stored.
- Public clients are assigned with a client ID but no secret. Typically these profiles pertain to browser based applications like Javascript or can be mobile based apps.
- Mobile clients are assigned with a client ID and the secret is dynamically generated as part of a mobile client's registration flow with OAuth Services. (The registration flow is proprietary and was developed by extending the OAuth specification.)

The client ID and secret are explained in the following bullet points.

- The Client ID is a unique string that represents the registration information and is required for each client. You can create a unique client ID or have OAuth Services generate one. OAuth Services compares the defined Client ID with the value the client sends over HTTPS or HTTP as part of an authorization request. If the values do not match, the request is rejected. Client IDs are Base64 encoded when they are sent as authorization header.
- The Client secret is the client password. You can create a unique client secret or have OAuth Services generate one. Web clients are required to have a Client ID and a Client secret. Mobile clients and Public clients, on the other hand, do not have a client secret and are given only a Client ID.

To request an access token, the client obtains authorization from the resource owner. The authorization is expressed in the form of an authorization grant, which the client uses to request the access token.The OAuth 2.0 specification provides authorization grant types for different security use cases. OAuth Services has implemented some of

these grant types. Web, Public and Mobile Clients can access the various OAuth Services grant types that are appropriate to them. For example, the Client Verification Code grant type is only relevant to mobile clients. The following grant types are supported by OAuth Services.

Note: (For general information about the OAuth specification grant types, see http://tools.ietf.org/html/rfc6749#section-1.3.

- Authorization Code - The Resource Owner logs in using Oracle Access Management. The token endpoint exchanges the authorization code along with client credentials for an access token. The Authorization Code grant type is required for 3-legged flows.
- Resource Owner Credentials - The Resource Owner provides the client with a user name and password. This is only suitable for highly trusted client applications because the client could abuse the password, or the password could unintentionally be disclosed to an attacker. Per the OAuth 2.0 specification, the authorization server and client should minimize use of this grant type and utilize other grant types whenever possible. The Resource Owner Credentials grant type is required for 2-legged authorization scenarios.
- Client Credentials – The client requests an access token using only its client credentials (or another supported means of authentication). This is suitable if the client is requesting access to protected resources under its control, or those of another resource owner when previously arranged with the authorization server. The Client Credentials grant type is required for 2-legged authorization scenarios.
- Refresh Token - Select this option to return a refresh token together with an access token in the token response. See Section 1.5, "Understanding OAuth Services Tokens" for more information.
- JWT Bearer - Allows a JWT assertion to be used to request an OAuth Services access token.
- SAML 2 Bearer - Allows a SAML2 assertion to be used to request an OAuth Services access token.
- OAM Credentials - Used to request OAM tokens, such as a master token, an access token, or an OAuth Services access token.
- Client Verification Code - Used by Mobile OAuth Services clients to request a pre-verification code which subsequently gets used in mobile client flows.

Privileges and Allowed Scopes can also be configured on a client by client basis. OAuth Services allows for the configuration of scopes to bypass the need for user consent. Thus, you can configure Privileges to define which clients are allowed which grant types. If applicable, the Client will then obtain an authorization grant that can be exchanged with OAuth Services for an access token. For information on configuring Clients, see Section 2.3.3, "Configuring Clients."

1.4.4 Understanding Service Providers

The Service Provider settings are used to manage the connection between OAuth Services and Access Manager, the back-end authorization Service Provider that supports OAuth Services. The OAuthServiceProvider is the default Service Provider for the DefaultDomain Identity Domain although a Custom Service Provider can be created. In this release, OAuth Services provides support for both OAM authentication (most authentication modules can be invoked, usually user/credential based plugins)

and IDS authentication. Any features not provided through OAM or IDS authentication will require a custom service provider. Each Identity Domain can only have one Service Provider.

> **Note:** Currently, authentication modules that have multi step orchestration in OAM can not be invoked; this refers to 2-legged OAuth Services scenarios. For 3-legged scenarios, there is no limitation on invoking OAM authentication modules because authentication is done through a browser flow.

For information on configuring Service Providers, see Section 2.3.4, "Configuring the Service Provider."

1.4.5 Understanding Resource Servers

Resource Servers settings are independently configured in a profile for each remote resource server that contains applications or services to be protected by OAuth Services. The Resource Server profile does not define the resource server specific settings like the endpoint or security protection - only OAuth Services related configurations.

Part of the Resource Server configuration involves defining *scope*. Scope determines the range of access the Client will have to the protected resource. Based on the scope setting, Oracle Access Management restricts access and informs the Client of the scope in the access token issued. Thus, an OAuth Services access token with proper scope needs to be obtained in order for a Client application to access a Resource Server.

The client provides a scope string in it's request to OAuth Services. The scope can be a URL or a string literal. After successful authentication and authorization, OAuth Services includes the scope in the Access Token. For example, if an OAuth client requests access to a resource for a specific end user, OAuth Services would create an Access Token with the scope defined as `UserProfile.me` and the client could access the User Profile Resource Server with the `/me` endpoint. (See Table 1–1.) In turn, the User Profile Resource Server will decide whether the client can access the resource with acquired access token or not. The following is true in regards to scope.

- A Resource Server can have one or more scope(s) associated with it.
- Multiple Resource Servers can be created if there is a need, for example, to have different token settings or to modify the default security protection (like not allowing revocation of consent). Each will have it's own set of scope definitions. This is not a common scenario
- The client sends the scope parameter as part of an authorization request. If any part of the scope parameter value is invalid, OAuth Services sends the client application an `invalid_scope` error response. If the scope parameter value is valid, it gets embedded as part of the authorization code and access tokens.

OAuth Services provides two out-of-the-box services modeled as Resource Servers and protected with an Access Token. For information on the User Profile Services and Consent Management Services Resource Servers, see the following sections.

- Understanding User Profile Services
- Understanding Consent Management Services

For configuration information, see Section 2.3.5, "Configuring Custom Resource Servers," Section 2.3.6, "Configuring User Profile Services," and Section 2.3.7, "Configuring Consent Management Services."

1.4.5.1 Understanding User Profile Services

The default **UserProfile** User Profile Services configuration is a Resource Server created during Oracle Access Management OAuth Services installation. This configuration allows your organization to use OAuth 2.0 to interact with a back-end LDAP directory server and perform the REST operations documented in Table 1–1 on Person, Group, and Relationship entities. See the *Oracle Fusion Middleware Developer's Guide for Oracle Access Management* for more details on using REST to interact with the User Profile Services.

Table 1–1 Default User Profile Services Endpoint Operations

Resource Endpoint (URI)	HTTP(S) Methods	Use Details
`http://host:port/ms_oauth/resources/userprofile/me`	GET allows read PUT allows update	The OAuth client can request *read* and *update* privileges for the specified user's profile.
`http://host:port/ms_oauth/resources/userprofile/users`	GET allows read, search POST allows create PUT allows update DELETE allows delete	The OAuth client can request *create*, *read*, *search*, *update*, and *delete* privileges for any user profile.
`http://host:port/ms_oauth/resources/userprofile/groups`	GET allows read, search POST allows create PUT allows update DELETE allows delete	The OAuth client can request *create*, *read*, *search*, *update*, and *delete* privileges for any group profile.
`http://host:port/ms_oauth/resources/userprofile/secretkey`	GET allows read POST allows create DELETE allows delete	Used by Oracle Mobile Authenticator to read, create or delete secret key for given user. The secret key is used by OMA to generate a one time pin.

User Profile Services receives and responds to HTTPS requests using the service-specific endpoints for Person, Group, and Relationship entities. Each service endpoint can be individually disabled if it is not needed. If there are users across multiple user repositories, you can create multiple instances of User Profile Services; for example, if a company uses different repositories for different organizations this would be useful. Creating multiple user profile services may not be common though. See Section 48.2.5, "Introducing User Profile Services" for more information. For information on configuring User Profile Services, see Section 2.3.6, "Configuring User Profile Services." The following sections contain details on specific User Profile Services configurations.

- Using Proxy Authentication
- Securing User Profile Services Activity
- Understanding the Entity Relationship

1.4.5.1.1 Using Proxy Authentication Proxy authentication allows a user to control the security of middle tier applications by preserving client identities and privileges through all tiers, and auditing actions taken on behalf of clients. For example, this feature allows the identity of a user using a web application (also known as a "proxy")

to be passed through the application to the database server. Oracle Unified Directory (OUD) and Active Directory (AD) are few directory servers that support proxy authentication. Proxy authentication delivers the following security benefits.

- A limited trust model, by controlling the users on whose behalf middle tiers can connect, and the roles the middle tiers can assume for the user.
- Accountability, by preserving the identity of the real user through to the database, and enabling auditing of actions taken on behalf of the real user.
- Flexibility, by supporting environments in which users are known to the database, and in which users are merely "application users" of which the database has no awareness.

Oracle Access Management provides the ability to add proxy authentication features on top of directory servers that do not support it. The Access Control option is simply Proxy Authentication support for directory servers that do not have built in support for proxy authentication.

Proxy Authentication and Access Control were previously available in Mobile Services and now this support is available in OAuth Services. Without Proxy Authentication or Access Control enabled, communication with directory servers is done using the administrator user account; in this case, the administrator can perform any operations on any user. By enabling this feature, the logged in user can only perform operations for which privilege has been granted.

1.4.5.1.2 Securing User Profile Services Activity Security considerations are very important when implementing User Profile Services. For example, a user with write access to UserProfile.me can change their own UID or mail address causing a serious breach. Because of this it is possible to limit the scope of all URI to read only and you should be careful about granting write access to any scope. You can also configure read and write access independently on a per-attribute basis.

Security protection is defined within the Scopes table of the configured User Profile Service. Adding a URI allows you to select whether the service endpoint is enabled, whether read or write access is allowed, whether the URI is protected by an access token, and whether user consent is required.

The Oracle Access Management Console also allows fine-grained configuration of the attributes that can be modified. You can also add custom attributes or remove default attributes. Table 1–2 documents the out-of-the box configurable attributes for each scope setting.

Table 1–2 User Profile Resource Server - Scope Settings

Scope	HTTP(S) Method	Resource URI	Attributes
`UserProfile.me`	GET PUT	`/me`	`uid, mail, description, commonname, firstname, lastname`
`UserProfile.users`	GET POST PUT DELETE	`/users`	`uid, mail, description, commonname, firstname, lastname`

Table 1–2 (Cont.) User Profile Resource Server - Scope Settings

Scope	HTTP(S) Method	Resource URI	Attributes
UserProfile.groups	GET POST PUT DELETE	/groups	name, description
UserProfile.secretkey.management	GET POST DELETE	/secretkey	There are no attributes needed.

1.4.5.1.3 Understanding the Entity Relationship

An entity relationship is an association between two entities such as Users and Groups. The entity types can be the same or different. For example, the memberOf entity is a relationship between a user and a group while the manager entity is a relationship between two users. Client applications can create, read or delete relationships using the User Profile Services relationship endpoint. The following REST operations illustrate how to create a memberOf relationship. In these examples, the relationship endpoint is memberOf, the source entity URI is user-uri and the destination-entity URI is group-uri.

```
Create User "John"

curl -H "Content-Type: application/json" --request POST http://localhost:port/ms_
oauth/resources/userprofile/users -d '{"uid":"John"Anderson","commonname":"John
Anderson","firstname":"John"}'


Create Group "Group1"

curl -H "Content-Type: application/json" --request POST http://localhost:port/ms_
oauth/resources/userprofile/groups -d  '{"description":"group1
testing","commonname":"group1"}'


Create memberOf relationship

curl -H "Content-Type: application/json" --request POST http://localhost:port/ms_
oauth/resources/userprofile/users/memberOf -d '{"group-uri":"\/idaas_
rest\/rest\/userprofile\/group\/group1", "user-uri":"\/idaas_
rest\/rest\/userprofile\/people\/John"}'
```

1.4.5.2 Understanding Consent Management Services

The default Consent Management Services configuration is labeled **ConsentManagement** and handles consent storage, retrieval, revocation, and consent validation operations. If you select the **Require User Consent** option Oracle Access Management displays to the user a consent form so that access to the requested resource can be approved or denied by the user. The Require User Consent option can be enabled on a scope by scope basis. For example, you can require user consent for a scope request that allows "write" access but not "read" access. Consent data is stored in the Oracle Access Management database. For information on configuring Consent Management Services, see Section 2.3.7, "Configuring Consent Management Services."

Note: The Clients configuration page has a **Bypass User Consent** option. If this option is selected, the Client setting overrides the Resource Server setting. For information on configuring Clients, see Section 2.3.3, "Configuring Clients."

Any consent operation requires an access token of the Client Credentials grant type (as discussed in Understanding Clients) and the desired Consent Management scope. A user (through the client) requests access to a resource protected by OAuth Services. The request contains an access token, the client identifier and the user identifier. OAuth Services retrieves the configured scopes and, if allowed, grants consent by adding the scope to the access token. The access token is added to the authorization header in the HTTP request which is used to retrieve, grant or revoke consent using the endpoints provided by the Consent Management Service. For details on using REST interfaces to interact with Consent Management Services, see *Oracle Fusion Middleware Developer's Guide for Oracle Access Management*.

Note: Multiple Consent Management Services are not necessary.

1.4.6 Understanding Plug-Ins

Plug-ins enhance security by consulting additional logic for trust and risk analysis. (Such additional logic may deny certain risky operations.) Plug-ins apply the logic during authentication operations, including client application registration for mobile apps. The following plug-in types are available for use with OAuth Services and Mobile OAuth Services.

- The *Custom Token Attributes Plug-in* defines security policy around the token service provider.
- The *Authorization and Consent Service Plug-in* defines security policy around interactions where authorization and user consent are granted. This plug-in type can influence claims in a generated token as well.
- The *Client Plug-in* defines a security policy for Clients in an Identity Domain.
- The *Resource Server Profile Plug-in* defines a security policy for Resource Servers in an Identity Domain.

The following plug-in types are available for use with Mobile OAuth Services only:

- The *Mobile Security Manager Plug-in* is for use with Oracle Mobile Security Suite (OMSS). The Mobile Security Manager (MSM) component (part of OMSS) collects a rich set of mobile device data. This plug-in gathers that information and also invokes the MSM compliance policy, which checks the compliance status of the device. Finally, the plug-in sends the device information and the compliance status to the Adaptive Access Plug-in. For information about configuring the Mobile Security Manager plug-in, see Section 2.3.8.2, "Understanding the Plug-in Configuration Page."
- The *Adaptive Access Plug-in* is for use with Oracle Adaptive Access Manager (OAAM). It runs fraud detection and risk analysis policy checks that further validates that the user connection is authentic and can be trusted. The Adaptive Access Plug-in can utilize mobile device attribute values collected by the Mobile Security Manager Plug-in, or, if Oracle Mobile Security Suite is not available, the Adaptive Access plug-in can use mobile device attribute values that the Mobile OAuth Services obtains during mobile app requests. If the Mobile Security

Manager Plug-in is active, it runs first and passes device data to the Adaptive Access Plug-in, which runs second.

For more information, see Section 1.9, "Understanding OAuth Services Plug-ins."

For each plug-in type only one instance can be active at a time at the service profile level. For example, you can create and save different instances of the Client Plug-in at the Identity Domain level, but at the Service Profile level you can only assign one Client Plug-in instance at a time. Optional plug-ins can be configured to provide additional security. For information on configuring plug-ins, see Section 2.3.8, "Configuring Plug-Ins."

1.4.7 Understanding Server Settings

The Server Settings page is for configuring general server settings for the Identity Domain under which it is accessed. For information on configuring Server Settings, see Section 2.3.9, "Configuring Server Settings."

1.4.8 Understanding Jailbreak Detection Policy

A preconfigured Jailbreaking Detection Policy for iOS devices can search for files that indicate a device is jail broken and, if found, deny that device access to OAuth Services. This setting tab is displayed and for use with Mobile OAuth Services only. For information on configuring the Jailbreak Detection Policy, see Section 2.3.10, "Configuring the Jailbreak Detection Policy."

1.4.9 Understanding Token Life Cycle Management

Use this screen to search for and revoke tokens that have been issued. You can search for tokens using criteria such as user ID, client ID/name, client IP address, service profile, assertion token category, and token creation/expiration time. For information on configuring Token Life Cycle Management, see Section 2.3.11, "Configuring Token Life Cycle Management."

1.5 Understanding OAuth Services Tokens

OAuth Services generates a Client Token, a User Token and an Access Token. The Client Token is generated by OAuth Services when using the Client Credentials grant type without any scope for confidential clients, or for mobile clients. The User Token is generated by OAuth Services using the User Credentials grant type without any scope. The Access Token is generated with supported grant types using scope parameters. See Understanding Clients for details on grant types.

Note: A user and client can provide credentials in the form of a JWT token or an assertion for verification and generation of one of the three tokens. It is possible to use a Client Token or a User Token generated by OAuth Services as a Client Assertion or a User Assertion respectively but that is not common.

The Refresh Token is also generated by OAuth Services. It is issued when an offline scope is presented in the Access Token request and usually has a higher expiration time than the Access Token. A Refresh token can be used to get an Access Token. The following sections contain additional details.

- Understanding OAuth Services Access Tokens

- Understanding OAuth Services Refresh Tokens
- Understanding Mobile OAuth Services Client Tokens

1.5.1 Understanding OAuth Services Access Tokens

If OAuth Services determines that a user must consent to the request for access to a protected resource, a consent form is displayed. After the user consents, OAuth Services returns an authorization code to the Client service provider. The Client then sends the authorization code to the Token Endpoint and requests an OAuth Services Access Token. (When making the request, the Client authenticates with OAuth Services.) If received, the Access Token allows access to the protected resources. See Understanding the Authorization and Authentication Endpoints for details on the Token Endpoint.

Oracle Access Management can embed custom attributes in Access Tokens. Custom attributes are configured as part of the Service Profile or the Custom Resource Server. They are defined as static or dynamic.

- **Static Attributes** - Attribute name and value pairs where the value is fixed at the time that you define the attribute. For example, `name1=value1`.
- **Dynamic Attributes** - User-profile specific attributes. You must also configure the **User Store** setting on the Service Profile Configuration page. This setting defines the source of the User Profile attributes. The User Profile Service (and/or the underlying IDS interface) may be used to retrieve attribute names and values. Because dynamic attributes are user related, the user consent page (if configured) shows that the configured attributes are being shared with clients and resources.

Section 2.3.2, "Configuring Service Profiles" and Section 2.3.2, "Configuring Service Profiles" contain more information. Keep the following guidelines in mind when configuring custom attributes:

- Do not use the same name for a static and dynamic attribute.
- Avoid using the same name when adding custom attributes to the service profile configuration and the scope configuration. If you define the same attribute name in both locations, the scope-based attribute value takes precedence.

Custom attributes appear as claims in access tokens. JWT-based access tokens contain standard JWT claims along with OAuth Services specific ones. For example:

- Standard

```
"exp":1357596398000,
"iat":1357589198000,
"aud":"oam_server1",
"iss":"OAuthServiceProfile",
"prn":null,
"jti":"340c8324-e49f-43cb-ba95-837eb419e068",
```

- OAuth Services Specific

```
"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"
```

These claims are available as part of the access token generated by OAuth Services. Because the custom attributes appear as claims in a JWT-based access token, the following naming restrictions apply:

- Avoid JWT standard claim names.
- Avoid names with an "Oracle" prefix (as shown above)

1.5.2 Understanding OAuth Services Refresh Tokens

OAuth Services can be configured to allow the Client to use a refresh token to obtain additional access tokens with identical or narrower scope. The refresh token is used when the access token is no longer valid. The purpose of a refresh token is to improve security. Access tokens are short-lived, so if stolen, they are only useful for a limited period. Refresh tokens are longer-lived, but are less frequently sent to the server, thus reducing the likelihood that they will be stolen.

Any scope can request and use a refresh token, however, the refresh token is typically used when the user is offline. When configuring a Resource Server, the administrator can designate one scope to be the offline scope. If an access token request includes the scope designated as the offline scope, the server will include the refresh token with the access token. If the offline scope field is not configured, the server will not issue a refresh token. See Section 2.3.5.3, "Understanding the Custom Resource Servers Configuration Page" for details.

The client must be configured to use the refresh token. See Section 2.3.2.3, "Understanding the Service Profile Configuration Page" and Section 2.3.3.3, "Understanding the Web Clients Configuration Page" for information about refresh token settings.

1.5.3 Understanding Mobile OAuth Services Client Tokens

Mobile applications must register with Oracle Access Management prior to using OAuth Services and each registration is specific to one app on the device. After the application registration, the mobile app will have a Client Token. It uses this token as the security credential for making Access Token requests. See Understanding Mobile OAuth Services Server-Side Single Sign-on for details.

1.6 Understanding the Authorization and Authentication Endpoints

OAuth Services has four authentication endpoints that receive and respond to HTTPS requests: *the authorization endpoint*, *the token endpoint*, *the push endpoint*, and the *user consent revocation endpoint*. Each endpoint is a URL that clients use to make requests.

- **Authorization Endpoint** – The client uses the Authorization Endpoint to get authorization from the resource owner to access the requested resources. The client application initiates the Authorization Endpoint request by sending its identifier, a requested scope defining the resource to which it wants access, and a redirection URI to which OAuth Services will direct the web browser once access is granted or denied. This endpoint accepts the HTTPS request. The URI for this endpoint always ends in *authorize*. For example:

  ```
  http(s)://<host>:<port>/ms_
  oauth/oauth2/endpoints/<yourOauthServiceName>/authorize
  ```

- **Token Endpoint** – The client application interacts with the Token Endpoint to exchange an authorization code grant for an access token. It is also used for Client Credentials grant type and resource owner credentials grant type to get an access

token. The client uses a Refresh token to obtain a new access token. The URI for this endpoint always ends in *token*. For example:

```
http(s)://<host>:<port>/ms_
oauth/oauth2/endpoints/<yourOauthServiceName>/token
```

- **Push Endpoint** – Mobile OAuth Services client apps interact with the push endpoint to obtain (depending on configuration) part of the authorization codes, and/or part of the client tokens, access tokens, and refresh tokens that are sent through either the Apple Push Notification Service (APNS) or the Google Cloud Messaging (GCM) service. It can also be used for Mobile Client Verification code, Authorization Code and Client Tokens. For example, the end point for requesting data from APNS is:

  ```
  http(s)://<host>:<port>/ms_oauth/oauth2/endpoints/oauthservice/push
  ```

- **User Consent Revocation Endpoint** - Resource owners (end-users), who authenticate and authorize client applications using the browser-based authorization endpoint flow, use this endpoint to revoke their consent to client applications. For example:

  ```
  http(s)://<host>:<port>/ms_
  oauth/oauth2/ui/<yourOauthServiceName>/showrevokeconsent
  ```

When configuring clients with authorization code grant in the OAuth server, you also need to provide at least one client redirect URI where the server can return authorization credentials to the client.

- **Client Redirect URIs** – The OAuth Services server returns authorization credentials to the client using the URI specified in the request provided that it exactly matches a URI configured in the client profile.

1.7 Enforcing Access Control

Typically, an OAuth Services client application makes REST calls to services deployed on remote servers. These calls, carrying an access token, need to be validated before the call can go through. Enforcing access control is accomplished by sending a previously obtained access token to a resource server defined in OAuth Services. Exceptions to this are the native User Profile and Consent Management Services that are enforced by OAuth Services.

The options for validation within the Oracle stack are Oracle API Gateway (OAG) and Oracle Web Services Manager. (An OAG filter validates the Oracle Access Management OAuth Services token before allowing access to the resource.) Custom code can also be written to provide access control.

Note: WebGates do not support validating access tokens.

1.8 Understanding Mobile OAuth Services Server-Side Single Sign-on

The server-side single sign-on (SSO) feature allows multiple mobile apps on a device to share a single user session that resides on the OAM server and not with the client. This feature saves JWT and OAM user tokens in the Server-Side Device Store, and maintains the user session in the browser with cookies. Thus, the server session is not tied to the client. Session time-out values are configurable at the Service Profile level for the client token, user token, and access token. The access token time-out value can also be overridden at the Resource Server level.

Keeping sensitive session info on the server (and not on the device) reduces the risk of the tokens being copied if the device or client app is compromised. For 2-legged flows, if server-side single sign-on is turned off (it is on by default), the user token is not stored on the server but sent to the client on the mobile device. You can enable and disable this feature using the Mobile OAuth Services Service Profile configuration page. For 3-legged flows, server-side single sign-on is always automatically enabled.

- Understanding the Server-Side Single Sign-On Credential Collection Options
- Understanding Server-Side SSO For Mobile OAuth Services 3-Legged Flows
- Understanding Server-Side SSO For Mobile OAuth Services 2-Legged Flows

1.8.1 Understanding the Server-Side Single Sign-On Credential Collection Options

Developers in your organization can implement single sign-on in a client app by using an external browser, an embedded browser, or by using a native app registered with Mobile and Social Services as an SSO proxy. This section briefly discusses the different approaches.

- Using the External Browser Approach
- Using the Embedded Browser Approach
- Using the Native App Proxy Approach

1.8.1.1 Using the External Browser Approach

In this approach the mobile app switches to an external browser, which executes the logic for user authentication and user consent management. A shared browser cookie maintains the user session and can be used to provide SSO across multiple apps. The external browser uses a typical Web SSO mechanism that supports OAM user authentication, JWT user authentication, third-party user authentication, and social authentication (using the Social Identity service). One drawback to using an external browser is the screen “flickering” that occurs when the application context switches between the browser and the application.

1.8.1.2 Using the Embedded Browser Approach

In this approach the mobile app uses its own embedded browser. Because the browser cookie cannot be shared across multiple apps, OAM and third-party user authentication cannot be used. Instead, Mobile OAuth Services uses a JWT user session token stored in the Server-Side Device Store. When additional apps are launched, SSO is established using device identification together with the shared JWT user session token. The embedded browser approach eliminates the screen “flickering” that occurs when the application context switches between the application and the external browser (as discussed in Using the External Browser Approach).

1.8.1.3 Using the Native App Proxy Approach

In this approach, if a native app is already installed on the device, it can facilitate SSO by serving as a proxy between the browser-based app and the Mobile OAuth Services SSO Servlet on the OAM server. As needed, the Servlet registers the device and app with OAM and obtains the tokens required to authenticate the app in the browser so that it can access the OAM-protected resource. To use this approach, native apps must use Mobile and Social Services to authenticate with Access Manager. This approach can be used for 2-legged flows.

1.8.2 Understanding Server-Side SSO For Mobile OAuth Services 3-Legged Flows

Server-side SSO is always enabled for Mobile OAuth Services 3-legged scenarios. The Mobile OAuth Services server executes the logic for user authentication and user consent management, collects the user credentials, and provides the required SSO functionality. Users log in once and gain access to all systems without being prompted to log in again. For JWT, third-party, and social authentication, the OAM server stores the user token in the Device Store, whereas OAM authentication stores the OAM_ID (representing an OAM session) in the Device Store. When the user token or OAM_ID expires, the user is prompted to log in again.

When implementing mobile 3-legged scenarios, enable the **Authorization Code** grant type by going to the Mobile OAuth Services Clients configuration page in the Oracle Access Management Console. Apps should implement the mobile 3-legged flows documented in the *Oracle Fusion Middleware Developer's Guide for Oracle Access Management*.

Apps that use a Mobile OAuth Services 3-legged scenario should collect credentials using a mobile browser (as discussed in Understanding the Server-Side Single Sign-On Credential Collection Options). To specify an authentication type using the Oracle Access Management Console, open the Mobile OAuth Services Service Profile configuration page and under **Mobile Service Settings**, choose an option for **Consent Service Protection**. The following sections describe these options.

OAM Authentication or Third-Party Access Management

If using OAM authentication or Third-Party Access Management, Access Manager (or a third-party access management product) is responsible for user authentication. Any OAM authentication method that supports login through the browser is an option. Because Access Manager supports SSO using an OAM_ID cookie, this flow must use an external browser (as discussed in Using the External Browser Approach). Following authentication, the user token is stored in the Server-Side Device Store.

Oracle Access Management supports session synchronization so that 2-legged flows can get OAM tokens later. For example, the first app registers with the server using the 3-legged flow. Next, a second app completes the 2-legged registration using the existing session established by the first app. Thus a 3-legged flow session can register additional apps using a 2-legged flow by using the same session created during the first app registration.

JWT Authentication

JWT authentication is an authentication mechanism provided by Mobile and Social for mobile applications. In this case, Mobile and Social hosts the user interface for user login. It accepts a user name and password for authentication, using the configured user store for user authentication. The user store can be configured in Mobile and Social or in Oracle Access Management. It is configured using the User Authenticator under User Store in OAuth Services profile.

As discussed previously, JWT user authentication is the only authentication type that supports single sign-on using an embedded browser. (If the ability to use multiple apps on the same mobile device is not a requirement, then either OAM authentication or third-party authentication is sufficient.) JWT authentication can also be used with external browsers, but the application will need to switch from the app to the browser and back again when Mobile and Social checks the user token in the Device Store for single sign-on.

Social Authentication

Social authentication allows app users to authenticate using social identity providers such as Facebook and Twitter. This type of authentication requires the Oracle Access Management Social Identity service (part of Mobile and Social). If using social authentication, the Oracle Access Management server redirects the user to the social identity provider for authentication. For single sign-on, an external browser is required. Following authentication, the user token is stored in the Server-Side Device Store. Social Authentication is supported for 3-legged flows only.

1.8.3 Understanding Server-Side SSO For Mobile OAuth Services 2-Legged Flows

Apps that use Mobile OAuth Services 2-legged scenarios should collect user credentials using the native iOS or Android user interface although you are limited to username/password-based OAM authentication or IDS (directory server). By switching to an external browser, you can use the usual supported authentication schemes, including OAM user authentication, third-party authentication, and JWT user authentication.

Unlike with mobile 3-legged scenarios, you can choose to disable the server-side single sign-on feature with mobile 2-legged scenarios. To disable it, open the Mobile OAuth Services Service Profile Configuration page and clear the **Enable Server-Side Single Sign-On** option. (Server-Side SSO can also be set using WLST.)

- If Server-Side SSO is enabled, the server collects user credentials on behalf of the client and provides SSO. Additional apps (that is, apps that share the same device profile) have to register with the server the first time they are launched. You can require the user to authenticate every time an app is registered, or you can allow the registration to happen automatically in the background without involving the user. Configure the `msAlwaysShowLogin` attribute on the Service Profile page to select the desired behavior.
  - If the `msAlwaysShowLogin` Service Profile attribute is set to *true*, the user has to enter a user name and password in the native app to register the app and get a client token.
  - If the attribute is set to *false*, the server automatically registers apps using the server-side user token.

  Once registered, subsequent access requests from apps on the device typically result in the server providing a client token and allowing access. (In some cases the user may not get access right away—for example, if Oracle Adaptive Access Manager rules are active.)
- If Server-Side SSO is disabled, *the client* collects the user credentials and must also provide SSO. On a first access attempt the user enters a user name and password in the native app to register the device. The server returns a user token that the device stores locally. Subsequent access requests from apps on the device must use the stored user token to register the apps and gain access.

For mobile 2-legged scenarios, enable the **Resource Owner** grant type by going to the Mobile OAuth Services Clients configuration page in the Mobile Oracle Access Management Console. Apps should implement the mobile 2-legged flows documented in the *Oracle Fusion Middleware Developer's Guide for Oracle Access Management*. The following sections give an overview of SSO options.

- Using the OAuth Mobile SSO Servlet Authentication
- Using SSO Between Native Apps and an External Browser

1.8.3.1 Using the OAuth Mobile SSO Servlet Authentication

If a user using a mobile browser (external or embedded) needs to access a protected Web resource, and there is a native app that uses Mobile and Social Services to authenticate with OAM already installed on the device, the native app can register the device on behalf of the Web app and gain access to the resource without requiring the user to sign on. This single sign-on approach utilizes the OAuth Mobile SSO Servlet that was added to the OAM server in release 11.1.2.3. For configuration steps, see Section 2.7, "Configuring Mobile OAuth for SSO Servlet Authentication."

1.8.3.2 Using SSO Between Native Apps and an External Browser

Another alternative utilizing a native app and an app running in an external browser is to implement 2-legged flows in the native app using the OAuth Services REST API. (Server-side SSO can be enabled or disabled.) The native app registers the device and user, and exchanges an OAM token to get an OAM_ID cookie (which is the OAM master token). The native app can then launch the external browser and inject the OAM_ID cookie so that users accessing WebGate-protected resources with an external browser will not be prompted to log in every time. For information about implementing the 2-legged flows, see the *Oracle Fusion Middleware Developer's Guide for Oracle Access Management*.

1.9 Understanding OAuth Services Plug-ins

Oracle Adaptive Access Manager (OAAM) is an optional product that can screen Mobile OAuth Services transactions using a provided security plug-in. Together, OAAM and the plug-in provide mobile-client fraud detection, knowledge-based authentication (for two-factor authentication after user name and password authentication), and one time password functionality. If the Oracle Mobile Security Suite is deployed, the Mobile Security Manager plug-in gathers additional mobile device data for OAAM to screen.

To use OAAM with Mobile OAuth Services, the Adaptive Access security plug-in must be installed. This plug-in can add value during app registration when client tokens or user tokens are being validated or refreshed, and during token exchange.

OAAM rules and policies are defined in Oracle Adaptive Access Manager. The following is a brief description of the OAAM and Adaptive Access Plug-in features.

- The Adaptive Access Plug-in enhances security by screening mobile app registration requests for both 2-legged and 3-legged flows. The plug-in runs fraud detection and risk analysis policy checks.
- Knowledge-based authentication (KBA) and one time password authentication (OTA) can also be integrated into the mobile app registration process. The OAuth Service REST API flows include sample challenge requests and responses that a developer will need to implement in your app(s).
- Following registration, the Adaptive Access Plug-in screens user tokens for security violations instead of simply checking if the user token is valid. The result of this screening is either *allowed* or *denied*.

Using the Mobile Security Manager Plug-in Together With the Adaptive Access Plug-in

The Mobile Security Manager Plug-in is for use with Oracle Mobile Security Suite (OMSS). The Mobile Security Manager (MSM) component (part of OMSS) collects a rich set of mobile device data and passes it to the Adaptive Access Plug-in for use by OAAM. If Oracle Mobile Security Suite is not available, the Adaptive Access plug-in

uses mobile device attribute values that the Mobile OAuth Services server obtains during mobile app requests.

> **Note:** The Mobile Security Manager plug-in requires special configuration before it can be used. See Section 2.8, "Configuring the Mobile Security Manager Plug-in" for details.

If the Mobile Security Manager Plug-in is active, it runs first and sends its data to the Adaptive Access Plug-in, which runs second. The Adaptive Access Plug-in checks the results of the MSM compliance policy that reports the compliance status of the device. If the compliance policy response is negative, the Adaptive Access Plug-in denies the mobile app request; If the response is positive, the Adaptive Access Plug-in passes the device data to Oracle Adaptive Access Manager for stronger authentication checks and risk evaluation. The Mobile Security Manager plug-in gets device info and checks the MSM compliance policy in the following cases:

- During the app registration flow following user authentication.
- As part of the client token and user token validation process.

For more information:

- See "Understanding OAuth Services Authorization for Mobile Clients" for a detailed look at when in the Mobile OAuth Services flow OAAM interacts with mobile apps.
- See Section 1.4.6, "Understanding Plug-Ins" for summary information about the various OAuth Services plug-ins.
- See Section 2.3.8, "Configuring Plug-Ins" for information about configuring the adaptive-access plug-in.

2

Configuring OAuth Services

Oracle Access Management provides a graphical user interface for configuring OAuth Services. The configuration options are available within the Identity Federation or Mobile Services interfaces depending on the licensing procured.

This chapter describes how to use the Oracle Access Management Console to enable OAuth Services and configure the OAuth Services components.

- Enabling OAuth Services
- Configuring OAuth Services Components in an Identity Domain
- Configuring OAuth Services Settings
- Configuring OAuth Services for Third-Party JWT Bearer Assertions
- Configuring a WebGate to Protect OAuth Services
- Configuring OAM Session Synchronization
- Configuring Mobile OAuth for SSO Servlet Authentication
- Configuring the Mobile Security Manager Plug-in

> **Note:** OAuth Services can be configured from the command line using WLST. For more information about the Mobile and Social WLST commands, see the *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference.*

2.1 Enabling OAuth Services

Oracle Access Management OAuth Services has to be explicitly enabled in order to use it. A license for Oracle Access Management Identity Federation (if using web clients only) or Mobile and Social (if using web and mobile clients) is required to enable it. Once correct licensing is procured, enable Identity Federation or Mobile and Social by clicking Available Services in the Configuration Launch Pad of the Oracle Access Management Console. These section links contain more details.

- Section 2.4, "Understanding the Oracle Access Management Console"
- Section 3.2, "Enabling or Disabling Available Services"
- Section 37.8, "Enabling Identity Federation"
- Section 48.1.3, "Enabling Mobile and Social"

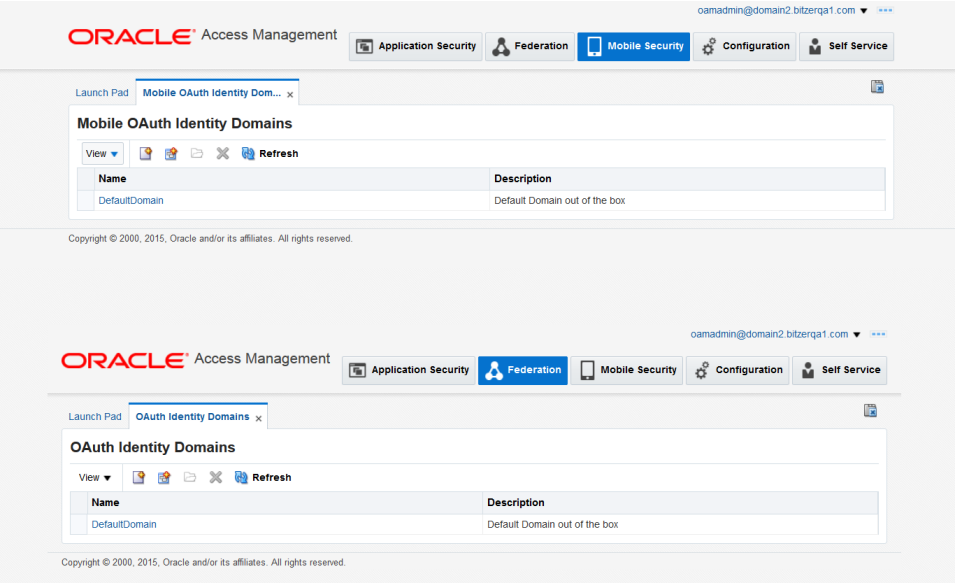
2.2 Configuring OAuth Services Components in an Identity Domain

In order to use Oracle Access Management OAuth Services, you will need to configure an Identity Domain. OAuth Services ships with a default Identity Domain named **DefaultDomain**. You can create additional domains as needed. Each OAuth Services Identity Domain has a universally unique identifier (UUID) that specifically identifies it on the Internet.

Access to the Identity Domain configuration page is dependent on whether you have enabled Identity Federation OAuth Services or Mobile and Social OAuth Services. Figure 2–1 contains two screenshots: the top displays the Mobile OAuth Identity Domains and the bottom displays the Federation (web only) OAuth Identity Domains. Note that both contain the DefaultDomain. To access the appropriate page, do the following.

- Click **Mobile Security** at the top of the Oracle Access Management Console and then click Mobile OAuth Services.
- Click **Federation** at the top of the Oracle Access Management Console and then click OAuth Services.

Figure 2–1 Mobile (top) and Federation (bottom) Identity Domain Screens

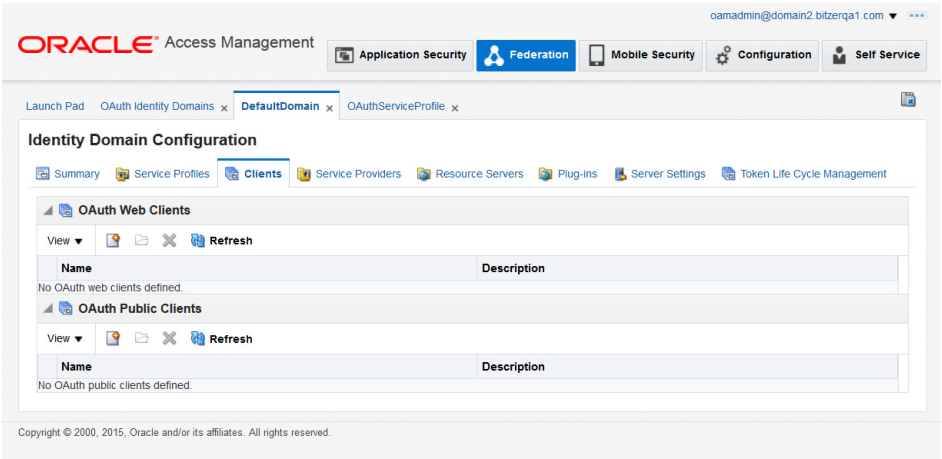


Screenshots of Mobile and Federation (web only) Identity Domain pages

****

Figure 2–2 is a screenshot of the Identity Federation OAuth Services DefaultDomain configuration page. To access this page, click DefaultDomain (or any custom domain that might have been created) from the Identity Federation Identity Domain configuration page. (If OAuth Services is accessed by clicking a Mobile Security configured domain, a third OAuth Mobile Clients table is also displayed on the Clients tab of the selected domain.)

Figure 2–2 Identity Federation DefaultDomain Configuration Page

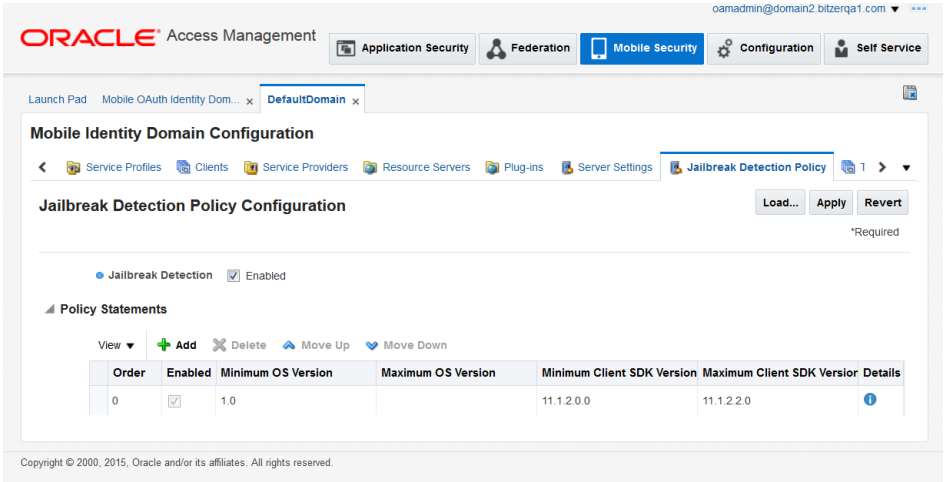


Screenshot of Identity Federation Default Domain configuration

****

When accessing the Identity Domain configuration page through Mobile Security, the Jailbreak Detection Policy tab is included with those previously listed for Identity Federation. Figure 2–3 is a screenshot of the Mobile Security OAuth Services DefaultDomain configuration page with the Jailbreak Detection Policy page displayed. To access this page, click DefaultDomain (or any custom domain that might've been created) from the Mobile Security Identity Domain configuration page.

Figure 2–3 Mobile Security DefaultDomain Configuration Page



Screenshot of Mobile Security DefaultDomain configuration

****

The Identity Domains pages list all configured OAuth Services Identity Domains. When the list is displayed, you can create a new Identity Domain by clicking Create Using Single Step or Create Using Wizard Flow. Click a domain name to modify an already configured profile.

2.3 Configuring OAuth Services Settings

OAuth Services has many components that must be configured before the authorization protocol can be used. Descriptions of the OAuth Services components and how they work together can be found in Section 1.4, "Understanding the OAuth Services Components." This section includes information on configuring the OAuth Services components using the Oracle Access Management Console only. It contains the following topics:

- Configuring Identity Domains
- Configuring Service Profiles
- Configuring Clients
- Configuring the Service Provider
- Configuring Custom Resource Servers
- Configuring User Profile Services
- Configuring Consent Management Services
- Configuring Plug-Ins
- Configuring Server Settings
- Configuring the Jailbreak Detection Policy
- Configuring Token Life Cycle Management

2.3.1 Configuring Identity Domains

See Section 2.2, "Configuring OAuth Services Components in an Identity Domain" for introductory information about Identity Domains. The following section describes how to use the user interface to configure an Identity Domain. It includes the following topics:

- Creating an Identity Domain
- Editing or Deleting an OAuth Identity Domain
- Understanding the Identity Domain Configuration Page - Summary Tab
- Understanding the Create Identity Domain Wizard Flow

2.3.1.1 Creating an Identity Domain

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain."
2. Choose one of the following:
   - To quickly create an Identity Domain with only basic information, click **Create using single step** (leftmost **+** button in the toolbar).

     The Identity Domain Configuration page opens.

     Complete the form and click **Create** to save your changes. You will need to provide additional configuration detail later.
   - To create an Identity Domain *and* configure essential Service Profile settings, click **Create using wizard flow** (rightmost **+** button in the toolbar).

     The Create OAuth Identity Domain wizard flow page opens.

Click **Back** and **Next** to move backwards and forward through the wizard flow. Click **Finish** to save your changes.

2.3.1.2 Editing or Deleting an OAuth Identity Domain

1. Open the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain."
   - To view or edit an Identity Domain, click its name in the table.
   - To delete an Identity Domain, select it by clicking the column to the left of the domain name and then click the delete button in the command bar.

2.3.1.3 Understanding the Identity Domain Configuration Page - Summary Tab

This section describes the form fields on the Identity Domain Configuration **Summary** tab when viewing an existing identity domain or creating a new one.

Identity Domain - The name of the identity domain. If creating or editing an identity domain, type a unique name without spaces.

Description - (Optional) A short description to help you or another administrator identify this identity domain in the future.

Identity Domain UUID - The identification code that uniquely identifies this identity domain on the Internet. Click **Generate** to populate this field with a universal unique identifier code.

Allow Multiple Resource Servers - Select this option if the identity domain supports more than one resource server.

> **Note:** Selecting multiple resources requires that scopes are prefixed with the Resource Server name. For example, if you add PhotoService as a Resource Server, the scopes must be prefixed with PhotoService. This is done automatically while adding scopes in the Resource Server. The prefix can be changed to something different but unique.

The fields listed below appear on the Create Identity Domain page.

Service Profile

(Service Profile) Name - The name of the identity domain's service profile. Each identity domain requires at least one service profile. See Section 1.4.2, "Understanding Service Profiles" for more information.

(Service Profile) Endpoint - The URL where the OAuth authorization service for this identity domain responds to authorization requests.

User Profile Service

(**User Profile Service) Name** - The name of the identity domain's user profile service. A user profile service is created automatically for each identity domain. See Section 1.4.5, "Understanding Resource Servers" for more information.

(User Profile Service) Endpoint - The URL where the User Profile Service receives and responds to create, read, update, and delete requests.

Consent Management Service

(**Consent Management Service) Name** - The name of the identity domain's consent management service. Each identity domain must have a consent management service,

which stores and retrieves consent records, and performs consent validation and consent revocation operations. See Section 1.4.6, "Understanding Plug-Ins" for more information.

(Consent Management Service) Endpoint - The URL where the Consent Management Service receives and responds to client and resource owner service requests.

2.3.1.4 Understanding the Create Identity Domain Wizard Flow

For help understanding the form fields on the Create OAuth Identity Domain wizard flow pages, refer to the following sections.

- **Information** - For help, see Section 2.3.1.3, "Understanding the Identity Domain Configuration Page - Summary Tab."
- **Service Profile** - For help, see Section 2.3.2.3, "Understanding the Service Profile Configuration Page."
- **Mobile Service** - For help, see "Mobile Service Settings" in Section 2.3.2.3.
- **Tokens** - For help, see "Tokens (Token Settings)" in Section 2.3.2.3.
- **Summary** - Review your settings and click **Finish** to create the identity domain.

2.3.2 Configuring Service Profiles

See Section 1.4.2, "Understanding Service Profiles" for introductory information about Service Profiles. The following section describes how to use the user interface to configure a Service Profile. It includes the following topics:

- Creating a Service Profile
- Editing or Deleting a Service Profile
- Understanding the Service Profile Configuration Page

2.3.2.1 Creating a Service Profile

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it.
2. Select the **Service Profiles** tab.
3. Click **Create** to complete the wizard.

2.3.2.2 Editing or Deleting a Service Profile

1. Open the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click an identity domain to open it for editing.
2. Click the **Service Profiles** tab.
3. Do the following:
   - To edit a service profile, click its name in the table.
   - To delete a service profile, select it by clicking the box to the left of the name and then click the delete button in the command bar.

2.3.2.3 Understanding the Service Profile Configuration Page

Identity Domain - The name of the identity domain to which this service profile applies. (Read-only)

Name - The name of this service profile.

Description - (Optional) A short description to help you or another administrator identify this service profile in the future.

Service Enabled - Select to activate the service profile, or clear the option box to inactivate it.

Service Provider - The name of the OAuth Service Provider that corresponds with this OAuth Service Profile.

Service Endpoint - The URL where the OAuth authorization service responds to authorization requests.

User Store

User Authenticator - For user authentication, choose **OAM** to use the Oracle Access Management token provider, or choose **IDS** to use the Identity Directory Service token provider. Only choose IDS authentication if the OAM token is not used at all (for example, if only the JWT token is used). If both OAM and JWT tokens are used, choose OAM authentication to avoid duplicated authentication attempts sent by both IDS and OAM.

Identity Store Name - The name of the identity store when IDS is configured as the user authenticator.

User Profile Service

(**User Profile Service) Name** - The name of the identity domain's user profile service. A user profile service is created automatically for each identity domain. See Section 1.4.5.1, "Understanding User Profile Services" for more information.

(User Profile Service) Endpoint - The URL where the User Profile Service receives and responds to create, read, update, and delete requests.

Consent Management Service

(**Consent Management Service) Name** - The name of the identity domain's consent management service. Each identity domain must have a consent management service, which stores and retrieves consent records, and performs consent validation and consent revocation operations. See Section 1.4.5.2, "Understanding Consent Management Services" for more information.

(Consent Management Service) Endpoint - The URL where the Consent Management Service receives and responds to client and resource owner service requests.

Plug-Ins

Choose available plug-ins from the menus in the following categories. See Section 1.4.6, "Understanding Plug-Ins" for more information.

Adaptive Access - Runs Oracle Adaptive Access Manager (OAAM) fraud detection and risk analysis policy checks, enhancing authenticity and the trust level of a user.

Mobile Security Manager - Gathers mobile device data from the Mobile Security Manager (MSM) component (part of Oracle Mobile Security Suite) and sends it, as well as the MSM compliance status, to the Adaptive Access Plug-in for stronger authentication checks and risk evaluation.

Custom Token Attributes - Defines security policy around the token service provider. See Section 1.5.1, "Understanding OAuth Services Access Tokens" for more information.

Client - Delegates the following to an external security module: confidential client authentication, client authorization, and client profile reading.

Resource Server Profile - Delegates the following to an external security module: confidential resource server authentication, resource server authorization, and resource server profile reading.

Authorization & Consent Service - Defines security policy around interactions where authorization and user consent are granted. This plug-in can influence claims in a generated token as well.

Attributes

Add or delete service profile attributes and their values to further configure the OAuth service profile.

For JWT token generation and validation, configure the following parameters:

- `jwt.cert.alias`
- `jwt.trusted.issuer.size`
- `jwt.trusted.issuer.1`
- `jwt.trusted.issuer.2`

> **Note:** For details, see Section 2.4, "Configuring OAuth Services for Third-Party JWT Bearer Assertions."

Table 2–1 OAuth Service Profile Configuration Attributes

Name	Value	Notes
`jwt.cert.alias`		Private key alias name for the signing certificate in the keystore. The default alias will be used if this attribute is not specified.
`jwt.CryptoScheme`	`RS512`	The cryptographic algorithm used to sign the contents of the JWT token. The default value is `RS512`. (RSA encryption using SHA-512 hash algorithm.)
`jwt.issuer`	`www.oracle.example.com`	This issuer of the tokens (that is, the `iss` claim value in the JWT token generated by OAuth Services). The default value, `www.example.oracle.com`,needs to be changed in the deployment.
`jwt.trusted.issuer.size`	2	The number of trusted issuers. The value can be any number of trusted issuers. For example, if the number is 2, the following matching params need to be specified.
`jwt.trusted.issuer.1`		The alias name for the public key of the first trusted issuer in the key store. See `jwt.trusted.issuer.size` for details.
`jwt.trusted.issuer.2`		The alias name for the public key of the second trusted issuer in the key store. See `jwt.trusted.issuer.size` for details.
`createdByDefault`	`true`	If set to true, the current OAuth Services profile is created automatically as part of domain creation. Otherwise, it's created manually.

Table 2–1 (Cont.) OAuth Service Profile Configuration Attributes

Name	Value	Notes
clientPWDValidation	false	If set to true, a client ID and secret (password) can be used as credentials to interact with OAuth Services for token validation and termination requests. If set to false, only a JWT/SAML client assertion can be used as client credentials to interact with OAuth Services for token validation and termination requests.
tokenTenantClaimName	user.tenant.name	The tenant claim name in the tokens issued by OAuth Services. By default this is set using the identity domain name.
oauthServerSelfClientId	Value to be specified	By default this is set with the value of the jwt.issuer attribute. This attribute gets used when OAuth Services generates a client assertion for itself when interacting with other services such as service-to-service interactions.
oauthServerSelfCTValidityInSec	Value in seconds to be specified	The default value is 300sec. This attribute is related to oauthServerSelfClientId (that is, the OAuth Services own client assertion validity period).
msAlwaysShowLogin	true/false	This attribute applies to mobile clients using the JWT SSO authentication mechanism. It is used with 2-legged flows only. (For 3-legged flows, the browser manages the session.) **true** - The user must authenticate for each app registration. (Mobile apps are not registered using the server-side JWT user token.) OAuth Services shows a login page for the user to submit credentials. **false** - Mobile apps are registered using the server-side JWT user token. By default true.. If this attribute is not defined in the service profile, the server does not allow mobile apps to use the server-side user token to register without a user name and password. For more information see Section 1.8.3, "Understanding Server-Side SSO For Mobile OAuth Services 2-Legged Flows."

Mobile Service Settings

Supported Platforms - Choose iOS, Android, and/or Others:

- **iOS** - The authorization server accepts requests from iOS clients if selected.
- **Android** - The authorization server accepts requests from Android clients if selected.
- **Others** - The authorization server accepts requests from clients other than iOS or Android if selected.

iOS Security Level - Choose Advanced or Standard:

- **Advanced** - All client registrations and token acquisitions are done using both push notification and HTTP(S).

- **Standard** - All client registrations and token acquisitions are done using HTTP(S)

Android Security Level - Choose Advanced or Standard:

- **Advanced** - All client registrations and token acquisitions are done using both push notification and HTTP(S).
- **Standard** - All client registrations and token acquisitions are done using HTTP(S)

Android Sender ID - Enter the GCM sender ID that is required for Android push notification.

Android API Key - Enter the API key required for Android push notification.

Consent Service Protection - Authorization requests are routed to the consent service, which requires the user to log in and give consent. Select **OAM or Third-Party Access Management**, **JWT Authentication**, or **Social Authentication**.

- **OAM or Third-Party Access Management** - Use either Oracle Access Management or a third-party option for consent page protection.
- **JWT Authentication** - Use the OAuth server itself for consent page protection. If using the OAuth server for consent page protection, the authentication flow is determined by the **User Store** setting.
- **Social Authentication** - Use the **Social Identity** service for consent page protection.

Require User Consent for Client Registration - Select this option to require the user to give authorization before registering each Mobile OAuth application installation instance on a mobile device.

Enable Server-Side Single Sign-On - Determines if the server will provide single sign-on among multiple apps on the same device or if it is the client responsibility. Single sign-on is either achieved by storing a JWT user token or an OAM user token in the Server-Side Device Store, or by returning the user token to the client to manage. Server-side SSO applies to 2-legged Mobile OAuth flows only. If this option is selected, after registering the first app the server stores the user token and does not return it to the mobile device. If this option is not selected, the tokens are sent to the mobile device and are not stored in the Server Device Store. For more information, see Section 1.8, "Understanding Mobile OAuth Services Server-Side Single Sign-on."

Preferred Hardware IDs - Use the list to prioritize the hardware ID attributes that should be used to uniquely identify mobile devices. The first available hardware ID from the list will be used.

Mobile Client Attributes - Add or delete mobile client attributes and their values as needed if the server requires additional attributes.

Configuration Settings

Clients

Allow access to all clients - Select if all clients in the identity domain should use this service profile. Clear this option to select which clients will be able to access the service profile.

Client Table - Add to the table the clients that should be able to access the service profile. Click **Browse Clients**, then select the clients to add to the table. To assign a client to a different service profile, click the box to the left of the client name and click **Remove**.

Tokens (Token Settings)

Use this tab to configure token settings, as well as settings for custom attribute that OAuth Services should embed in access tokens.

Tokens

- **Token Name** - The name of the token.
- **Expires** - The length of time in minutes after which the token is no longer valid.
- **Refresh Token Enabled** - Select this option to allow a refresh token to be used. A refresh token cannot be used with a client verification code or an authorization code. See Section 1.5, "Understanding OAuth Services Tokens" for more information.
- **Refresh Token Expires** - The length of time in minutes after which the refresh token is no longer valid.
- **Life Cycle Enabled** - Select this option if OAuth Services should cache a token and save it in the database until the token expires.

Custom Attributes

Use this section to define custom attributes that OAuth Services embeds in the access tokens. See Section 1.5.1, "Understanding OAuth Services Access Tokens" for more information about custom attributes.

- **Static Attributes** - Attribute name and value pairs where the value is fixed at the time that you define the attribute. For example, `name1=value1`.
- **Dynamic Attributes** - User-profile specific attributes.

Resource Servers (Custom Resource Servers)

Use this tab to choose which custom resource servers clients should have access to. A custom resource server is any resource server that is not the User Profile and Consent Management resource servers that are included with OAuth Services.

Allow clients access to all resource servers - Select to allow clients to access all resource servers configured in the identity domain. Clear this option to select which resource servers clients will be able to access.

Available Servers / Selected Servers - Use the arrows to move the resource servers that clients should be able to access from the **Available Servers** box to the **Selected Servers** box. (This option is only available if the **Allow clients access to all resource servers** option is not selected.)

System Resource Servers

Use this tab to configure if clients should have access to the user profile service and/or consent management service.

User Profile Services -Use the arrows to move the user profile server that clients should be able to access from the **Available Servers** box to the **Selected Servers** box. Services listed in the **Selected Servers** box are active services.

Consent Management Services - Use the arrows to move the consent management server that clients should be able to access from the **Available Servers** box to the **Selected Servers** box. Services listed in the **Selected Servers** box are active services.

Trusted Issuers

Use this tab to add certificate issuers who can be used to validate tokens. Click **Add** to add a record to the table; select a row and click **Remove** to delete a record from the table.

Certificate Alias -The alias name.

Trusted Issuer - The name of the trusted certificate issuer.

Certificate Thumb Print - x5t - The base64url encoded digest of the DER encoding of the X.509 certificate corresponding to the key used to digitally sign certificates.

Key identifier - kid - The key ID value that indicates which key is used to secure certificates.

2.3.3 Configuring Clients

See Section 1.4.3, "Understanding Clients" for introductory information about OAuth Services Clients. The following section describes how to use the user interface to configure a Web client and a mobile client. It includes the following topics:

- Creating a Client
- Editing or Deleting a Client
- Understanding the Web Clients Configuration Page
- Understanding the Mobile Clients Configuration Page

2.3.3.1 Creating a Client

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it.
2. Select the **Clients** tab.
3. Click Create and a Create Client tab will open as follows.
   - To create an OAuth Services Web (non-mobile) client, click the Create button located directly under the **OAuth Web Clients** heading.

     See Understanding the Web Clients Configuration Page.
   - To create an OAuth Services Public client, click the Create button located directly under the **OAuth Public Clients** heading.

     See Understanding the Public Clients Configuration Page.
   - To create an OAuth Services mobile client, click the **Create** button located directly under the **OAuth Mobile Clients** heading.

     See Understanding the Mobile Clients Configuration Page.
4. Enter the appropriate values in the form displayed under the Create Client tab.

2.3.3.2 Editing or Deleting a Client

1. Open the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click an identity domain to open it for editing.
2. Click the **Clients** tab.
3. Do the following:

- To edit a client configuration, click its name on the page.

  The client configuration page opens in a new tab.

- To delete a client, select it by clicking the box to the left of the name and then click the delete button in the command bar.

2.3.3.3 Understanding the Web Clients Configuration Page

This section describes the form fields on the Web Client Configuration page when viewing an existing Web client or creating a new one. The *Mobile OAuth* Client Configuration page is described in the next section.

Identity Domain - The name of the identity domain in which this OAuth Web client is registered. (Read-only)

Name - The name of this OAuth client.

Description - (Optional) A short description to help you or another administrator identify this OAuth Web client in the future.

Allow Token Attributes Retrieval - Select this option to allow custom attributes (both attribute names and values) to be shared with resource servers and the resource owner. See Section 1.5.1, "Understanding OAuth Services Access Tokens" for more information about custom attributes.

Client ID - The unique ID that the authorization server created for this client during registration. (Read-only).

Client Secret - A secret value known to the OAuth authorization service and the client. The authorization service checks the client secret and the client ID when it receives token endpoint requests from the client.

HTTP Redirect URIs - The client URIs that the OAuth server is allowed to redirect the user-agent to once access is granted or denied.

Privileges

Bypass User Consent - If selected, the client will not ask for the user's explicit authorization to access the user's protected resources. If this option is selected, this setting overrides the resource server setting. Clear this option if the client should be subject to the resource server setting.

Allow Access to all Scopes - If selected, the client can obtain an access token regardless of scope limitations for any resource server in the identity domain. Clear this option if the client should be subject to scope limitations.

Allowed Scopes - Lists the range of access the client has to the requested resources. To grant additional access, click **Add** to add a row to the table, then choose from the drop-down menu the scope to be added. To restrict access, select the scope that you want to remove by clicking the table row, then click **Delete** to remove the highlighted row. Click **OK** at the prompt to confirm that you want to remove the selected scope.

Grant Types - The OAuth 2.0 specification provides several authorization grant types for different security use-cases. Before obtaining an access token, the client must obtain an authorization grant that it can exchange with the OAuth service for an access token. Client privileges determine which clients are allowed which grant types. The following grant types are supported in OAuth Services:

- **Authorization Code** - This grant type is required for 3-legged flows. The resource owner logs in using the authorization server. The token endpoint exchanges the authorization code along with client credentials for an access token.

- **Resource Owner Credentials** - This grant type is used for 2-legged flows. The resource owner provides the client with his or her user name and password. This is only suitable for highly trusted client applications because the client could abuse the password, or the password could unintentionally be disclosed to an attacker. Per the OAuth 2.0 specification, the authorization server and client should minimize use of this grant type and utilize other grant types whenever possible.
- **Client Credentials** - This grant type is used for 2-legged flows. The client requests an access token using only its client credentials (or another supported means of authentication). This is suitable if the client is requesting access to protected resources under its control, or those of another resource owner when previously arranged with the authorization server.

In addition to the grant types defined in the OAuth 2.0 standard, the following options are also available:

- **Refresh Token** - Select this option to return a refresh token together with an access token in the token response. See Section 1.5, "Understanding OAuth Services Tokens" for more information.
- **JWT Bearer** - Allows a JWT assertion to be used to request an OAuth access token.
- **SAML 2 Bearer** - Allows a SAML2 assertion to be used to request an OAuth access token.
- **OAM Credentials** - Used to request OAM tokens, such as a master token, an access token, or an OAuth access token.

Attributes

Add or delete custom attributes that the authorization server returns to the client along with the scope settings.

Avoid using the same name when adding custom attributes to the service profile configuration and the scope configuration. If you define the same attribute name in both locations, the scope-based attribute value takes precedence.

Table 2–2 Web Client Attributes Names and Values

Name	Value	Notes
`jwt.audience`	Space separated values.	Used when the OAuth server generates a client assertion and a user assertion. The `aud` claim for those JWT tokens contain the defined values in this token.

2.3.3.4 Understanding the Public Clients Configuration Page

This section describes the form fields on the Web Client Configuration page when viewing an existing Web client or creating a new one. The *Mobile OAuth* Client Configuration page is described in the next section.

Identity Domain - The name of the identity domain in which this OAuth Web client is registered. (Read-only)

Name - The name of this OAuth client.

Description - (Optional) A short description to help you or another administrator identify this OAuth Web client in the future.

Allow Token Attributes Retrieval - Select this option to allow custom attributes (both attribute names and values) to be shared with resource servers and the resource owner. See Section 1.5.1, "Understanding OAuth Services Access Tokens" for more

information about custom attributes.

Client ID - The unique ID that the authorization server created for this client during registration. (Read-only).

HTTP Redirect URIs - The client URIs that the OAuth server is allowed to redirect the user-agent to once access is granted or denied.

Privileges

Bypass User Consent - If selected, the client will not ask for the user's explicit authorization to access the user's protected resources. If this option is selected, this setting overrides the resource server setting. Clear this option if the client should be subject to the resource server setting.

Allow Access to all Scopes - If selected, the client can obtain an access token regardless of scope limitations for any resource server in the identity domain. Clear this option if the client should be subject to scope limitations.

Allowed Scopes - Lists the range of access the client has to the requested resources. To grant additional access, click **Add** to add a row to the table, then choose from the drop-down menu the scope to be added. To restrict access, select the scope that you want to remove by clicking the table row, then click **Delete** to remove the highlighted row. Click **OK** at the prompt to confirm that you want to remove the selected scope.

Grant Types - The OAuth 2.0 specification provides several authorization grant types for different security use-cases. Before obtaining an access token, the client must obtain an authorization grant that it can exchange with the OAuth service for an access token. Client privileges determine which clients are allowed which grant types. The following grant types are supported in OAuth Services:

- **Authorization Code** - This grant type is required for 3-legged flows. The resource owner logs in using the authorization server. The token endpoint exchanges the authorization code along with client credentials for an access token.
- **Implicit** - This grant type is used for 2-legged flows. The resource owner provides the client with his or her user name and password. This is only suitable for highly trusted client applications because the client could abuse the password, or the password could unintentionally be disclosed to an attacker. Per the OAuth 2.0 specification, the authorization server and client should minimize use of this grant type and utilize other grant types whenever possible.

Attributes

Add or delete custom attributes that the authorization server returns to the client along with the scope settings.

Avoid using the same name when adding custom attributes to the service profile configuration and the scope configuration. If you define the same attribute name in both locations, the scope-based attribute value takes precedence.

2.3.3.5 Understanding the Mobile Clients Configuration Page

This section describes the form fields on the Mobile Client Configuration page when viewing an existing Mobile client or creating a new one. The OAuth *Web Client* Configuration page is described in the previous section.

Identity Domain - The name of the identity domain in which this OAuth mobile client is registered. (Read-only)

Name - The name of this OAuth client.

Description - (Optional) A short description to help you or another administrator identify this OAuth mobile client in the future.

Allow Token Attributes Retrieval - Select this option to allow custom attributes (both attribute names and values) to be shared with resource servers and the resource owner. See Section 1.5.1, "Understanding OAuth Services Access Tokens" for more information about custom attributes.

Client ID - The unique ID that the authorization server created for this client during registration. (Read-only).

Jailbreaking Detection - Select to enable jail breaking detection for mobile devices. See Section 1.4.8, "Understanding Jailbreak Detection Policy" for more information.

Mobile Redirect URIs - The client URIs that the OAuth server is allowed to redirect the user-agent to once access is granted or denied.

Privileges

Bypass User Consent - If selected, the client will not ask for the user's explicit authorization to access the user's protected resources. If this option is selected, this setting overrides the resource server setting. Clear this option if the client should be subject to the resource server setting.

Allow Access to all Scopes - If selected, the client can obtain an access token regardless of scope limitations for any resource server in the identity domain. Clear this option if the client should be subject to scope limitations.

Allowed Scopes - Lists the range of access the client has to the requested resources. To grant additional access, click **Add** to add a row to the table, then choose from the drop-down menu the scope to be added. To restrict access, select the scope that you want to remove by clicking the table row, then click **Delete** to remove the highlighted row. Click **OK** at the prompt to confirm that you want to remove the selected scope.

Grant Types - The OAuth 2.0 specification provides several authorization grant types for different security use-cases. Before obtaining an access token, the client must obtain an authorization grant that it can exchange with OAuth Services for an access token. Client privileges determine which clients are allowed which grant types. The following grant types are supported in OAuth Services:

- **Authorization Code** - This grant type is required for 3-legged flows. The resource owner logs in using the authorization server. The token endpoint exchanges the authorization code along with client credentials for an access token.
- **Resource Owner Credentials** - This grant type is used for 2-legged flows. The resource owner provides the client with his or her user name and password. This is only suitable for highly trusted client applications because the client could abuse the password, or the password could unintentionally be disclosed to an attacker. Per the OAuth 2.0 specification, the authorization server and client should minimize use of this grant type and utilize other grant types whenever possible.
- **Client Credentials** - This grant type is used for 2-legged flows. The client requests an access token using only its client credentials (or another supported means of authentication). This is suitable if the client is requesting access to protected resources under its control, or those of another resource owner when previously arranged with the authorization server.
- **Refresh Token** - Select this option to return a refresh token together with an access token in the token response. See Section 1.5, "Understanding OAuth Services Tokens" for more information.
- **JWT Bearer** - Allows a JWT assertion to be used to request an OAuth access token.

- **SAML 2 Bearer** - Allows a SAML2 assertion to be used to request an OAuth access token.
- **OAM Credentials** - Used to request OAM tokens, such as a master token, an access token, or an OAuth access token.
- **Client Verification Code** - Used by mobile clients to request a pre-verification code from OAuth server, which subsequently gets used mobile client flows.

Apple Push Notification

Applies to iOS devices only. The OAuth authorization server can restrict token delivery to a specific app installed on a specific mobile device by sending part of the client registration handle through HTTPS, and sending the other part through push notification using the Apple Push Notification Service (APNS). Use the following fields to configure how the OAuth server connects to APNS for this specific client app.

Connection Settings - Select **Enabled** to send a portion of security codes and tokens to the mobile client app using APNS. (The portions not sent using APNS are sent using HTTPS.) Clear this option if you do not want to use APNS for this mobile client app.

Minimum Connection Pool Size - Specifies the minimum number of connections in the connection pool.

Maximum Connection Pool Size - Specifies the maximum number of connections in the connection pool.

Keep Alive - The Apple Push Notification keep alive value in seconds.

Certificate for APNS Communication Setup - Choose **Development** to use the Apple development environment for initial development and testing of the application; choose **Production** to use Apple's production environment.

SSL/TLS Certificate for Development - Click **Browse** to navigate to the development SSL/TLS certificate issued by Apple for the Apple Push Notification Service.

Development Certificate Password - Type the development password for the Apple Push Notification certificate.

SSL/TLS Certificate for Production - Click **Browse** to navigate to the production SSL/TLS certificate issued by Apple for the Apple Push Notification Service.

Production Certificate Password - Type the production password for the Apple Push Notification certificate.

Google Application Settings

Applies to Android devices only. The OAuth authorization server can restrict token delivery to a specific app installed on a specific mobile device by sending part of the client registration handle through HTTPS, and sending the other part through push notification using Google Cloud Messaging (GCM) for Android. Use the following fields to configure how the OAuth server connects to the GCM service for this specific client app.

Restricted Package Name - The Google restricted package name.

Mobile Service Settings

Override the default settings - By enabling Override the default settings in a Mobile Client profile, an administrator can set the security level and enable server-side single sign on at the client level. When set, these client settings over ride same settings at the OAuth Services Service Profile mobile configuration setting. This can be used if a

particular client in an identity domain needs a behavior that is different from what is defined in the OAuth Services Service Profile.

Configuration Settings

Device Claim Attributes - Specifies the device attributes that the system should collect for device fingerprinting. If empty, the system collects every attribute in the SDK.

Mobile Custom Attributes - Specifies key-value pairs that should be sent to mobile applications using app profiles. (Mobile applications request app profiles that contain server-side settings, including endpoints, jail break detection policies, and security level details.

Attributes

Add or delete custom attributes that the authorization server returns to the client along with the scope settings.

Avoid using the same name when adding custom attributes to the service profile configuration and the scope configuration. If you define the same attribute name in both locations, the scope-based attribute value takes precedence.

2.3.4 Configuring the Service Provider

See Section 1.4.4, "Understanding Service Providers" for introductory information about Service Providers. The following section describes how to use the user interface to configure a Service Provider. It includes the following topics:

- Editing or Deleting the Service Provider
- Understanding the Service Provider Configuration Page

> **Note:** Only one Service Provider can be configured at a time.

2.3.4.1 Editing or Deleting the Service Provider

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it for editing.
2. Select the **Service Providers** tab.
3. Do the following:
   - To edit a service provider, click its name in the table.
   - To delete a service provider, select it by clicking the box to the left of the name and then click the delete button in the command bar.

2.3.4.2 Understanding the Service Provider Configuration Page

This section describes the form fields on the Service Provider Configuration page.

Identity Domain - The name of the identity domain with which this Service Provider is registered. (Read-only)

Name - The name of this service provider.

Description - (Optional) A short description to help you or another administrator identify this service provider.

Service Provider Java Class - The Java class that implements this service provider.

Attributes

Use the attribute settings in Table 2–3 to configure the Service Provider connection with Access Manager.

Table 2–3 OAuth Service Provider Attributes for Access Manager

Name	Value	Notes
oam.OAM_VERSION	OAM_11G	Either **OAM_11G** or **OAM_10G**, depending on the Oracle Access Manager version in use.
oam.Webgate_ID	accessgate-oic	
oam.ENCRYPTED_PASSWORD		
oam.DEBUG_VALUE	0	
oam.TRANSPORT_SECURITY	OPEN	Specify the method for encrypting messages between this AccessGate and the Access Servers. The encryption methods need to match. Valid values include: ■ **OPEN** ■ **SIMPLE** ■ **CERT** To update these settings, see Section 49.9.1.1, "Configuring Mobile and Social Services to Work With Access Manager in Simple and Certificate Mode."
oam.OAM_SERVER_1	localhost:5575	Specify the host name and port number of the primary Oracle Access Management server.
oam.OAM_SERVER_1_MAX_CONN	4	Specify the maximum number of connections that this Mobile and Social instance can establish with OAM_SERVER_1. The default value is 4.
oam.OAM_SERVER_2	oam_server_2:5575	Specify the host name and port number of the secondary Oracle Access Management server.
oam.OAM_SERVER_2_MAX_CONN	4	Specify the maximum number of connections that this Mobile and Social instance can establish with OAM_SERVER_2. The default value is 4.
oam.AuthNURLForUID	wl_authen://sample_ldap_no_pwd_protected_res	

Table 2–3 (Cont.) OAuth Service Provider Attributes for Access Manager

Name	Value	Notes
oam.OAM_LOCAL_MODE	true	Specifies if Mobile and Social should use "local mode" or "remote mode" to communicate with the OAM server. If the attribute value is set to **false**, Mobile and Social communicates with OAM over TCP/IP. If set to **true** (or if this attribute is undefined), Mobile and Social uses a direct connection to communicate with OAM. Prior to version 11.1.2.3, Mobile and Social only communicated with OAM using TCP/IP (that is, remote mode). Now communication defaults to local, which is faster. To configure Mobile and Social to communicate with OAM 10g, set the OAM_LOCAL_MODE attribute to **false**.

2.3.5 Configuring Custom Resource Servers

See Section 1.4.5, "Understanding Resource Servers" for introductory information about Resource Servers. The following section describes how to use the user interface to configure a Resource Server. It includes the following topics:

- Creating a Custom Resource Server
- Editing or Deleting a Resource Server
- Understanding the Custom Resource Servers Configuration Page

OAuth Services provides two out-of-the-box services modeled as Resource Servers and protected with an Access Token. For configuration information on the User Profile Services and Consent Management Services Resource Servers, see Configuring User Profile Services and Configuring Consent Management Services respectively.

2.3.5.1 Creating a Custom Resource Server

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it.
2. Select the **Resource Servers** tab.
3. To define a new resource server for use with OAuth Services, click the Create button in the **Custom Resource Servers** section.

   The Custom Resource Server Configuration page opens.

2.3.5.2 Editing or Deleting a Resource Server

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it for editing.
2. Click the **Resource Servers** tab.
3. To open a configured custom Resource Server for editing, click its name in the Custom Resource Servers table.

   The Custom Resource Server Configuration page opens.

2.3.5.3 Understanding the Custom Resource Servers Configuration Page

Identity Domain - The name of the identity domain to which this resource server applies. (Read-only)

Name - The name of this resource server (or *resource service*).

Description - (Optional) A short description to help you or another administrator identify this resource server in the future.

Allow Token Attributes Retrieval - Select this option to allow custom attributes (both attribute names and values) to be shared with clients and the resource owner. See Section 1.5.1, "Understanding OAuth Services Access Tokens" for more information about custom attributes.

Authorization & Consent Service Plug-in - From the menu, choose an authorization plug-in for the resource server. This plug-in type defines security policy around interactions where authorization and user consent are granted. It can influence claims in a generated token as well. See Section 1.4.6, "Understanding Plug-Ins" for plug-in descriptions.

Audience Claim - Identifies the audiences for which the OAuth token is intended. Each principal intended to process the OAuth token must identify itself with a value in **Audience Claim**.

Resource Server ID - The unique ID created for this resource server during registration. (Once the resource server configuration is saved, this field cannot be changed.)

Scopes

Click **Add** to add a new row to the scopes table. Click to select a row, then click **Delete** to remove it.

Name - Type a scope definition. Use dot notation, for example: `photo.read`

Description - Type a short note that describes the scope.

Require User Consent - Select to require the authorization server to display a user consent form so that the user can approve (or deny) the access request.

Offline Scope - Allows client applications to request a refresh token that can be used to obtain an access token even when the user is offline or not present. Client applications use the refresh token to get a new access token to access resources. See Section 1.5, "Understanding OAuth Services Tokens" for more information.

Token Settings

Override the default settings - Select this option if the token settings defined on the resource server configuration page should override the default token settings defined on the OAuth Services profile page.

Token Name - The name of the token.

Expires - The length of time in minutes after which the token is no longer valid.

Refresh Token Expires - The length of time in minutes after which the refresh token is no longer valid.

Custom Attributes

Use this section to define custom attributes that OAuth Services embeds in the access tokens. See Section 1.5.1, "Understanding OAuth Services Access Tokens" for more information about custom attributes.

- **Static Attributes** - Attribute name and value pairs where the value is fixed at the time that you define the attribute. For example, `name1=value1`.
- **Dynamic Attributes** - User-profile specific attributes.

2.3.6 Configuring User Profile Services

See Section 1.4.5.1, "Understanding User Profile Services" for introductory information about the User Profile Services. The following section describes how to use the console to configure an instance for the User Profile Services.

- Creating a New User Profile Service
- Editing the User Profile Service
- Understanding the User Profile Services Configuration Page

2.3.6.1 Creating a New User Profile Service

1. Open the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it.
2. Click the **Resource Servers** tab.
3. Click the Create button in the **User Profile Services** section.

   The **User Profile Services Configuration** page opens.

2.3.6.2 Editing the User Profile Service

1. Open the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it for editing.
2. Click the **Resource Servers** tab.
3. In the **User Profile Services** section, click the service name to edit it.

   The **User Profile Services Configuration** page opens.

2.3.6.3 Understanding the User Profile Services Configuration Page

Use this page to configure the User Profile Service. This service supports OAuth 2.0 authorization and allows clients to interact with a back-end directory server and perform User Profile REST operations on Person, Group, and Relationship entities.

Identity Domain - The name of the identity domain to which this service profile applies. (Read-only)

Name - The name of this service profile.

Description - (Optional) A short description to help you or another administrator identify this service profile in the future.

Service Enabled - Select to enable the service, or clear the option box to disable it.

Allow Token Attributes Retrieval - Select this option to allow custom attributes (both attribute names and values) to be shared with clients. If enabled, the user consent form notifies the user that user-profile-specific details will be shared with the client. See Section 1.5.1, "Understanding OAuth Services Access Tokens" for more information about custom attributes.

Audience Claim - Identifies the audiences that the OAuth token is intended for. Each principal intended to process the OAuth token must identify itself with a value in audience claim.

Resource Server ID - The unique ID that OAuth Services created for this User Profile resource server. (Read-only)

Service Endpoint - The URI where the service receives and responds to create, read, update, and delete user profile service requests. Create a unique uniform resource identifier (URI) address for this service; for example, `localhost:5575`

Authorization & Consent Service Plug-in - From the menu, choose an authorization plug-in for the service. This plug-in type defines security policy around interactions where authorization and user consent are granted. It can influence claims in a generated token as well. See Section 1.4.6, "Understanding Plug-Ins" for plug-in descriptions.

Protected by OAuth Service Profile - From the menu, choose the OAuth service profile that protects the user profile service.

Identity Store Name - The name of the identity store that contains the user records.

Scopes

Security Protection

Configure individual permission settings for person, relationship, and group entities. Click **Add** to add a record to the table; select a row and click **Delete** to remove the record. The service uses the following default entity names:

URI - The URI segment for which the scope is defined.

- **/me** - Designates operations that apply to the user logged in to the client
- **/users** - Designates operations that apply to other users
- **/groups** - Designates operations that apply to groups
- **/secretkey** - Designates operations that apply to secret key management

Service Enabled - Select to enable the service for this scope.

Allow Read - Select to allow read operations for this scope.

Allow Write - Select to allow write operations for this scope.

Unprotected - Select this option if you do not want to limit access, or clear this option to limit access by scope.

OAuth Scope - Type a scope definition. Use dot notation, for example:
`UserProfile.me.write`

Description - Type a short note that describes the scope.

Require User Consent - Select to require the authorization server to display a user consent form so that the user can approve (or deny) the access request.

Identity Attributes of the Selected Scope - Click an entity row in the **Security Protection** table to view the Attribute table for that entity. Click **Add** to add a record to the table; select a row and click **Delete** to remove the record.

Offline Scope - Allows client applications to request a refresh token that can be used to obtain an access token even when the user is offline or not present. Client applications use the refresh token to get a new access token to access resources. See Section 1.5, "Understanding OAuth Services Tokens" for more information.

Token Settings

Override the default settings - Select this option if the token settings defined on the resource server configuration page should override the default token settings defined on the OAuth Services profile page.

Token Name - The name of the token.

Expires - The length of time in minutes after which the token is no longer valid.

Refresh Token Expires - The length of time in minutes after which the refresh token is no longer valid.

Proxy Authentication

Select Proxy Authentication to allow the identity of a user using a web application (also known as a "proxy") to be passed through the application to the database server. Oracle Unified Directory (OUD) and Active Directory (AD) support proxy authentication. The Access Control option is simply to provide Proxy Authentication support for directory servers that do not have it built in. See Section 1.4.5.1.1, "Using Proxy Authentication" for more details.

Attributes

Use this section to define user-profile specific (dynamic) attributes.

Table 2–4 User Profile Service Attributes

Name	Value
`accessControl`	`false`
`adminGroup`	`cn=Administrators,ou=groups,ou=myrealm,dc=base_domain`
`selfEdit`	`true`

Resource URIs

Use this section to enable or disable the **/me**, **/users**, **/groups**, **/secretkey** services, and define the service endpoint URIs and provider implementation class paths for these services.

Service Endpoint - The URI where the service receives and responds to service requests. Create a unique uniform resource identifier (URI) address for this service; for example, `localhost:5575`

Entities

Use the fields in this section to configure entity relationships.

- **Name** - The name of the defined entity relationship.
- **Identity Directory Service Relation** - Choose the directory service relationship that is to be accessed by the relationship **End Point** segment.
- **End Point** - Type an entity relationship URI segment that will be used to access a corresponding data column in the Identity Directory service. For example, if `memberOf` is the End Point URI, then:

  `http://<host>:<port>/.../idX/memberOf`

  would be the URI to access related entities of an entity with ID `idX`.
- **Source Entity URI** - The URI (or URL) of the source entity.
- **Destination Entity URI** - The URI (or URL) of the destination entity.

- **Scope for Requesting Recursion** - Use Scope attribute values with the scope query parameter to retrieve a nested level of attributes in a relationship search. To access related entities recursively, type the value to be used. The default configuration uses two scope attribute values: `toTop` and `all`. If the **Scope for Requesting Recursion** value is the attribute value `all`, then the following REST URI example is used to make the request:

  ```
  http://host:port/.../idX/reports?scope=all
  ```

  In this example, the URI returns the entities related to the entity with ID `idX`, as well as all further related entities.

Attributes

Use this section to define user-profile entity specific (dynamic) attributes.

2.3.7 Configuring Consent Management Services

See Section 1.4.5.2, "Understanding Consent Management Services" for introductory information about the Consent Management Services. The following section describes how to use the user interface to configure the Consent Management Services.

- Creating a New Consent Management Service
- Editing an Existing Consent Management Service
- Understanding the Consent Management Services Configuration

2.3.7.1 Creating a New Consent Management Service

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it.
2. Click the **Resource Servers** tab.
3. Click the **Create** button in the **Consent Management Services** section.

   The **Consent Management Service Configuration** page opens.

2.3.7.2 Editing an Existing Consent Management Service

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it for editing.
2. Click the **Resource Servers** tab.
3. In the **Consent Management Services** section, click the service name to edit it.

   The **Consent Management Service Configuration** page opens.

2.3.7.3 Understanding the Consent Management Services Configuration

The Consent Management Services handle consent storage, retrieval, revocation, and consent validation operations.

Identity Domain - The name of the identity domain to which this consent management service applies. (Read-only)

Name - The name of this consent management service.

Description - (Optional) A short description to help you or another administrator identify this service in the future.

Service Enabled - Select to enable the service, or clear the option box to disable it.

Allow Token Attributes Retrieval - Select this option to allow custom attributes (both attribute names and values) to be shared with clients, resource servers, and the resource owner.

Audience Claim - Identifies the audiences that the OAuth token is intended for. Each principal intended to process the OAuth token must identify itself with a value in audience claim.

Resource Server ID- The unique ID that the authorization server created for this resource server during registration. (Read-only)

Service Endpoint - The URL where the Consent Management Service receives and responds to client and resource owner service requests.

Authorization & Consent Service Plug-in - From the menu, choose an authorization plug-in for the service. This plug-in type defines security policy around interactions where authorization and user consent are granted. It can influence claims in a generated token as well. See Section 1.4.6, "Understanding Plug-Ins" for plug-in descriptions.

Protected by OAuth Service Profile - From the menu, choose the OAuth service profile that protects the consent management service.

Scopes

Security Protection

Configure individual permission settings. Click **Add** to add a record to the table; select a row and click **Delete** to remove the record. The service uses the following default entity names:

URI - The URI segment for which the scope is defined.

- **/retrieve**
- **/grant**
- **/revoke**

Allow Read - Select to allow read operations for this scope.

Allow Write - Select to allow write operations for this scope.

Unprotected - Select this option if you do not want to limit access, or clear this option to limit access by scope.

OAuth Scope - Type a scope definition. Use dot notation, for example:
`UserProfile.me.write`

Description - Type a short note that describes the scope.

Require User Consent - Select to require the authorization server to display a user consent form so that the user can approve (or deny) the access request.

Offline Scope - Allows client applications to request a refresh token that can be used to obtain an access token even when the user is offline or not present. Client applications use the refresh token to get a new access token to access resources. See Section 1.5, "Understanding OAuth Services Tokens" for more information.

Token Settings

Override the default settings - Select this option if the token settings defined on the resource server configuration page should override the default token settings defined on the OAuth service profile page.

Token Name - The name of the token.

Expires - The length of time in minutes after which the token is no longer valid.

Refresh Token Expires - The length of time in minutes after which the refresh token is no longer valid.

Attributes

Use this section to define custom attributes

Resources URIs

Use this section to enable or disable the **retrieve**, **grant**, and **revoke** services. You can also define the service endpoint URIs and provider implementation class paths for these services.

Service Endpoint - The URI where the service receives and responds to requests. Create a unique URI address for this service.

Service Enabled - Select to enable the service, or clear the option box to disable it.

Attributes

Use this section to define consent management entity-specific (dynamic) attributes.

2.3.8 Configuring Plug-Ins

Use this page to configure security plug-ins. See Section 1.4.6, "Understanding Plug-Ins" for plug-in descriptions.

2.3.8.1 Creating a new Plug-in

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it.
2. Click the **Plug-ins** tab.
3. Click the **Create** button in one of the plug-in category sections.

   The Plug-in Configuration page opens.

2.3.8.2 Understanding the Plug-in Configuration Page

Use this page to add a plug-in to an Identity Domain or edit an existing plug-in configuration. Only some of the fields listed below will apply to the plug-in you are configuring.

Identity Domain - The name of the identity domain where the plug-in is located.

Name - The name of the plug-in.

Description - (Optional) A short description to help you or another administrator identify this plug-in in the future.

Implementation Class - Choose the class from the menu that implements the plug-in interface. Applies to the Mobile Client Plug-in Configuration page, the Mobile Resource Server Plug-in Configuration page, and the Mobile Authorization & Consent Service Plug-in Configuration page. See the *Oracle Fusion Middleware Developer's Guide for Oracle Access Management* for details.

Interface Class - Lists the interface class for this plug-in. Applies to the Mobile Client Plug-in Configuration page, the Mobile Resource Server Plug-in Configuration page, and the Mobile Authorization & Consent Service Plug-in Configuration page.

Security Handler Class - Choose the Java class that defines the Security Handler Plug-in. Applies to the Mobile Adaptive Access Plug-in Configuration page and the Mobile Custom Token Attributes Plug-in Configuration page.

Mobile Security Manager Plug-in Class - Choose the Java class that defines the Mobile Security Manager Plug-in. Applies to the Mobile Security Manager Plug-in Configuration page only.

MSM Device Inventory Attributes Precedence - When enabled, if both the Mobile Security Manager (MSM) component and the Mobile OAuth server supply a value for the same device attribute, the value supplied by the Mobile Security Manager is used. If the attribute value from the MSM component is not available, the value from the Mobile OAuth server is used instead.

MSM Attributes - Lists the attributes that the Mobile Security Manager plug-in harvests from mobile devices. The first column lists the mobile device attributes that the Mobile Security Manager component collects. The second column lists the mobile device attributes that the Mobile OAuth server collects during mobile app requests. If the same device attribute is available from both the MSM component and the server, both are listed in the same row. (For example, the MSM attribute "imei" appears in column one, and the matching server attribute "oracle:idm:claims:client:imei" appears in column two of the same row.) For a list of the device attributes that the Mobile OAuth server collects during app requests, open the OAuth Mobile Client Configuration page and locate the **Device Claim Attributes** list in the **Configuration Settings** section. To add additional attributes, click **Add** to add a row at the bottom of the table, and enter the attribute name. (Enter attributes sourced from Mobile Security Manager in the first column, and attributes from the Mobile OAuth server in the second column. Enter attributes one per row unless the attributes are equivalent and should be mapped to one another.)

Attributes - Use this section to define custom plug-in attributes.

2.3.9 Configuring Server Settings

Use the Server Settings Configuration page to configure general server settings for the identity domain named.

> **Note:** See Section 48.1.2, "Deploying Mobile and Social" for information about deploying Mobile and Social with a WebGate.

Identity Domain - The name of the identity domain to which the settings on this configuration page apply. (Read-only)

HTTP Proxy Settings

Configure the following settings if a proxy server is in place between the OAuth Token Service (the Push Service) and the Apple Push Notification Service (APNS) or Google Cloud Messaging (GCM) service.

Proxy URL - Choose the protocol to use to connect to the proxy server (HTTP or HTTPS), then type the proxy server host name and port number.

Proxy Authentication - Type the user name and password required to authenticate with the proxy server.

Apple Push Notification

Configure the default values that should be used for this identity domain. Use the OAuth Mobile Client Configuration page to customize these settings on an app by app basis.

Minimum Connection Pool Size - Specifies the minimum number of connections in the connection pool.

Maximum Connection Pool Size - Specifies the maximum number of connections in the connection pool.

Keep Alive - The Apple Push Notification keep alive value in seconds.

Token Life Cycle Management

Maximum Search Results - Specify the maximum number of token entry search results that should be returned on the Token Life Cycle Management page.

Attributes

Attributes - Use this section to define custom attributes.

Table 2–5 OAuth Server Settings Attributes

Name	Value	Notes
wgAuthnUserHeader	OAM_REMOTE_USER	This attribute usage is optional. If an OAM Webgate is front ending/proxying requests to an OAuth server, set this attribute. The OAM Webgate sets the OAM_REMOTE_USER header to identify the authenticated user. If a deployment uses another header name instead of OAM_REMOTE_USER, then this attribute needs to be set with that header name.

2.3.10 Configuring the Jailbreak Detection Policy

See Section 1.4.8, "Understanding Jailbreak Detection Policy" for introductory information about the jail breaking detection policy. The following section describes how to use the user interface to configure the policy.

Jailbreak Detection - Select **Enabled** to turn the Jailbreaking Detection Policy on, or clear this option to turn it off for all client application instances. If you enable the Jailbreaking Detection Policy here, you can disable it on an application by application basis. If you disable the Policy here, you cannot enable or disable the feature on an application by application basis.

Policy Statements

Use the buttons in the menu to add, delete, and re-order policy statements.

Order - The sequential row number assigned to each row in the table.

Enabled - Select this option to activate the policy statement condition.

Minimum OS Version - The minimum iOS version to which the policy applies. If the value is 1.0, the policy will apply to iOS devices running at least version 1.0 of iOS.

Maximum OS Version - The maximum iOS version to which the policy applies. If the value is empty, a maximum iOS version number is not checked so the policy applies to

any iOS version higher than the value specified for Min OS Version. Once set you cannot remove the value and leave this field empty.

Minimum Client SDK Version - The minimum Mobile and Social Client SDK version number. For example, 11.1.2.0.0.

Maximum Client SDK Version - The maximum Mobile and Social Client SDK version number. For example, 11.1.2.3.0.

Details - Additional details about the Jailbreak Detection Policy policy statement. Hover the mouse over the info icon to view the details in a pop-up.

Policy Statement Conditions

Click to select a row in the table to view or edit its values in this section. See the previous section (Policy Statements) for field descriptions.

Policy Statement Detection Logic

Policy Expiration Duration - Type the length of time in seconds that the SDK on the mobile client device should wait before expiring the local copy of the policy and retrieving a newer version.

Auto Check Period - Type the interval of time in minutes that the client device should wait before executing the Jailbreaking Detection Policy statements again.

Detection Location - The iOS client device uses a logical-OR operator to evaluate Policy statements. Add a Detection Location as follows:

- **File Path** - Type the absolute path to the file or directory on the device for which the Detection Policy should search.
- **Action** - Select **Exists** which instructs the Detection Policy to evaluate whether it can access a file path.
- **Success** - Select if the Policy should flag the device as jail broken if the specified files or directories are found on the device. Use this option if the policy is checking for unauthorized files or directories. Clear this option if the Policy should flag the device as jail broken if the specified files or directories are *not* found. (Use this option if checking for *required* files or directories.)

2.3.11 Configuring Token Life Cycle Management

Use this screen to search for and revoke tokens that have been issued. You can search for tokens using criteria such as user ID, client ID/name, client IP address, service profile, assertion token category, and token creation/expiration time. Enter your criteria and click **Search**. The maximum number of token entry search results returned is determined by the **Maximum Search Results** setting on the OAuth Server Settings page.

Search Criteria

Identity Domain - The name of the identity domain that you are searching for tokens. (Read only)

User - Specify an LDAP UID (`john.smith`) or an LDAP Fully Qualified DN (`cn=jane.smith,dc=example,dc=com`) to search by.

Client - Specify a client ID to search for tokens by.

Client IP Address - Specify a client IP address (for example, `192.168.100.1`) to search for tokens by

Service Profile - Choose a profile from the menu, or leave this selection empty.

Assertion Token Category - Choose a category from the menu, or leave this selection empty.

Token Issued - Search for tokens by the date and time that they were issued.

Token Expiring at - Search for tokens by the date and time that they expire.

Mobile Device Claim Attributes

IMEI - Specify the unique 15-digit IMEI (International Mobile Equipment Identity) code to search by. The IMEI can be displayed on most mobile handsets by dialing ***#06#**.

MAC Address - Specify the unique MAC (Media Access Control) address to search by.

Phone Number - Specify a phone number to search by.

2.4 Configuring OAuth Services for Third-Party JWT Bearer Assertions

OAuth Services accepts third-party (non-Oracle) JWT assertions. You must, however, configure a trust relationship by adding the third-party's certificate into the OAuth Services Service Profile keystore. OAuth Services uses the keystore to verify the JWT assertion's digital signature. Create a separate keystore for each Service Profile that needs its own signing certificate. This section covers the following topics:

- Understanding the Default Service Profile Keystore
- Creating a Non-Default Keystore for a Service Profile
- Configuring a Third-Party JWT Trust Issuer

2.4.1 Understanding the Default Service Profile Keystore

The role of the Service Profile is described in Section 1.4.2, "Understanding Service Profiles" The default Service Profile (OAuthServiceProfile) created in the DefaultDomain uses the Java Keystore (JKS) included with Oracle Access Management. It consists of the following files.

Table 2–6 Default OAuth JKS Keystore File and Settings File

File	Path
JKS keystore file	*$DOMAIN_HOME*/config/fmwconfig/default-keystore.jks
Keystore settings file	*$DOMAIN_HOME*/config/fmwconfig/jps-config.xml

> **Note:** Oracle Web Services Manager also uses the `default-keystore.jks` service. For details see Section 44.2.2, "About the Oracle Web Services Manager Keystore (default-keystore.jks)."

You can use the following Java `keytool` command to list all of the private key and certificate information in the default keystore (`default-keystore.jks`).

```
keytool -list -keystore default-keystore.jks
```

Note: When a new key is added to the OAM keystore, the OAM server needs to be restarted since keystore changes are not automatically refreshed.

Use the following procedure to find the keystore credential with Oracle Enterprise Manager Fusion Middleware Control Console.

1. Login to the Oracle Enterprise Manager Fusion Middleware Control Console.

   `http://host_name.domain_name:port_number/em`

2. Navigate through WebLogic Domain --> <domain name>.

   <domain name> is the name of the domain in which the information is stored.

3. Click on the WeblogicDomain found in the right corner.
4. Select System MBean Browser ---> com.oracle.jps ---> Server:oam_server1 ---> JPS Credential Store.
5. Navigate through JPS Credential Store Mbean --> Operations --> getPortableCredentialMap.
6. Enter the p1 parameter as oracle.wsm.security.
7. Click Invoke.
8. Expand Data Element2.

   The password value is displayed.

2.4.2 Creating a Non-Default Keystore for a Service Profile

The steps in this section describe how to:

- Create a separate keystore to store the third-party's certificates
- Import the certificates into the keystore
- Configure the keystore
- Create a CSF Entry for the keystore service instance
- Add the keystore service to the appropriate Service Profile

Note: Any changes made during this procedure require a restart of the OAM server.

Create the Keystore

Create a new Java Keystore (JKS) using the `keytool` utility that is distributed with the Java JDK.

1. Go to *`$JDK_HOME`*`/jdk/bin` and open a prompt.
2. Using `keytool`, generate a key pair:

   `keytool -genkeypair -keyalg RSA -dname` *`dname`*

   `-alias` *`aliasname`* `-keypass` *`key_password`* `-keystore` *`keystore`*

   `-storepass` *`keystore_password`* `-validity` *`days_valid`*

   Where:

- *dname* is the X.500 Distinguished Name to be associated with `alias`, and is used as the issuer and subject fields in the self-signed certificate. This can be any string as long as it's in the correct format (for example, `cn=spaces,dc=example,dc=com`).
- *aliasname* is a short name that identifies the new keystore entry
- *key_password* is the password for the new public key
- *keystore* is the keystore name, (for example, `oauth-xyz-keystore.jks`)
- *keystore_password* is the keystore password
- *days_valid* is the number of days for which the certificate should be considered valid (for example, 1064).

Example 2–1 Creating the Keystore

```
keytool -genkeypair -keyalg RSA -dname "cn=spaces,dc=example,dc=com"
-alias oauthkey -keypass password123 -keystore oauth-xyz-keystore.jks
-storepass passwordxyz -validity 1064
```

Load the Certificates Into the Keystore

Use the `keytool` utility to import the certificates into the keystore.

1. Using `keytool`, type the following command:

   `keytool -importcert -alias` *aliasname* `-file` *certfile*

   `-keystore` *keystore* `-storepass` *keystore_password*

   Where:

   - *aliasname* is a short name that identifies the keystore
   - *certfile* is the file containing the certificates to load
   - *keystore* is the keystore name, (for example, `oauth-xyz-keystore.jks`)
   - *keystore_password* is the keystore password

Example 2–2 Loading the Certificates

```
keytool -importcert -alias oauthkey_123 -file samplekey.cer -keystore
oauth-xyz-keystore.jks -storepass passwordxyz
```

Add the Keystore Instance to jps-config.xml

Configure the keystore service and update the credential store so that OAM can read the keystore and keys correctly. In the `jps-config.xml` keystore settings file, add the following new keystore service instance in the `<serviceInstances>` element.

1. In a text editor, open the keystore settings file:

   $DOMAIN_HOME`/config/fmwconfig/jps-config.xml`

2. Find the `<serviceInstances>` node for the keystore.provider `Provider`, and add the following:

```
<serviceInstance name="<service-instance-name>" provider="keystore.provider"
location="<keystore-location>">
  <property name="keystore.provider.type" value="db"/>
  <property name="keystore.sig.csf.key" value="sign-csf-key"/>
  <property name="keystore.enc.csf.key" value="enc-csf-key"/>
```

```
    <property name="keystore.csf.map" value="oracle.oauth.security"/>
    <property name="keystore.pass.csf.key" value="keystore-csf-key"/>
    <property name="keystore.type" value="JKS"/>
    <propertySetRef ref="props.db.1"/>
  </serviceInstance>
```

Where:

- *service-instance-name* = Any service-instance-name
- *keystore-location* = Path to the keystore file

Example 2–3 Update jps-config.xml

```
<serviceInstance name="oauth-xyz-keystore.db" provider="keystore.provider"
location="./oauth-xyz-keystore.jks">
  <property name="keystore.provider.type" value="db"/>
  <property name="keystore.sig.csf.key" value="sign-csf-key"/>
  <property name="keystore.enc.csf.key" value="enc-csf-key"/>
  <property name="keystore.csf.map" value="oracle.oauth.security"/>
  <property name="keystore.pass.csf.key" value="keystore-csf-key"/>
  <property name="keystore.type" value="JKS"/>
  <propertySetRef ref="props.db.1"/>
</serviceInstance>
```

3. Find the <jpsContexts> node and add the new service instance into the default context section. (Do not remove any pre-existing service instances.) The following example shows the addition of a <serviceInstanceRef> element with a ref to the oauth-xyz-keystore.db service instance (defined in the previous step).

Example 2–4 Adding the new Service Instance

```
<jpsContexts default="default">
  <jpsContext name="default">
     <serviceInstanceRef ref="oauth-xyz-keystore.db"/>
     ... other serviceInstanceRef elements ...
  </jpsContext>
```

Create a CSF Entry for the Keystore Service Instance

Use the following WLST commands to create the necessary Credential Store Framework (CSF) entries. Restart the server when you are done.

```
createCred(map="oracle.wsm.security", key="sign_csf_key", user="alias_
name", password=keystore_password, desc="Description of the signing key
credential")
```

```
createCred(map="oracle.wsm.security", key="enc_csf_key", user="alias_
name", password=keystore_password, desc="Description of the encryption key
credential")
```

```
createCred(map="oracle.wsm.security", key="keystore_csf_key",
user="oauth", password=keystore_password,desc="Description of the keystore
credential")
```

Where:

- *sign_csf_key* = the password for the signing key
- *alias_name* = the alias name for the key
- *keystore_password* = the keystore password

- *enc_csf_key* = the password for the encryption key
- *keystore_csf_key* = the password for the keystore

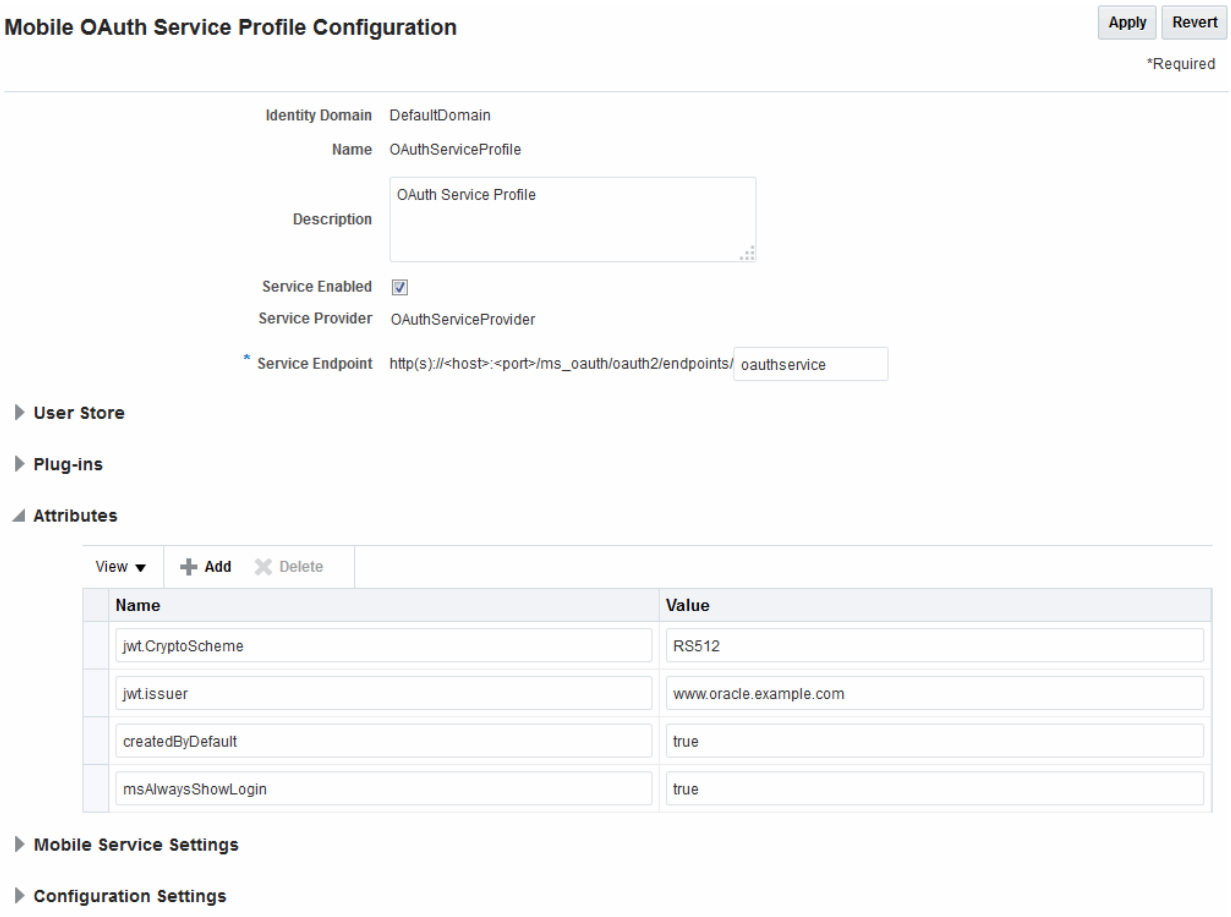
Example 2–5 Creating Credential Store Entries

```
createCred(map="oracle.wsm.security", key="oauth-sign-csf-key",
user="ms-oauth-key", password=passwordxyz, desc="Signing key credential")

createCred(map="oracle.wsm.security", key="oauth-enc-csf-key",
user="ms-oauth-key", password=passwordxyz, desc="Encryption key credential")

createCred(map="oracle.wsm.security", key="keystore_csf_key", user="oauth",
password=passwordxyz, desc="Keystore credential")
```

Add the Provider Service Name to the Service Profile

Apply the updated configuration to the Service Profile. See Section 2.3.2.1, "Creating a Service Profile" if you have not yet created an OAuth Service Profile for the third-party service.

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it for editing.
2. Click the **Service Profiles** tab.
3. Click the Service Profile name in the table.
4. Expand the **Attributes** section.
5. Add the keystore service information you configured in Example 2–4, "Adding the new Service Instance" to the **Attributes** table using the `keystore.service` name. Refer to the following screen capture.



Add new attribute named "keystore.service" to Service Profile and set value to service instance previously defined

****

2.4.3 Configuring a Third-Party JWT Trust Issuer

This section describes how to configure OAuth Services to support specific JSON Web Token (JWT) issuers. All Trusted Issuers must be defined so the OAM Server can validate the token, the thumbprint (x5t) and the key identifier based on this configuration. If there is a trusted entry already available with the same claim, header values and alias name, that entry will be used.

1. Access the Identity Domains page as described in Section 2.2, "Configuring OAuth Services Components in an Identity Domain" and click the identity domain to open it for editing.
2. Click the **Service Profiles** tab.
3. Click the Service Profile name in the table.

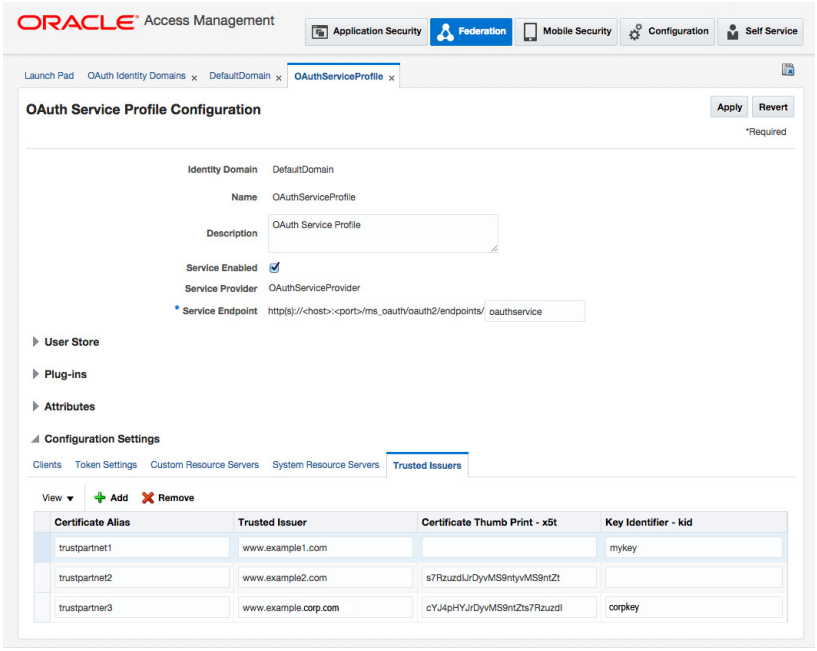
   In this example, OAuthServiceProfile.
4. Expand the **Configuration Settings** section and click the **Trusted Issuers** tab.
5. Configure the following and click Apply.
   - The certificate alias

- The URL of the Trusted Issuer
- The key identifier (KID) of the certificate alias.
- The X.509 Certificate Thumbprint

Figure 2–4 is a screenshot of the OAuth Services Service Profile Configuration Page where Trusted Issuers are defined under the Configuration Settings heading. Three Trusted Issuers are configured in it and the following is true of these configurations.

- If a request comes with an assertion in which the Trusted Issuer is "www.example1.com", the kid header is "mykey" and there is no x5t header, the runtime uses the certificate with the "trustpartner1" alias name.
- If a request comes with an assertion in which the Trusted Issuer is "www.example2.com", the value of the x5t header is "s7RzuzdlJrDyvMS9ntyvMS9ntZt" and there is no kid header, the runtime uses the certificate with the "trustpartner2" alias name.
- If a request comes with an assertion in which the Trusted Issuer is "www.example.corp.com", the value of the kid header is "corpkey" and the value of the x5t header is "cYJ4pHYJrDYvMS9ntZts7Rzuzdl", the runtime uses the certificate with the "trustpartner3" alias name.

Figure 2–4 OAuth Services Service Profile Configuration Page



OAuth Services Service Profile Configuration Page screenshot

****

2.5 Configuring a WebGate to Protect OAuth Services

This section describes how to configure a WebGate for use with OAuth Services. The WebGate protects the OAuth Services consent page and serves as a proxy so that client authorization and token endpoint requests access the WebGate rather than the Oracle

Access Management server directly. WebGates cannot be used to protect OAuth Services Resource Servers. These steps are for WebLogic environments only.

> **Note:** A WebGate proxy is required to use the 3-legged authorization scenario with an external LDAP directory server.

1. Install the Oracle HTTP Server 11g Webgate for OAM using the instructions in *Installing Webgates for Oracle Access Manager*.
2. Configure the WebGate by defining the following resource and creating an authentication policy and authorization policy.
   a. In the Oracle Access Management console, click **Application Security** at the top of the window.
   b. Under **Access Manager**, click **Application Domains**, then click **Search** to view the Application Domains on the Search Application Domains page.
   c. Click the target domain to open it for editing.
   d. Select the **Resources** tab.
   e. Create the following resource. If you are using the existing IAMSuiteAgent Host Identifier, the resource is already present and can be searched on using the **Resource URL** field.

      ```
      /ms_oauth/oauth2/ui/**
      ```

      Click to select the resource, then click the **Edit** button.
   f. Under the **Protection** heading, choose the following options from the menus and click **Apply**:

      Protection Level - Protected

      Authentication Policy - Protected HigherLevel Policy

      Authorization Policy - Protected Resource Policy

      These settings allow the Webgate to perform user authentication and user authorization.
   g. Add the following resources and set the **Protection Level** to **Excluded**:

      ```
      /ms_oauth/oauth2/endpoints/**
      /ms_oauth/oauth2/oammsui/**
      /ms_oauth/style/**
      /ms_oauth/img/**
      /oam/**
      ```

      The Webgate does not protect Excluded resources and allows them to be accessed.
3. Add the following lines to the `mod_wl_ohs.conf` file and restart the Webgate. For `WebLogicPort`, be sure to add the managed port details for your environment.

   ```
   # the following directive proxies all the OAuth requests
   <IfModule weblogic_module>
         WebLogicHost host123.us.example.com
         WebLogicPort 17100
         Debug ON
         WLLogFile /tmp/weblogic.log
         MatchExpression /ms_oauth/*
   ```

```
</IfModule>
# the following directive proxies all the OAM managed server requests.

<IfModule weblogic_module>
      WebLogicHost host123.us.example.com
      WebLogicPort 17100
      Debug ON
      WLLogFile /tmp/weblogic.log
      MatchExpression /oam/*
</IfModule>
```

4. Update the Access Manager Load Balancing settings as follows:
   a. In the Oracle Access Management console, click **Configuration** at the top of the window.
   b. Select **Access Manager** from the **View** menu in the **Settings** section.
   c. In the **Load Balancing** section, change the **OAM Server Host** and the **OAM Server Port** settings to the Webgate's host and port settings.
   d. Click **Apply**.

2.6 Configuring OAM Session Synchronization

The OAM User session synchronization feature prevents multiple OAM sessions from being created by a mobile user. The initial OAM session is created during the 3-legged Mobile scenario when the authorization code is created (provided that the OAuth consent UI pages are protected by OAM). This session is stored in the device keystore and used for subsequent OAM token requests for as long as the session is valid.

A one-time Authorization Policy change in Oracle Access Management is required for OAM session synchronization to work. The following steps configure OAM to send Session ID values to OAuth Services. Once configured, OAM session synchronization will always be used for mobile authorization requests when using OAM protection (as opposed to Mobile and Social protection) for the authorization endpoint.

> **Note:** OAM Session Synchronization requires a WebGate protecting the OAuth Services consent UI pages. See Section 2.5, "Configuring a WebGate to Protect OAuth Services" for details.

1. In the Oracle Access Management console, click **Application Security** at the top of the window.
2. Under **Access Manager**, click **Application Domains**.
3. Under **Search Application Domains**, enter the name of the target WebGate domain (or enter a partial name and wild card, *, or leave the field blank to retrieve all domains). For example:

   DesiredDomain

4. Click **Search**.
5. In the **Search Results** section, highlight the WebGate domain and click **Edit**.
6. Click the **Authorization Policies** tab.
7. In the policies table, click **Protected Resource Policy** to open it for editing.
8. Click the **Responses** tab.

9. Click **Add**.
10. Enter the following values in the **Add Response** dialog:
    - **Type** - choose **Header** from the menu.
    - **Name** - Enter any name, for example: *mysession*.
    - **Value** - Enter: `${session.id}`

    Click **Add**.
11. Click **Apply**.

2.7 Configuring Mobile OAuth for SSO Servlet Authentication

The Mobile OAuth SSO Servlet makes it possible to use a device-native app as a single sign-on proxy app when a user signs on to an app running in a mobile browser (external or embedded) in a 2-legged flow. In this arrangement, the native app implements the login page and uses Mobile and Social Services to authenticate with Oracle Access Management.

You can configure this authentication scheme to try multiple native apps on the device. If the first app does not respond within a half second (500ms) the servlet redirects the browser to the next app in the order that the apps are listed. If the servlet gets to the end of the list and there is not an application installed with the specified client ID, the servlet re-directs the user to the OAM login page. Similarly, if the request is received from a desktop browser, the request is forwarded to the OAM login page. Following is a description of the Mobile SSO Servlet authentication flow:

1. The user opens a URL in a mobile browser.
2. The web server hosting the application redirects the browser to OAM.
3. Access Manager redirects the browser to the native app on the device.
4. The browser launches the native app in response to the redirect.
5. The native app displays the user login page.
6. The user enters a user name and password.
7. The native app does a 2-legged device registration flow. It collects user credentials, does device registration, and gets the client token. If server-side single sign-on is enabled, the user token is stored on the server and is not returned to the client. If server-side single sign-on is turned off, the user token is returned to the client.
8. The native app exchanges the client token and user token for the OAM master token (OAM_ID). (If server-side single sign-on is enabled, the user token is exchanged from the server-side keystore.)
9. The native app directs the browser to the Oracle Access Management Mobile and Social server, which injects the OAM master token as a cookie.
10. The native app sends the mobile browser a URL redirect and OAM master token as a cookie.
11. The mobile browser opens the original URL now that the access request includes an OAM master token.
12. The web server sends the requested pages to the mobile browser.

The following sections contain the configuration steps.

- Configuring OAM and Your App to use the Mobile SSO Servlet

- Configuring the MobileSSOServlet Authentication Scheme

For information about Mobile OAuth SSO options, see Section 1.8, "Understanding Mobile OAuth Services Server-Side Single Sign-on."

2.7.1 Configuring OAM and Your App to use the Mobile SSO Servlet

To use the Mobile OAuth Services single sign-on authentication scheme, complete the following configuration tasks:

- In the Oracle Access Management console, protect the Web resource(s) with an OAM WebGate. To learn how, see Chapter 15, "Registering and Managing OAM 11g Agents."
- In Access Manager, create a custom MobileSSOServlet authentication scheme on the OAM server and configure it with a list of mobile application IDs. Then configure the authentication scheme in OAM to protect the Web resource(s). To learn how, see Section 2.7.2, "Configuring the MobileSSOServlet Authentication Scheme."
- If necessary, register the native mobile app client with Mobile OAuth Services. To register the native mobile app client with OAuth Services see Section 2.3.3, "Configuring Clients."
- In Mobile OAuth Services, assign the **Resource Owner** grant type to the native mobile app(s) that will be configured to serve as single sign-on proxies. This grant type is required to be able to perform the 2-legged client registration.

  To assign the client the **Resource Owner** grant type, use the OAuth Services Mobile Clients configuration page. For more information, see Section 2.3.3.5, "Understanding the Mobile Clients Configuration Page."
- To code the device-native app to authenticate with OAM using Mobile and Social Services, refer to the following sections in the *Oracle Fusion Middleware Developer's Guide for Oracle Access Management*:
  - For iOS, see "Invoking Authentication Services With the iOS Client SDK."
  - For Android, see "Invoking Authentication Services With the Android Client SDK."
  - For the REST API, see "Mobile and Social Services REST Reference: Authentication and Authorization."

  When coding the app(s), implement the `/authenticate` endpoint, which is required to use this feature. This endpoint provides client registration, OAM token exchange, and URL redirection to the MobileSSOServlet. The `/authenticate` endpoint should accept a resource URL (`resource_url`) and Return URL (`http://<OAM-HOST>:<OAM-PORT>/ms_oauth/mobilesso`) as parameters.

  Once the app completes registration and the OAM token exchange, it returns control of the authentication process (along with the OAM_ID and resource_url) using the value of the return_url that was sent to `/authenticate` endpoint. Redirect the app back to the MobileSSOServlet using this URL:

  `/mobilesso?OAM_ID=`*`1234`*`&resource_url=http://<`*`OAM-host`*`>:7777/index.html`

  where:
  - `OAM_ID` = the OAM master token value
  - `resource_url` = URL of the resource

In this case, the redirect URL used to redirect the app back to the MobileSSOServlet will be `http://<OAM-HOST>:<OAM-PORT>/ms_oauth/mobilesso?OAM_ID=1234&resource_url=http://<OAM-host>:7777/index.html`

2.7.2 Configuring the MobileSSOServlet Authentication Scheme

Use these steps to configure a new authentication scheme (MobileSSOServlet authentication scheme) in Access Manager to protect the Web resource(s).

1. Log on to the Oracle Access Management Administration Console.

   The Application Security Launch Pad opens.

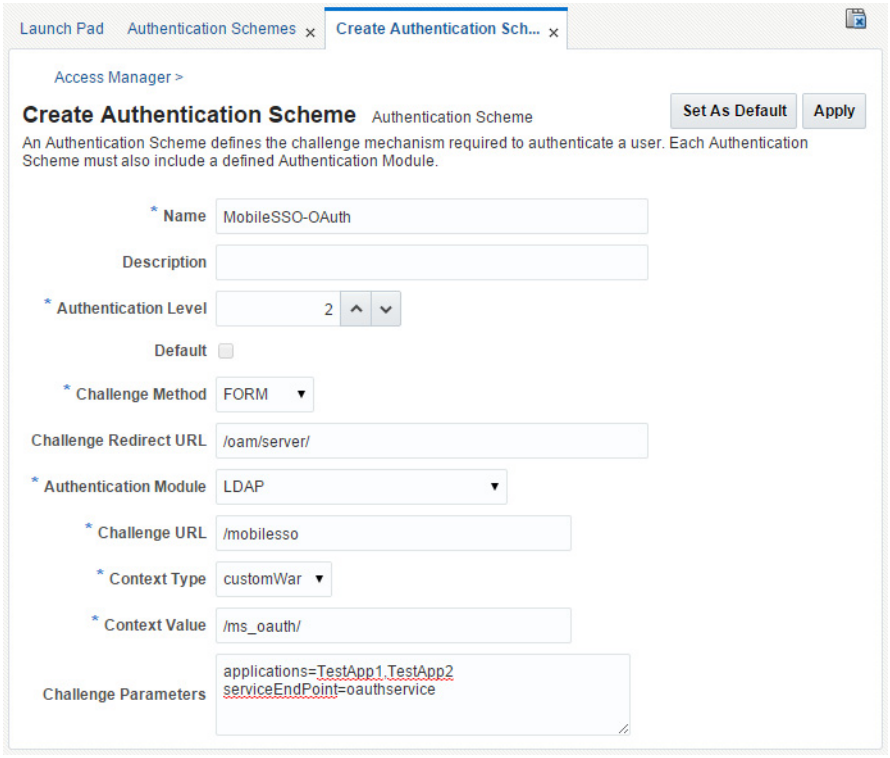
2. Under **Access Manager** click **Authentication Schemes**, then click the **Create Authentication Scheme** button.

   The "Create Authentication Scheme" tab opens.

3. Create a new Authentication Scheme by completing the form as follows:

   - **Name:** MobileSSO-OAuth
   - **Authentication Level:** 2
   - **Challenge Method:** FORM
   - **Challenge Redirect URL:** `/oam/server/`
   - **Authentication Module:** LDAP
   - **Challenge URL:** `/mobilesso`
   - **Context Type:** customWar
   - **Context Value:** `/ms_oauth/`
   - **Challenge Parameters:**

     ```
     applications=TestApp1,TestApp2
     serviceEndPoint=oauthservice
     ```



Screen capture of Create Authentication Scheme form configured with values

****

4. In the Oracle Access Management Administration Console, do the following:

   a. Create a new Authentication Policy in an Application Domain and assign it the following Authentication Scheme:

      Authentication Scheme: MobileSSO-OAuth

      (*MobileSSO-OAuth* is the scheme that was created in step one.)

   b. Create an HTTP Resource, for example `/mobileoauthapp`, and protect the resource using the created Authentication Scheme (MobileSSO-OAuth). This is the URI that will be accessed from the mobile web browser (mobile Safari for iOS) and protected by a WebGate.

2.8 Configuring the Mobile Security Manager Plug-in

The Mobile Security Manager Plug-in is an optional plug-in for use with Oracle Mobile Security Suite (OMSS). If you will be using Oracle Mobile Security Suite, see Section 1.9, "Understanding OAuth Services Plug-ins" to learn more about this plug-in.

Before you use this plug-in, you must configure the MSM data source in Oracle Access Management. Complete these steps to change the target of the `omsm_ds` JDBC data source to `oam_server1`.

1. Log in to the WebLogic console:

   `http://host:port/console`

2. Navigate to: `base_domain > Services > Data Sources`

3. In the **Data Sources** table, click `omsm_ds`.

4. Click the **Targets** tab.
5. Select `oam_server1`.

3

Using the OAuth Services API

This chapter describes the Oracle Access Management OAuth Services API. This chapter includes the following topics:

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 3–1 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 3–2.

```
https: //client.example.com/return?code=eyJhbG...rWWk8hbs_o6uY&state=xyz
```

Table 3–2 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
- Getting a User Identity Token

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

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'
```

4

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:

4.1 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.

4.2 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.

4.2.1 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.

4.2.2 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
```

4.2.3 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:

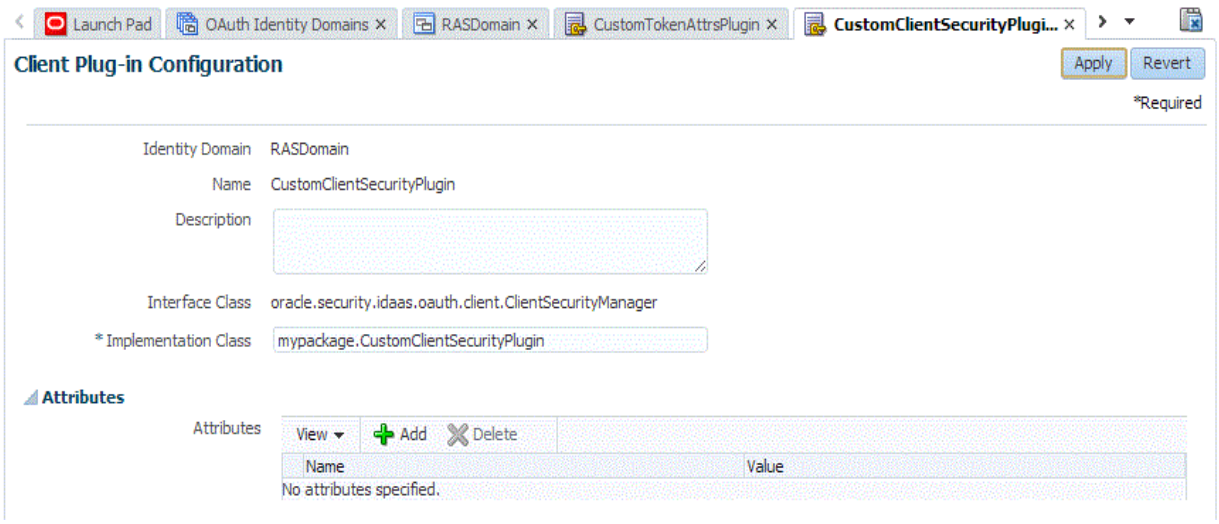
  ```
  $ORACLE_HOME/user_projects/domains/base_domain/config/fmwconfig/oic/plugins/
  ```

- 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.



4.2.4 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;

    }
}
```

4.3 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.

4.3.1 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.)

4.3.2 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.

4.3.3 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.

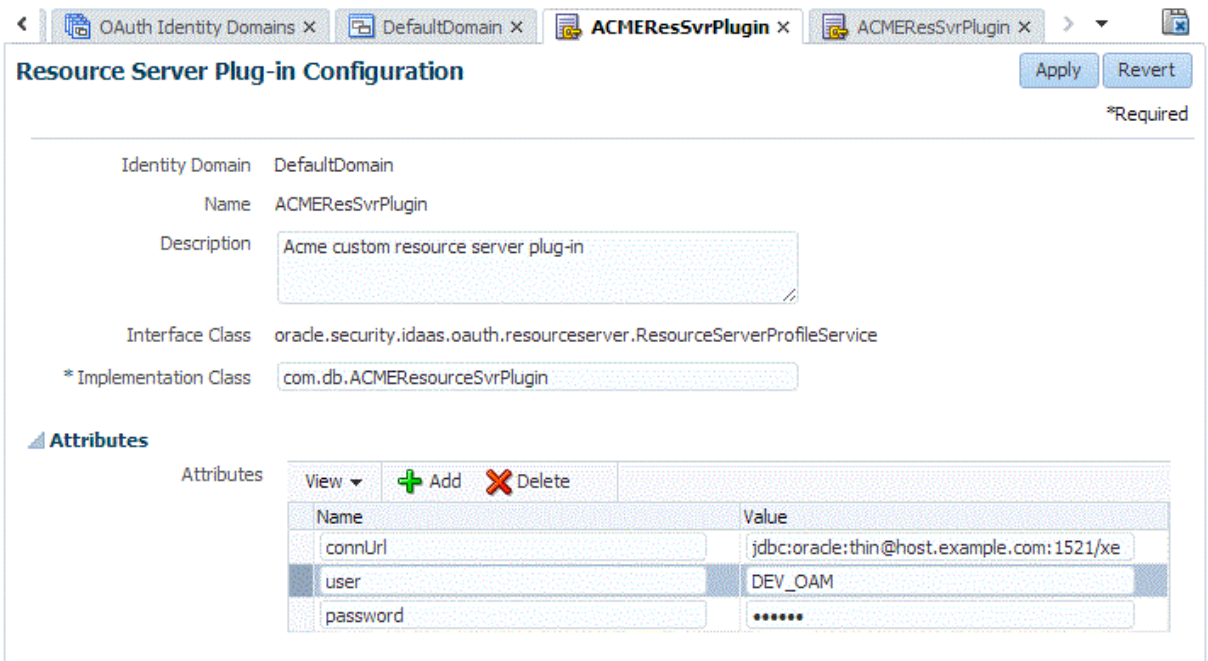
Use the OAM console to create your new client security plug-in.

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;
}
```

4.3.4 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();
    }
}
```

4.4 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.

4.4.1 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:

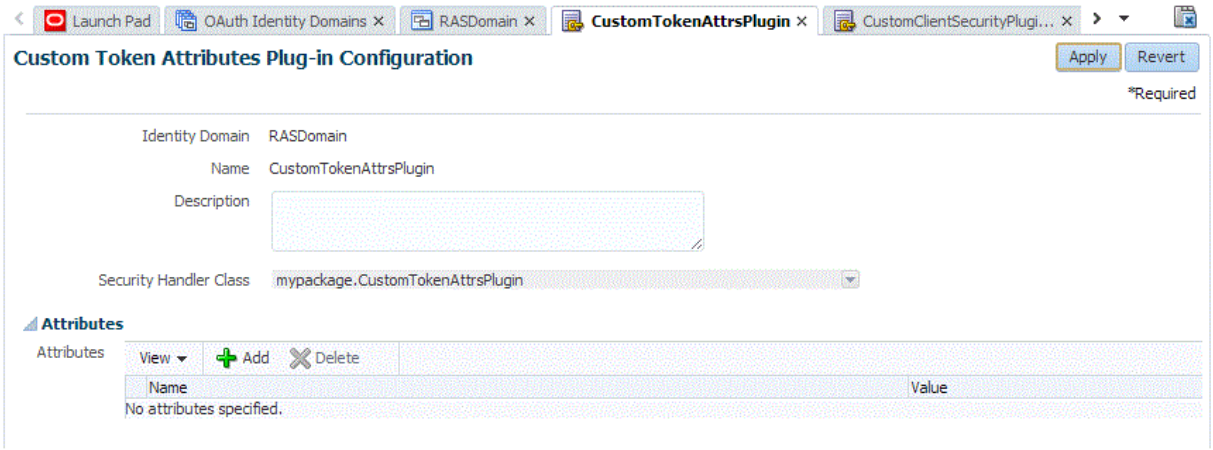
  ```
  $ORACLE_HOME/user_projects/domains/base_domain/config/fmwconfig/oic/plugins/
  ```

- 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.



4.4.2 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()";
    }
```

```
}
```

4.5 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).

4.5.1 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.