Oracle® Cloud
Administering Oracle Cloud Identity Management
Release 17.2
E59052-15

May 2017
Documentation for Oracle Cloud account administrators, security administrators, and identity domain administrators that explains how to configure Federation SSO and how to provision OAuth resources and clients using the self-service user interface (UI) and how to protect Oracle Cloud services using two-legged OAuth, service-to-service authorization.
## Contents

Preface .................................................................................................................................................................. v
Audience ............................................................................................................................................................ v
Scope of the Guide ............................................................................................................................................... v
Related Resources ............................................................................................................................................... v
Conventions..................................................................................................................................................... vi

1 Managing Oracle Single Sign-On
   Overview of SSO Configuration Tasks ........................................................................................................ 1-1
   Exploring the SSO Configuration Page in My Services .............................................................................. 1-2
   Configuring Oracle Cloud as the Service Provider ..................................................................................... 1-3
   Configuring an Identity Provider ................................................................................................................... 1-5
   Testing SSO ................................................................................................................................................... 1-6
   Problems Identified by Testing SSO .............................................................................................................. 1-6
   Enabling SSO ............................................................................................................................................... 1-7
   Enabling Sign In With Identity Domain Credentials .................................................................................. 1-7
   Removing Users .......................................................................................................................................... 1-8
   Updating SSO Metadata ............................................................................................................................... 1-8
   Troubleshooting SSO .................................................................................................................................... 1-9

2 Managing OAuth Resources and Clients
   Exploring the OAuth Administration Page in My Services ....................................................................... 2-2
   How Do I Set Up OAuth in Oracle Cloud? .................................................................................................... 2-4
   How Do I Administer OAuth in Oracle Cloud? ............................................................................................ 2-5
   Registering New Resources in Oracle Cloud ............................................................................................... 2-6
   Overview of Managing OAuth Resources ................................................................................................... 2-7
   Viewing OAuth Resources ............................................................................................................................. 2-7
   Updating OAuth Resources ............................................................................................................................ 2-8
   Deleting OAuth Resources ............................................................................................................................ 2-8
   Overview of OAuth Client Configuration Tasks .......................................................................................... 2-10
   Overview of Registering OAuth Clients ....................................................................................................... 2-10
   Registering Client Information in OAuth ..................................................................................................... 2-11
   Registering an Untrusted OAuth Client ...................................................................................................... 2-11
Registering a Trusted OAuth Client ................................................................. 2-12
Importing an OAuth Certificate from a Key Pair ............................................ 2-14
Extracting a Certificate by Using openssl ..................................................... 2-15
Extracting a Certificate by Using the Certificate Import and Certificate Export Wizards 2-15
Associating a Certificate with an OAuth Client .............................................. 2-16
Overview of Managing OAuth Clients ............................................................. 2-16
Viewing OAuth Clients .................................................................................... 2-17
Updating OAuth Clients .................................................................................. 2-19
Managing Client Certificates .......................................................................... 2-20
Deleting OAuth Clients .................................................................................... 2-21
Troubleshooting OAuth .................................................................................. 2-22

3 Securing Authorizations in Oracle Cloud

How Do I Use Authorization Grants? ............................................................... 3-1
Resource Owner Password Credentials Workflow ......................................... 3-3
Step-by-Step Workflow of the Resource Owner Password Credentials Grant ........................................ 3-3
Using REST API Calls for the Resource Owner Password Credentials Grant ........................................ 3-4
  Obtaining an Access Token by Using the User Credentials Without a Client Assertion ..... 3-4
  Obtaining an Access Token by Using the User Credentials and a JWT Client Assertion .... 3-7
Client Credentials Grant Workflow ................................................................. 3-9
Step-by-Step Workflow of the Client Credentials Grant .................................. 3-10
Using REST API Calls for the Client Credentials Grant ................................... 3-11
  Obtaining an Access Token by Using a Client Authorization Header ................. 3-11
  Obtaining an Access Token by Using a Self-Signed Client Assertion ............... 3-13
User Assertion Workflow .................................................................................. 3-16
Using REST API Calls for the User Assertion Grant ........................................ 3-17
  Obtaining an Access Token by Using a Self-Signed User Assertion and the Client
  Credentials ..................................................................................................... 3-18
  Obtaining an Access Token by Using a Self-Signed User Assertion and a Client Assertion
  ...................................................................................................................... 3-20
Successful Authorization .................................................................................. 3-23
Authorization Error ............................................................................................ 3-26
Oracle® Cloud Administering Oracle Cloud Identity Management explains how to provision Oracle Single Sign-On (SSO) and configure various OAuth resources and clients using the self-service user interface.

Topics:
- Audience
- Scope of the Guide
- Related Resources
- Conventions

Audience
This guide is intended for Oracle Cloud account administrators and customers buying Oracle Cloud services, who want to configure SSO and Identity Federation using Security Assertion Markup Language (SAML), and manage various OAuth resources and clients.

Scope of the Guide
The tasks explained in the guide include:
- Single Sign-On (SSO)
- OAuth resource management
- OAuth client management

Shared Identity Management (SIM) uses SAML to function as a SAML service provider to Oracle Fusion Applications SAML identity provider. This is done through Oracle Public Cloud support. In addition, SIM operates as a SAML service provider to federate with a SAML identity provider, such as Oracle Fusion Applications, Oracle Access Management, Microsoft Active Directory Federation Services (ADFS), and Shibboleth.

Related Resources
For additional documentation related to your Oracle Cloud service, visit the Oracle Cloud website at:

http://cloud.oracle.com
Open the **Support** menu at the top of the page and select **Documentation** to access the Oracle Cloud Documentation home page. Search or browse the library for documentation specific to your application, infrastructure, or platform cloud service.

## Conventions

The following text conventions are used in this guide:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
Managing Oracle Single Sign-On

By implementing Oracle Single Sign-On, your users can access multiple Oracle Cloud services using one set of credentials. Also, logging out of one service logs a user out of all other services.

As administrator, you configure SSO because you want to use identity federation between Oracle Cloud as service provider and an external identity provider. This task requires you to configure Oracle Cloud as service provider, prepare your identity provider, test your SSO configuration, and finally, enable SSO.

Topics:

• Overview of SSO Configuration Tasks
• Exploring the SSO Configuration Page in My Services
• Configuring Oracle Cloud as the Service Provider
• Configuring an Identity Provider
• Testing SSO
• Problems Identified by Testing SSO
• Enabling SSO
• Enabling Sign In With Identity Domain Credentials
• Removing Users
• Updating SSO Metadata
• Troubleshooting SSO

Note: To learn more about the concepts of Oracle Single Sign-On, see About SSO in Understanding Identity Concepts.

Overview of SSO Configuration Tasks

As administrator, you enable SSO so your users can use their company credentials to log in to all applications, including Oracle Cloud applications. This requires you to configure SAML 2.0 between Oracle Cloud and the identity provider.

The following table shows you the steps that you must follow when configuring SSO on the SSO Configuration page from My Services in Oracle Cloud:
Task | Description | Additional Information
--- | --- | ---
Configure Oracle Cloud as a service provider. | Go to the Users page and then click the SSO Configuration tab to configure Oracle Cloud as the service provider. | Configuring Oracle Cloud as the Service Provider

Configure an identity provider. | After you configure Oracle Cloud as a service provider, you configure your identity provider. | Configuring an Identity Provider

Test Single Sign-On. | Test your SSO configuration before enabling SSO. | Testing Single Sign-On

Identify problems by testing SSO. | Testing SSO can identify a number of problems that you must fix before you can enable SSO. | Problems Identified by Testing SSO

Enable SSO. | You must enable SSO before you can use it. | Enabling SSO

Enable sign in with identity domain credentials. | If you want users (such as identity domain administrators) to log in using their identity domain credentials, you must enable this option. | Enabling Sign In With Identity Domain Credentials

Remove users. | After you enable SSO, ensure that users do not have credentials in Oracle Cloud. | Removing Users

Update SSO metadata. | At some point, after you’ve enabled SSO in production, you might need to update the SSO metadata. | Updating SSO Metadata

Troubleshoot SSO. | If you can’t resolve a configuration problem by testing SSO, then you must troubleshoot the configuration. | Troubleshooting SSO

Exploring the SSO Configuration Page in My Services
The SSO Configuration page in My Services helps Oracle Cloud account administrators and customers buying Oracle Cloud services to configure SSO between your identity provider and with Oracle Cloud as the service provider.

What You Can Do from the SSO Configuration Page
The following table describes what you can do from the SSO Configuration page:
<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Remove Users</strong></td>
<td>Click Remove Users to remove users that you added in Oracle Cloud before enabling SSO. To learn more about why you should remove these users, see Removing Users.</td>
</tr>
<tr>
<td><strong>Configure SSO</strong></td>
<td>Click Configure SSO to start a set of tasks to configure an identity provider, service provider, and SSO. To learn more about the configuration steps and the tasks that you must perform, see Managing Oracle Single Sign-On. The Configure an Identity Provider with Oracle Cloud - Tutorial Series guides you through the configuration steps for different identity providers.</td>
</tr>
</tbody>
</table>

### What You Can See from the SSO Configuration Page

The SSO Configuration page displays the following information:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Not Configured" /></td>
<td>You can see the status of the SSO configuration when you access the SSO Configuration page. Before configuring SSO, it shows that SSO is Not Configured.</td>
</tr>
<tr>
<td><img src="image" alt="Configure SSO" /></td>
<td>You can start configuring SSO between Oracle Cloud and your identity provider.</td>
</tr>
</tbody>
</table>

### Configuring Oracle Cloud as the Service Provider

To configure SSO, start with configuring Oracle Cloud as service provider.

To configure SAML 2.0 SSO between Oracle Cloud as service provider and the identity provider:

1. Go to the My Services dashboard page and click Users. Then click the SSO Configuration tab.

2. Click Configure SSO.

   The Configure SSO page is displayed.

3. Select whether to import identity provider metadata or enter provider the metadata manually. Your choice depends on whether your identity provider can export metadata.

4. The next step depends on the selection that you made in Step 3.

   a. If your Identity Provider can export metadata, then you can import the metadata into Oracle Cloud. Select Import Identity Provider metadata. Click Choose File and upload the identity provider metadata file (such as idp_metadata.xml).
b. For the SSO Protocol field, HTTP POST is recommended and is the default. The SSO Protocol field value refers to the SAML binding that’s used. SAML bindings define how the SAML protocols map to the type of transport used. Oracle Cloud supports HTTP POST and HTTP Artifact. The HTTP POST binding defines how SAML protocol messages can be transported with the base64-encoded content of a form control within an HTML form. The HTTP Artifact binding defines how a reference (or artifact) to a SAML request or response is transported over HTTP; the artifact (a small representation of a complete SAML assertion) can be embedded in a URL as a query string parameter, or it can be placed in a hidden form control.

c. Select the User Identifier field. The user identifier is the Oracle LDAP directory attribute that’s used to map the user information contained in the incoming SSO SAML assertion to an Oracle Cloud user. It’s either the user’s email address or the user ID. Select User’s Email Address.

d. Select the contained in field. If the User Identifier is the user’s email address, then the contained in field must be NameID.

**Note:**

If the User Identifier value is the user ID, then the contained in field must be the SAML attribute and you must specify the name of the SAML attribute for the contained in field such as SamAccountName in the case of Microsoft Active Directory Federation Services.

e. Click Save.

f. If your identity provider can’t export metadata, then you must enter metadata information manually, which means you must also provide the Issuer ID and SSO Service URL (this is the SAML assertion consumer URL), and indicate whether Global logout should be enabled. You must also load your identity provider’s signing certificate and encryption certificate.
Configuring an Identity Provider

After you configure Oracle Cloud as a service provider, configure your identity provider in the Configure your Identity Provider Information section of the SSO Configuration page.

1. Go to the Users page and click the SSO Configuration tab. Then scroll down to the Configure your Identity Provider Information section.

2. What you need to configure the identity provider depends on one of the following:

   • If your identity provider can import metadata, export the metadata from the Service Provider to import into the Identity Provider by doing the following:

     a. In the Configure your Identity Provider Information section click Export Metadata, then select Provider Metadata.

     b. Save the metadata to a local file as SP_metadat.xml.

   • If your identity provider can’t import metadata, then copy and paste the provider ID and URLs into a SAML 2.0 file to be used by the identity provider. Download the certificates from the service provider.

3. Configure your identity provider, using its configuration interface. The configuration steps are specific to each identity provider.
Testing SSO

Test SSO to identify any SSO configuration problems.

Go to the Users page and then click the SSO Configuration tab.

1. On the SSO Configuration page in the Test your SSO section, click Test.

   The Initiate Federation SSO page appears.

2. Click Start SSO.

   Clicking Start SSO triggers a Federation SSO workflow. You’re redirected to the identity provider’s login page and challenged for authentication.

3. Log in as an administrator. After the Federation SSO is performed, the result is displayed in the Test SSO page.

4. The next step depends on whether the test is successful:

   • If the test is successful, then proceed to Enabling SSO.
   • If the test is unsuccessful, then view the test results to determine the cause. See Problems Identified by Testing SSO

Problems Identified by Testing SSO

The Test SSO feature can identify various problems.

The Assertion Couldn’t be Mapped to an Oracle Cloud User

This may occur for the following reasons:

• The SIM user corresponding to the identity provider user doesn’t exist.
• Oracle Cloud was incorrectly configured to map the incoming SSO assertion.

An Error Occurs When Oracle Cloud Consumes the SAML Assertion

To resolve this problem:

• Ensure that the Oracle Cloud federation server has the latest identity provider metadata and signing certificate.
• If the identity provider encrypts the assertion, ensure that the identity provider has the correct Oracle Cloud encryption certificate.

After Logging Out, the User is Automatically Logged in Again

This typically occurs when Oracle Cloud is wired with the identity provider using HTTP basic authentication or with Microsoft Active Directory Federation Services identity provider using Windows Integrated Authentication as the challenge mechanism. Upon logging out and performing the SAML 2.0 logout protocol, the user is automatically logged in again. The identity provider can’t log the user out because:

• The browser caches the HTTP basic authentication credentials and thus the identity provider can’t log the browser out.
• The Windows Desktop machine where the user is signed in automatically signs in in the browser with Microsoft Active Directory Federation Services identity provider, so the identity provider can’t log the browser out.

To resolve this problem, change the authentication mechanism at the identity provider.

**The Identity Metadata Fails to Be Uploaded from the Console.**

To resolve this problem:

• Ensure that the metadata wasn’t modified.

• When downloading the metadata from the identity provider, save it using the **File —> Save As** command. That is, don’t copy and paste the contents of the browser, because this action modifies the contents of the metadata.

**SSO Fails Because the Assertion Isn’t Signed.**

The Oracle Cloud federation server requires the SAML assertion to be signed. Ensure that the assertion is signed and contains a digital signature element, even if the SSO response is signed.

**Problems that Can’t Be Resolved**

If you can’t resolve the problem using the Test feature, proceed to Troubleshooting SSO.

**Enabling SSO**

Until you specifically enable SSO, you can’t use it. After SSO is enabled, you should be able to authenticate through the identity provider, after selecting **Sign in using your company ID** on the **Sign In to Oracle Cloud** page.

Go to the **Users** page and then click the **SSO Configuration** tab. If the status in the **Enable SSO** section is **SSO is Not Enabled**, and you tested SSO successfully, and you want to enable SSO, then click **Enable SSO** to enable SSO for all Oracle Cloud services. Until you do this, SSO isn’t enabled.

After you enabled SSO, you can disable it from the **Enable SSO** section of the **SSO Configuration page**.

**Enabling Sign In With Identity Domain Credentials**

After SSO is enabled, users typically sign in using their identity provider credentials. If you want your users to be able to sign in with their identity domain (Oracle Cloud) credentials, you need to enable this option.

After you enable SSO, you have the option to allow users to sign in with their identity domain credentials as well. This option is disabled by default because typically, as administrator you want to force users to log in using their identity provider credentials.

To enable the option for users to sign in with their identity domain credentials:

1. Go to the **Users** page and then click the **SSO Configuration** tab.

2. Go to the **Enable Sign In to Oracle Cloud Services with Identity Domain credentials** section. Click **Enable**.
3. A confirmation window appears informing you that after enabling, users that do have credentials in their identity domains (for example identity domain administrators), will be able to sign in to Oracle Cloud services using either their identity provider or identity domain credentials.

**Note:** You can’t enable signing in with identity domain credentials, if SSO was auto-configured for your system. The Enable Sign In to Oracle Cloud Services with Identity Domain credentials button is disabled in this case.

After you enabled sign in to Oracle Cloud with identity domain credentials, you can disable it from the Enable Sign In to Oracle Cloud Services with Identity Domain credentials section of the page. This is necessary, if you want to force users to sign in only with their identity provider credentials.

### Removing Users

Remove all users without the identity domain administrator role after you enable SSO.

After you enable SSO, only users that have the identity domain administrator role or were created before SSO was enabled, have credentials in Oracle Cloud. To avoid maintaining credentials in two places after enabling SSO, you typically delete the existing users and then reimport them. This step ensures that the users don’t have credentials in Oracle Cloud and can access Oracle Cloud applications only with their company credentials.

To delete all users that don’t have the identity domain administrator role assigned:

1. Go to the Users page and then click the SSO Configuration tab.
2. Click Remove Users.
3. A window appears confirming that all users without the identity domain administrator role will be removed, and that this operation can’t be undone.
4. Click Remove Users to remove all users who don’t have the identity domain administrator role assigned.
5. A window displays the progress of the removal process, and then the number of users removed.

### Updating SSO Metadata

After you’ve enabled SSO in production, you might want to update the SSO metadata.

Reasons for updating the metadata include:

- The identity provider or service provider certificate has expired.
- The identity provider or provider key has been compromised.
- The identity provider URL endpoints need to be updated.

If any of these reasons applies, then:

1. Schedule an update of the SSO metadata in advance, because it requires an outage.
2. Disable SSO using Disable SSO.
3. Update the identity provider or service provider metadata as needed.

4. Test the configuration, as described in Testing SSO.

5. After testing shows that SSO is working correctly, reenable SSO by clicking Enable SSO as described in Enabling SSO.

Troubleshooting SSO

If you can’t resolve a configuration problem by using the Test feature, then troubleshoot the configuration by following these steps.

1. Review the Known Issues guide for any similar problem.

2. Review any changes made on the identity provider and Oracle Cloud service provider before the problem in the SSO workflow.

3. Capture an HTTP trace of the SSO workflow, using a tool such as Fiddler Web Debugging Tool.

4. Review the workflow to determine the point where the SSO workflow terminated and which identity-related components are involved: identity provider, service provider, web tier, gateways, proxies, and firewalls.

5. Review the protocol messages and component logs to identify exceptions.

6. Go to MyOracle Support to review known issues and find out if your problem exists there.

7. If you’ve performed all troubleshooting steps and you’re confident that the problem is due to Oracle Cloud, then contact Oracle Support Services. Be ready to provide all your information, including a Fiddler trace, identity provider metadata, and identity provider logs.
Managing OAuth Resources and Clients

OAuth 2.0 is an authorization framework that enables an application or service to obtain limited access to a protected HTTP resource. In OAuth, the applications are called clients; they access protected resources by presenting an access token to the HTTP resource. As an administrator, you configure OAuth resources and clients and administer them.

Topics:

- Exploring the OAuth Administration Page
- How Do I Set Up OAuth in Oracle Cloud?
- How Do I Administer OAuth in Oracle Cloud?
- Registering New Resources in Oracle Cloud
- Overview of Managing OAuth Resources
- Viewing OAuth Resources
- Updating OAuth Resources
- Deleting OAuth Resources
- Overview of OAuth Configuration Tasks
- Overview of Registering OAuth Clients
- Registering Client Information In OAuth
- Registering an Untrusted OAuth Client
- Registering a Trusted OAuth Client
- Importing an OAuth Certificate from a Key Pair
- Extracting a Certificate by Using openssl
- Extracting a Certificate by Using the Certificate Import and Certificate Export Wizards
- Overview of Managing OAuth Clients
- Viewing OAuth Clients
- Updating OAuth Clients
- Deleting OAuth Clients
Managing Client Certificates
Troubleshooting OAuth

Exploring the OAuth Administration Page in My Services
The OAuth Administration page in My Services helps Oracle Cloud account administrators and customers buying Oracle Cloud services to configure OAuth clients and resources. You can register new OAuth clients and resources, grant/revoke API access, and manage the settings of resources and clients.

What You Can Do from the OAuth Administration Page
The following table describes what you can do from the OAuth Administration page.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register Resource</td>
<td>Register a new OAuth resource.</td>
</tr>
<tr>
<td>Import</td>
<td>Register resources by importing data from a file.</td>
</tr>
<tr>
<td>Find Resource</td>
<td>Enter the name of a resource (or a part of the name), and then search for OAuth resources.</td>
</tr>
<tr>
<td>Register Client</td>
<td>Register a new OAuth client.</td>
</tr>
<tr>
<td>Show: All Clients</td>
<td>Show all registered OAuth clients.</td>
</tr>
<tr>
<td>Find Client</td>
<td>Enter the name of a client (or a part of the name), and then search for OAuth clients.</td>
</tr>
<tr>
<td>Manage Certificates</td>
<td>Create a key pair for the OAuth client certificate.</td>
</tr>
<tr>
<td>Download</td>
<td>Download OAuth client certificates.</td>
</tr>
</tbody>
</table>
### What You Can See from the OAuth Administration Page

The OAuth Administration page displays the following information:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Register Resource" /></td>
<td>Register a new OAuth resource.</td>
</tr>
<tr>
<td><img src="image" alt="Register Resource" /></td>
<td>Register resources by importing data from a file.</td>
</tr>
<tr>
<td><img src="image" alt="Modify" /></td>
<td>View the various options to edit an existing OAuth resource:</td>
</tr>
<tr>
<td></td>
<td>- Select <strong>Modify</strong> to change the properties of the resource.</td>
</tr>
<tr>
<td></td>
<td>- Select <strong>Remove</strong> to delete the OAuth resource.</td>
</tr>
<tr>
<td><img src="image" alt="Find Resource" /></td>
<td>Enter the name of a resource (or a part of the name), and then search for OAuth resources.</td>
</tr>
<tr>
<td><img src="image" alt="Register Client" /></td>
<td>Register a new OAuth client.</td>
</tr>
<tr>
<td><img src="image" alt="Show: All Clients" /></td>
<td>Show all registered OAuth clients.</td>
</tr>
<tr>
<td><img src="image" alt="Show: Trusted Clients" /></td>
<td>Show all OAuth clients that are trusted.</td>
</tr>
<tr>
<td><img src="image" alt="Show: Untrusted Clients" /></td>
<td>Show all OAuth clients that are untrusted.</td>
</tr>
<tr>
<td><img src="image" alt="Show: User Defined Clients" /></td>
<td>Show all OAuth clients that are user-defined</td>
</tr>
<tr>
<td><img src="image" alt="Show: Infrastructure Clients" /></td>
<td>Show all OAuth Clients that are created by the infrastructure.</td>
</tr>
</tbody>
</table>
### How Do I Set Up OAuth in Oracle Cloud?

You're an administrator at a company that has purchased some Oracle Cloud services. You want to configure OAuth to secure access to those services. As an administrator, you are responsible for setting up OAuth. During set up, you need to configure OAuth clients and resources in Oracle Cloud.

You have the following responsibilities as an administrator:

- Configure and manage OAuth resources.
- Configure and manage OAuth clients.
- Ensure that the communication between different services (on-premises and cloud) is secure.

The following table describes the steps to follow when setting up OAuth using the OAuth Administration page from My Services in Oracle Cloud:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Client</td>
<td>Enter the name of a client (or a part of the name), and then search for OAuth clients.</td>
</tr>
<tr>
<td>Show Secret</td>
<td>Shows the secret of the OAuth client.</td>
</tr>
<tr>
<td>Modify</td>
<td>View the various options to edit an existing OAuth client:</td>
</tr>
<tr>
<td>Import Certificate</td>
<td>• Select <strong>Modify</strong> to change the client properties.</td>
</tr>
<tr>
<td>Export Certificate</td>
<td>• Select <strong>Import Certificate</strong> to import a new certificate for the OAuth client.</td>
</tr>
<tr>
<td>Remove</td>
<td>• Select <strong>Export Certificate</strong> to export the existing client certificate.</td>
</tr>
<tr>
<td></td>
<td>• Select <strong>Remove</strong> to delete the client.</td>
</tr>
<tr>
<td>Manage Certificates</td>
<td>Create a key pair for the OAuth client certificate.</td>
</tr>
<tr>
<td>Download</td>
<td>View a list of the various client certificate that you can download. Select the appropriate option to save the certificate locally.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Download Menu</th>
<th>OAuth Signing Certificate</th>
<th>Root CA Certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Description</td>
<td>Additional Information</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>Register an OAuth resource.</td>
<td>Go to the OAuth Administration page to register a new OAuth resource.</td>
<td>Registering New Resources in Oracle Cloud</td>
</tr>
<tr>
<td>Register a trusted OAuth client.</td>
<td>After you register an OAuth resource, configure and register an OAuth client.</td>
<td>Registering a Trusted OAuth Client</td>
</tr>
<tr>
<td></td>
<td>Decide whether you want the OAuth client to be trusted or untrusted.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To register a new trusted OAuth client, go to the OAuth Administration page.</td>
<td></td>
</tr>
<tr>
<td>Register an untrusted OAuth client.</td>
<td>After you register an OAuth resource, you can configure and register either a trusted or an untrusted OAuth client.</td>
<td>Registering an Untrusted OAuth Client</td>
</tr>
<tr>
<td></td>
<td>To register a new untrusted OAuth client, go to the OAuth Administration page.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** To learn the difference between trusted and untrusted OAuth clients, see OAuth Client Types and Digital Signatures in Understanding Identity Concepts.

---

**How Do I Administer OAuth in Oracle Cloud?**

As an administrator, you’re responsible for managing OAuth clients and resources in Oracle Cloud.

**Administering OAuth in Oracle Cloud: Task Flow**

An OAuth client and an OAuth resource can be managed in different ways. You may need to get information about an existing client or resource. You can search for an OAuth client or a resource to get this information. As an administrator, you’re required to modify the configuration of an OAuth client or a resource. If security is compromised, then you may even need to remove the OAuth client. Similarly, a protected OAuth resource can also be removed from Oracle Cloud.

This section describes the high-level tasks for administering OAuth in Oracle Cloud. Both OAuth clients and resources can be administered in Oracle Cloud.

OAuth clients and resources can be managed by using the OAuth Administration page.

To manage existing OAuth resources:

- Search: See Viewing OAuth Resources.
- Update: See Updating OAuth Resources.
- Delete (Remove): See Deleting OAuth Resources
To manage existing OAuth clients (both trusted and untrusted):

- Search: See Viewing OAuth Clients.
- Update: See Updating OAuth Clients
- Delete (Remove): See Deleting OAuth Clients

To modify the certificates of both trusted and untrusted OAuth clients, see Managing Client Certificates

Registering New Resources in Oracle Cloud

From the OAuth Administration page, you can register a new resource in Oracle Cloud. A resource is a protected service in Oracle Cloud. When you register a new resource, you define some parameters and these parameters are used in authorizing the client request to those services,

To register new resources in Oracle Cloud using the UI:

1. Log in to the user’s identity domain in Oracle Cloud and click Users.
2. Click the OAuth Administration tab.
   You can see the list of existing resources.
3. To register multiple resources at one time by using the data from a comma-separated values (CSV) file, click Import.

4. To register a new resource, click Register. To register the resource (the * indicates mandatory fields), enter the following information in these fields:
   a. Name: Enter the name of the OAuth resource. If another resource in the same application already uses this name, then an error is thrown.
   b. Application: Enter the name of the application, which is the service name. The resource is part of the application.
   c. Description: Enter a description of the OAuth resource. A description is optional. If this field is left blank, the description value defaults to the name of the resource.
**Overview of Managing OAuth Resources**

You can search for an existing OAuth resource, modify its properties, and remove an OAuth resource, if necessary.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>View OAuth resources.</td>
<td>View an existing OAuth resource from the OAuth Administration page.</td>
<td>Viewing OAuth Resources</td>
</tr>
<tr>
<td>Modify OAuth resources.</td>
<td>Change the properties of an OAuth resource by using the OAuth Administration page.</td>
<td>Updating OAuth Resources</td>
</tr>
<tr>
<td>Delete OAuth Resources.</td>
<td>Remove OAuth resources by using the OAuth Administration page.</td>
<td>Deleting OAuth Resources</td>
</tr>
</tbody>
</table>

**Viewing OAuth Resources**

An existing OAuth resource can be viewed at any time from the OAuth Administration page.

1. Log in to the user’s identity domain and click **Users**.
2. Click the OAuth Administration tab.

A list of all resources appears, including the resource name, application name, resource description, resource identifier, and API path.

3. To search for a resource, entering the name (partial name or full name) in the Search field.

All resources whose names match the pattern text that you enter in the Search field appear. The following example uses, res2 as search pattern in the Search field. As a result, only one resource appears (test_res2) because only one resource has res2 in its name.

---

**Updating OAuth Resources**

The properties of an existing OAuth resource can be modified.

To change properties of an existing OAuth resource using the UI:

1. Log in to the user's identity domain in Oracle Cloud and click Users.
2. Click the OAuth Administration tab. The list of resources appears.
3. Click the Action menu icon to the right of the resource that you want to update. A menu is displayed with two options: Modify and Remove.
4. To change the properties of the resource such as the resource’s description and the API path, click **Modify**. The name of the resource and the name of the application can’t be modified.

5. Click **Save**.

**Deleting OAuth Resources**

An existing OAuth resource can be removed at any time.

To delete a resource using the **OAuth Administration** page:

1. Log in to the user’s identity domain in Oracle Cloud and click **Users**.
2. Click the **OAuth Administration** tab. The list of resources appears.
3. Click the **Action** menu icon to the right of the resource that you want to delete. Select **Remove**.
4. The **Remove Resource** confirmation window appears. Click **Remove** to delete the resource.

![Remove Resource confirmation window]

### Overview of OAuth Client Configuration Tasks

As an administrator, you configure and administer OAuth clients for Oracle Cloud. These configuration and administration tasks consist of the following:

1. Registering trusted and untrusted OAuth clients using the OAuth administration UI.
2. Importing an OAuth certificate for an OAuth client.
3. Testing the OAuth client registration to ensure that the client is registered successfully.
4. Viewing OAuth clients from the OAuth administration UI in My Services.
5. Updating OAuth configuration settings and properties by using the OAuth administration UI.
6. Managing client certificates by using the OAuth Administration UI.
7. Testing the OAuth configuration.
8. Troubleshooting problems that may occur if OAuth isn’t working properly.

### Overview of Registering OAuth Clients

An OAuth client can be confidential or public. A confidential client is an application that’s capable of keeping a client password confidential to the world, whereas a public client doesn’t keep a client password confidential. A confidential client can be trusted or untrusted. OAuth clients, either trusted or untrusted, can be registered using the OAuth Administration page.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register an untrusted OAuth client.</td>
<td>Register a new untrusted client in the <strong>Register Client</strong> section of the <strong>OAuth Administration</strong> page in My Services.</td>
<td>Registering an Untrusted OAuth Client</td>
</tr>
<tr>
<td>Register a trusted OAuth client.</td>
<td>Register a new trusted client in the <strong>Register Client</strong> section of the <strong>OAuth Administration</strong> page in My Services.</td>
<td>Registering a Trusted OAuth Client</td>
</tr>
<tr>
<td>Task</td>
<td>Description</td>
<td>Additional Information</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Import an OAuth certificate from a key pair.</td>
<td>Associate a certificate with an OAuth client. The OAuth Administration page provides a helper function to generate and download a key pair.</td>
<td>Importing an OAuth Certificate from a Key Pair</td>
</tr>
<tr>
<td>Extract a certificate by using openssl.</td>
<td>On a Linux or UNIX system, use the openssl command to extract the certificate from the key pair that you downloaded from the OAuth Administration page.</td>
<td>Extracting a Certificate by Using openssl</td>
</tr>
<tr>
<td>Extract a certificate by using the Certificate Import and Certificate Export wizards.</td>
<td>Use the Windows Certificate Import and Certificate Export wizards to extract a certificate from the generated key pair that you downloaded from the OAuth Administration page.</td>
<td>Extracting a Certificate by Using the Certificate Import and Certificate Export Wizards</td>
</tr>
</tbody>
</table>

### Registering Client Information in OAuth

When an OAuth client is registered in OAuth, the client information is stored in the client profile.

The client information includes the client name and its attributes. An OAuth client created by an identity domain administrator is a user-managed OAuth client. If Oracle Cloud (Service Deployment Integration team) creates the OAuth client using a script, then it is a tenant-managed OAuth client. The attributes of the client stored in the client profile include whether it is tenant-managed or user-managed. The client profile also includes whether the OAuth client is trusted or untrusted. In addition to these attributes, the API path information of the client is stored in the Audience attribute.

### Registering an Untrusted OAuth Client

To register a new client, from My Service, go to the OAuth Administration page, and then to the Register Client section.

An OAuth client can be trusted or untrusted. By default, any new OAuth client created in Oracle Cloud is a confidential client (that is, the OAuth client’s credentials are never exposed directly). To create an untrusted client:

1. Click Register. The Register Client dialog box appears. The * indicates mandatory fields.

2. In the Name field, specify the name of the client.

3. In the Description field, provide explanatory information about the OAuth client.

4. In the Accessible Resources field, select the API resources to which this client should have access. This corresponds with the registered resources at the bottom of the OAuth Configuration page. This is a mandatory field. Select at least one of the listed APIs. If a single API path has more than one scope defined, the scopes are listed below the API path. One or more scopes can be selected for a given API path.

5. Leave the Trusted check box empty because this client is an untrusted client.
6. To upload a client certificate for an untrusted client, click **Browse** next to **Load Certificate**. This isn’t mandatory.

7. To register the untrusted OAuth client, click **Register**.

To learn more about the different fields in a client profile for an OAuth client, see [Registering Client Information in OAuth](#).

A portion of the client profile for the untrusted client previously registered is in the following example. The **isTrusted** flag has a value of **FALSE**.

```xml
<tenant>dom1</tenant>
<name>test_client_1</name>
<description>Sample untrusted client</description>
<isDisabled>false</isDisabled>
<appId>948bb730-a101-43b9-9497-d3ca33eb8d7f</appId>
<appSecret>948bb730-a101-43b9-9497-d3ca33eb8d7f</appSecret>
<clientType>CONFIDENTIAL_CLIENT</clientType>
<clientMetadata>
    <isTrusted>FALSE</isTrusted>
    <isTenantManaged>true</isTenantManaged>
</clientMetadata>
<activityData> <createdOn>06/05/2015 02:23:18 </createdOn> </activityData>
<audiences>http://www.example.com</audiences>
```

### Registering a Trusted OAuth Client

To register a new trusted client from **My Services**, go to the **OAuth Administration** page, and then to the **Register Client** section.
An OAuth client can be trusted or untrusted. To create a trusted OAuth client:

1. Click **Register**. The Register Client dialog box appears. The * indicates mandatory fields.

2. In the **Name** field, specify the name of the client.

3. In the **Description** field, provide explanatory information about the OAuth client.

4. In the **Accessible Resources** field, select the API resources to which this client should have access. This corresponds with the registered resources at the bottom of the OAuth Configuration page. This is a mandatory field. Select at least one of the listed APIs. If a single API path has more than one scope defined, then the scopes are listed below the API path. You can select one or more scopes for a given API path.

5. To indicate that the client is trusted, select the **Trusted** check box. For a trusted client, you must generate and upload a client certificate, as described in Importing an OAuth Certificate from a Key Pair. The signing algorithm must be **RS256**: RSASSA-PKCS-v1_5 using the SHA-256 hash algorithm.

6. Click **Register**.

To learn more about the different fields in a client profile for an OAuth client, see Registering Client Information in OAuth.

A portion of the prior client profile for the trusted client registered follows. The `isTrusted` flag has a value of TRUE.

```xml
<tenant>dom1</tenant>
<name>test_client_2</name>
<description>Sample trusted client</description>
<isDisabled>false</isDisabled>
```
Importing an OAUTH Certificate from a Key Pair

Import and associate an OAuth certificate with an OAuth client. This is mandatory for trusted clients and optional for untrusted clients.

The OAuth Administration page provides a helper function to generate and download a key pair, which contains a private key and the corresponding certificate. The key pair file is in the PKCS#12 format. PKCS #12 is one of the standards called Public-Key Cryptography Standards (PKCS) published by RSA Laboratories. The file name extension is usually .p12, but may have the older .pfx extension. You don’t have to use the helper function. You can generate the key pair by other means. (Some applications and operating systems include key pair generators.) If you’ve a certificate from another signing authority, there is no need to generate a key pair.

1. From the OAuth Administration page, go to the Manage Certificates section, and then click Create Key Pair.

   The Generate Key Pair dialog box appears.

2. Enter the appropriate information in the Subject DN and Key Store Password fields.

3. Click Generate.

4. After downloading the generated key pair, extract the private key and the corresponding certificate by using a tool such as the openssl command-line tool on Linux or UNIX, or the Certificate Import and Certificate Export wizards on Windows. For more information, see Extracting a Certificate by Using openssl and Extracting a Certificate by Using the Certificate Import and Certificate Export Wizards. The extracted file is a DER-encoded certificate. Distinguished Encoding Rules (DER) define a set of rules for encoding. The certificate file has the extension .cer.

5. Extract the certificate from the key pair.

   The Import Certificate dialog box appears.

6. To associate the certificate with a specific client, see Associating a Certificate with an OAuth Client.
Store the PKCS#12 format key pair securely and don’t share it. The OAuth client uses this key pair to sign OAuth protocol messages sent to the OAuth service in Oracle Cloud.

**Extracting a Certificate by Using openssl**

On a Linux or UNIX system, you can use the `openssl` command to extract the certificate from a key pair that you downloaded from the OAuth Configuration page.

To extract the certificate, use these commands, where `cer` is the file name that you want to use:

1. `openssl pkcs12 -in store.p12 -out cer.pem`
   
   This extracts the certificate in a .pem format.

2. `openssl x509 -outform der -in cer.pem -out cer.der`
   
   This formats the certificate in a .der format.

You can then associate `cer.der` with a client.

You can also extract the private key by using the command:

`openssl pkcs12 -in store.p12 -out pKey.pem -nodes -nocerts`

For more information, see the OpenSSL documentation.

**Extracting a Certificate by Using the Certificate Import and Certificate Export Wizards**

You can use the Windows Certificate Import and Certificate Export wizards to extract a certificate from the generated key pair that you downloaded from the OAuth Administration page.

To import the key pair and export the certificate, follow these steps:

1. Open the Certificate Import Wizard on your computer. You might be able to open it by double-clicking the key pair file. If not, follow these steps:

   a. In Internet Explorer, click **Tools**, and then select **Internet Options**.

   b. Click the **Content** tab. Click **Certificates**.

   c. Click **Import**. The Certificate Import Wizard opens.

2. Enter or browse to the file path and click **Next**. When prompted, enter the password for the private key. This was specified while generating the key pair earlier.

3. Select the **Automatically select the certificate store based on the type of certificate** check box.

4. Click **Next**.

5. Click **Finish**. A dialog box appears indicating that the key pair was imported into Internet Explorer successfully.

6. In Internet Explorer, click **Tools**, and then select **Internet Options**.
7. Click the **Content** tab, and then click **Certificates**.

8. Click the **Personal** tab, and then select the certificate that you just imported.


10. Click **Next**.

11. Select the **No, do not export the private key** check box.

12. Click **Next**.

13. Select the **DER encoded binary X.509 (CER)** check box.

14. Click **Next**.

15. Specify a file path, and then click **Save**.

16. Click **Next**.

17. To export the file to your local file system, click **Finish**. A dialog box appears indicating that the export was successful.

After completing all steps, you can associate the DER-encoded file with an OAuth client. For more information, see **Associating a Certificate with an OAuth Client**.

---

**Associating a Certificate with an OAuth Client**

Associate an OAuth certificate with an OAuth client. This is mandatory for trusted clients and optional for untrusted clients.

To associate an OAuth certificate with a specific OAuth client:

1. From the **My Services** page, go to the **OAuth Administration** page, and then to the **Register Client** section.

2. To associate the certificate with a specific client, click the **Action** menu for that client, and then select **Import Certificate**.

3. To select the certificate file, click **Browse**, and then click **Import**.

A success message is displayed.

---

**Overview of Managing OAuth Clients**

An existing OAuth client (trusted or untrusted) can be viewed, the properties of the client can be modified, certificates associated with the client can be managed, and the client can be removed.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>View OAuth clients.</td>
<td>An existing OAuth client can be viewed from the OAuth Administration page.</td>
<td>Viewing OAuth Clients</td>
</tr>
<tr>
<td>Modify OAuth clients.</td>
<td>The properties of an OAuth client can be changed using the OAuth Administration page.</td>
<td>Updating OAuth Clients</td>
</tr>
</tbody>
</table>
### Task Description Additional Information

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage client certificates.</td>
<td>Client certificates can be managed from the OAuth Administration page. You can access the root certificate and the OAuth certificate.</td>
<td>Managing Client Certificates</td>
</tr>
<tr>
<td>Delete an OAuth client.</td>
<td>An OAuth client can be removed by using the OAuth Administration page.</td>
<td>Deleting OAuth Clients</td>
</tr>
</tbody>
</table>

### Viewing OAuth Clients

View OAuth clients on the OAuth Administration page in My Services.

The OAuth Administration page in My Services lists registered clients. You can view information about clients in the Register Client section of the page.

- Trusted clients are indicated with the label **Trusted**.
- The first column of the list indicates the client name.
- The second column lists the ID, type (confidential trusted or untrusted), and modification information.
- Clicking **Show Secret** displays the client secret string. The client secret is used to authenticate the identity of the client to the service API when the client requests to access a user’s account, and must be kept private between the client and the API. Think of the secret string as a passphrase that proves to the authentication server that the client is authorized to make a request on behalf of the user.
Search for a Specific OAuth Client Type

• **View All Trusted Clients:** From the drop-down menu, select Show: Trusted Clients. All trusted clients (which are confidential by default) are listed. A trusted client has the Confidential (Trusted) client type.

• **View All Untrusted Clients:** From the drop-down menu, select Show: Untrusted Clients. All untrusted clients (which are confidential by default) are listed below. An untrusted client has the Confidential client type.

• **View All User-Defined Clients:** From the drop-down menu, select Show: User Defined Clients. A user-defined client is one that’s created by an identity domain administrator. All user-defined clients, trusted and untrusted, appear.

• **View All Infrastructure Clients:** From the drop-down menu, select Show: Infrastructure Clients. An infrastructure client is one that’s automatically created by the Service Deployment Integration (SDI) team (using a script). No infrastructure clients are available in the example.
Updating OAuth Clients

Modify the properties of an OAuth client on the Modify Clients page.

The properties of an OAuth client such as the description, can be modified. A previously untrusted OAuth client can be changed to be a trusted OAuth client, so that the client can now obtain a user token and propagate an end-user identity. APIs can be granted to or revoked from existing clients. The client certificates can be reloaded.

Modifying Properties of an Untrusted OAuth Client

You can modify the following client properties of an existing untrusted client:

- Edit the description. The client name can’t be changed.
- Add new resources or remove existing resources.
- Change the untrusted client to a trusted client by selecting the Trusted check box. To change the client from untrusted to trusted, you must upload a certificate.
- Upload a new certificate or use an existing certificate.

Modifying Properties of a Trusted OAuth Client

You can modify the following properties of an existing trusted client:
• Edit the description. The client name cannot be changed.
• Add new resources or remove existing resources.
• Update the client’s public key (reload the client certificate).
• Change the trusted client to an untrusted client by clearing the Trusted check box.
• Upload a new certificate.

Overwriting the Client Profile:
The existing client profile is overwritten based on the changes made to the client. If the api_path is changed (added or removed), then the corresponding Audience attribute of the client changes.

Managing Client Certificates
Trusted and untrusted OAuth clients can use certificates. Client certificates can be managed from the UI console.

You can fetch a root certificate authority or an OAuth signing certificate from the OAuth Administration page.

Download Existing Certificates
To download an OAuth signing certificate or a root certificate authority:
1. Log in to the user’s identity domain in Oracle Cloud, and then click Users.

2. Go to the OAuth Administration page.

3. In the Manage Certificates section, click Download.

4. To save the OAuth signing certificate to a local folder, click OAuth Signing Certificate.

5. To save the root certificate authority to a local folder, click Root CA Certificate.

Deleting OAuth Clients

An OAuth client can be deleted at any time.

If either of the following is true, then you must delete an OAuth client and register a new one:

- The client’s credentials (password) have been compromised.
- You want to change certain attributes of the client that can’t be edited through the UI. Examples of such attributes include the client identifier and the client password.

When an OAuth client is removed, the tokens supplied earlier for the client can no longer be validated. Also, no new access tokens are provided for the client. Persistent expired tokens are removed from the database periodically.

1. To delete a client, click the Action menu icon to the right of the client that you want to remove. Select Remove.

2. The Remove Client confirmation dialog box appears. To delete the client, click Remove.
Troubleshooting OAuth

If OAuth isn’t working properly, you must troubleshoot it.

Before contacting Oracle Support:

1. Capture the protocol message requests and responses by using a proxy or logging the messages.

2. To identify exceptions or deviations, review the protocol messages.

3. Ensure that the credentials being used are valid and that the OAuth client hasn’t been disabled.

4. Access the MyOracle Support knowledge base to review known problems and determine whether the problem that you’re having has been experienced by others.

5. If you’ve performed all these troubleshooting steps and you believe that the problem is because of Oracle Cloud, then contact Oracle Support Services. Be ready to provide all your information, including diagnostic data such as protocol message logs.
This chapter describes when and how to use authorization grants. An authorization grant is a credential representing the resource owner’s authorization (to access its protected resources) used by the client to obtain an access token. The OAuth 2.0 core specification describes different authorization grants. Oracle Cloud supports the following grant types: resource owner password credentials, client credentials, and user assertion.

### OAuth Endpoints

OAuth endpoints are the URLs you use to make OAuth authentication requests. The OAuth server exposes a token endpoint that you use to obtain an access token. The format of the OAuth token endpoint URL is `https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/tokens`. For example, if the tenant is tenant1, the data center is Chicago, and the data center code is us2, then the OAuth token endpoint URL is: `https://tenant1.identity.us2.oraclecloud.com/oauth/tokens`.

### Topics:

- **How Do I Use Authorization Grants**
- **Resource Owner Password Credentials Workflow**
- **Step by Step Workflow of the Resource Owner Password Credentials Grant**
- **Using REST API Calls for the Resource Owner Password Credentials Grant**
- **Client Credentials Workflow**
- **Step by Step Workflow of the Client Credentials Grant**
- **Using REST API Calls for the Client Credentials Grant**
- **User Assertion Workflow**
- **Using REST API Calls for the User Assertion Grant**
- **Successful Authorization**
- **Authorization Error**

### How Do I Use Authorization Grants?

In Oracle Cloud, an OAuth client makes a Representational State Transfer (REST) API call to access a protected service. As an administrator, when you secure cloud services, follow the guidelines to decide which type of authorization grant is suitable. An authorization grant is a credential representing the resource owner’s authorization to...
access its protected resource. The authorization grant is used by the OAuth client to obtain an access token.

Oracle Cloud supports the following grant types:

- Resource owner password credentials grant
- Client credentials grant
- User assertion grant

**Guidelines to Choose an OAuth Workflow**

Use the following guidelines to determine which workflow or grant type to use:

Use the resource owner password credentials workflow when:

- The OAuth clients are confidential clients.
- The resource owner has a trust relationship with the client.
- The client application doesn’t need to store the credentials of the resource owner within the application or on the device.

Using the resource owner password credentials workflow, there are two ways to request an access token:

- By sending a simple client header in the token request in addition to the user’s credentials. If you don’t want to use a client assertion, but just the user’s credentials with a basic client header, then see Obtain an Access Token by Using the User Credentials Without a Client Assertion.
- By using a client assertion in addition to the user’s credentials. To use the client token and the user’s credentials to request an access token, see Obtain an Access Token by Using the User Credentials and a JWT Client Assertion.

Using the client credential workflow, there are two ways to request an access token:

- By using a simple client header. If you want to use a simple client header, then see Obtain an Access Token by Using the Client Authorization Header.
- By using a client assertion. After you’ve a self-assigned client assertion, see Obtain an Access Token by Using a Self-Signed Client Assertion to request an access token.

Use the user assertion workflow when:

- The OAuth clients are confidential clients.
- The user’s credentials should never be accessible to the client application.
- The OAuth clients are trusted to assert a user identity on behalf of the user.

Using the user assertion workflow, there are two ways to request an access token:

- By using a user assertion with a simple client header. If you want to use a simple client header with a self-signed user assertion, then see Obtain an Access Token by Using a Self-Signed User Assertion and the Client Credentials to request an access token.
- By using a user assertion with a client assertion. If you do not have a user token, you first need to build one. If you want to use a client assertion, but don’t have a
client token yet, then build your own assertion. After you have a client token and a self-assigned user assertion, see Obtain an Access Token by Using a Self-Signed User Assertion and a Client Assertion to request an access token.

Resource Owner Password Credentials Workflow

When using the resource owner password credentials grant, the user provides the credentials (user name and password) directly to the application. The application then uses the credentials to obtain an access token from the OAuth token service.

The resource owner password credentials grant is a grant workflow where the client application, together with its client identifier and secret, sends the user name and password in exchange for an access token. Instead of the user having to log in and approve the authorization request in a web interface, the user can enter the user name and password in the client application UI directly. This workflow has different security properties than other OAuth workflows. The primary difference is that the user’s password is accessible to the application. This requires a strong trust of the application by the user.

Security Properties

If the resource owner password credentials workflow is used, the application needs access to the user’s credentials only once, on first use, when the credentials are exchanged for an access token. This means that there is no requirement for the application to store these credentials within the application or on the device, and revoking access is easy as well.

Key Characteristics of the Resource Owner Password Credentials Grant

The resource owner password credentials grant:

• Is used with confidential clients.

• Uses the user name and password of the resource owner.

• Isn’t redirection-based; it takes a request only from the client application to the authorization server, and the user isn’t redirected between interfaces to authorize the request.

Step-by-Step Workflow of the Resource Owner Password Credentials Grant

The resource owner password credentials grant workflow allows for the exchanging of the user name and password of a user for an access token.

When using the resource owner password credentials grant, the user provides the credentials (user name and password) directly to the application. The application then uses the credentials to obtain an access token from the service.

Workflow of Resource Owner Password Credentials Grant

1. **Obtain user credentials**: The user provides the credentials to the application. The user credentials are the resource owner’s user name and password.

2. **Request an access token**: The user credentials are exchanged for an access token. The client application makes a request to the authorization server and includes the user’s credentials and either the client credentials or a client assertion. The client application can use an already-generated client assertion or build a new assertion.
Obtain an access token by using different scenarios in the resource owner password credentials workflow:

- **Obtaining an Access Token by Using the User Credentials Without a Client Assertion**
- **Obtaining an Access Token by Using User Credentials and JWT Client Assertion**

3. **Receive an access token from the authorization server:** The authorization server authenticates the client based on the client identifier and secret, determines whether it’s authorized for making this request, and verifies that the resource owner credentials and other parameters are supplied. If everything is verified successfully, then the authorization server returns an access token in the response. This is described in Successful Authorization.

   If the authorization request fails for any reason, then the authorization server returns a response containing the information about the error. This is described in Authorization Error.

4. **Use the access token to make a service request:** The OAuth client makes a REST API call to the resource server using the access token to access the protected resource.

5. **Send a response:** The resource server sends a response to the service request.

6. **Grant access to the resource:** The enduser or service gets access to the protected resource.

### Using REST API Calls for the Resource Owner Password Credentials Grant

Get an access token by using different scenarios in the resource owner password credentials workflow. These scenarios include using the user’s credentials with either the client credentials or a client assertion.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain an access token by using the user’s credentials and the client credentials.</td>
<td>Get the access token by providing the resource owner’s user name and password and the client credentials.</td>
<td>Obtaining an Access Token by Using the User Credentials Without a Client Assertion</td>
</tr>
<tr>
<td>Obtain an access token by using the user’s credentials and a client assertion.</td>
<td>Get the access token by providing the resource owner’s user name and password and a client assertion.</td>
<td>Obtaining an Access Token by Using the User Credentials and a JWT Client Assertion</td>
</tr>
</tbody>
</table>

**Obtaining an Access Token by Using the User Credentials Without a Client Assertion**

Using the resource owner password credentials workflow, the OAuth client can obtain an access token by providing the user’s credentials (that is the user name and password).

This workflow has a resource owner request that uses the user identifier and password of the resource owner.
Obtaining an Access Token by Using the User Credentials

The resource owner password credentials grant workflow, allows you to obtain an access token by using the user’s credentials.

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Authorization**: Basic: The basic authorization header. The client identifier and client secret of the client application is base64-encoded and sent in the header. For example, the Authorization header has the value `base64encoded(client_id:client_secret)`.
- **Content-Type**: The type of content that’s sent in the request. It’s a URL-encoded application.
- **Request**: The type of request that’s sent. Here, it’s a POST request to obtain an access token. This is followed by the authorization server URL which provides tokens.
- **grant_type**: The grant type used to obtain the token. In the example that follows, it’s resource owner password credentials grant. The value of `password` is given for this grant type.
- **username**: The name of the user.
- **password**: The name of the password.
- **scope**: The limit of a particular scope for an access token.

To obtain an access token that contains the user and client credentials, use the following `curl` command:

```
curl -i -H 'X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant125' 
-H 'Authorization: Basic MzAzYTI0OTItZDY0Zi00ZTA0LWI3OGYtYjQzMzAwNDCzMTTiOl155k5NSkdFpc02q0xWZVzdVMz' 
-H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8' 
--request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/tokens 
-d 'grant_type=password 
&username=tenantAdminUser 
&password=Fusionapps1 
&scope=http://www.example.com'
```

The output of the `curl` command is:

```
{
  "expires_in":3600,
  "token_type":"Bearer",
  "access_token":"
```
The JavaScript Object Notation web token (JWT) obtained can be decoded and the claims in the access token can be viewed as follows:

**Access Token:**

```
{
  alg: "RS256",
  typ: "JWT",
  x5t: "Wwrepu2dasaIpGR-AlVpHkUB6Jg",
  kid: "OAuthTestTenant125.cert"
}
```

```
{
  sub: "tenantAdminUser",
  oracle.oauth.user_origin_id_type: "LDAP_UID",
  oracle.oauth.user_origin_id: "tenantAdminUser",
  iss: "OAuthTestTenant125",
  oracle.oauth.svc_p_n: "OAuthTestTenant125ServiceProfile",
  iat: 1425423619000,
  oracle.oauth.prn.id_type: "LDAP_UID",
  oracle.oauth.tk_context: "resource_access_tk",
  exp: 1425427219000,
  aud: ["http://www.example.com"]
}
```

Audience and scope claims in the output:

The audience claim in an access token always contains the API path of the resource. The oracle.oauth.scope claim contains the valid API path with the scope in the response. In the example above, the incoming request has a scope of `http://www.example.com`. The client audience configuration also has a value of `http://www.example.com`. The OAuth token service validates the incoming request scope.
Obtaining an Access Token by Using the User Credentials and a JWT Client Assertion

The OAuth client can request an access token by providing the user’s credentials (that is, the user name and password) and a JSON web token (JWT) client assertion.

This workflow has a resource owner request that uses the user identifier and password of the resource owner, and a JWT client assertion generated by a third party. When using the resource owner password credentials grant workflow, you can obtain an access token by providing the user’s credentials and a client assertion. See Step-by-Step Workflow of the Client Credentials Grant to identify the claims that need to be part of the client assertion.

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Content-Type**: The type of content that’s sent in the request. It’s a URL-encoded application.
- **Request**: The type of request that’s sent. The example uses a POST request to obtain an access token. This is followed by the authorization server URL, which provides tokens.
- **grant_type**: The grant type used to obtain the token. In the example that follows, the grant type is resource owner password credentials grant. The value of password is given for this grant type.
- **username**: The name of the user.
- **password**: The name of the password.
- **client_assertion_type**: The type of client assertion. In Oracle Cloud, it is jwt_bearer.
- **client_assertion**: The value of the client token obtained.
- **scope**: The limit of a particular scope for an access token.

The client credentials are available in the form of an already-generated JWT client assertion. This is sent along with the user’s credentials to obtain an access token.

To obtain an access token, use the following cURL command:

```bash
curl -i -H "X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant125" -H "Authorization: Basic MzAzYTI0OTItZDY0Zi00ZTA0LWI3OGYtYjQzMzAwNDBcZmtMTJjOTI5dS5wSkxcZUxvWzVzdGNvZz" -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8" --request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/tokens -d grant_type=password &username=tenantAdminUser &password=Fusionapps1 &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer &client_assertion=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJvcmFkbGVyIjoiaHR0cHM6Ly9jcmVhdC5vZmwrIiwicGxhbkI6dHJ1ZSwicmVzcGVyIjp7InVzZXQiOiJ3cmVfbGlkNGxwZDMzNWU0NjY4NzUxMjYiLCJpZCI6IjBwbGxvYXRlcyIsInN1YiI6IjUzNTgyMDI1MDY2IiwicmVzcGVyIjpbIlVTRVJFQURJIl0.eyJ0eXBlIjoiRl9TRVJFQURJIl0&scope=
```

Using REST API Calls for the Resource Owner Password Credentials Grant

Securing Authorizations in Oracle Cloud 3-7
The output of the cURL command is:

```json
{
   "expires_in":3600,
   "token_type":"Bearer",
   "access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0ZW5hbnRBZG1pblVzZXIiLCJvcmFjbGUub2F1dGgudXNlcl9vcmlnaW5faWRfdHlwZSI6IkxEQVBfVUlEIiwib3JhY2xlLm9hdXRoLnBybi5pZF90eXBlIjoiTERBUFRlc3RUZWNhbnQxMjVTZXJ2aWNlUHJvZmlsZSIsImlhdCI6MTQyNTQyODA2ODAwMCwib3JhY2xlLm9hdXRoLmNsaWVudF9vcmlnaW5faWQiOiIzMDNhMjQ5Mi1kNjRmLTRlMDQtYjc4Zi1iNDMzMDA0NzMxMmIiLCJvcmFjbGUub2F1dGguc2NvcGUiOiJodHRwOi8vd3d3LmV4YW1wbGUuY29tIiwidXNlci50ZW5hbnQubmFtZSI6Ik9BdXRoVGVzdFRlbmFudDEyNSIsIm9yYWNsZS5vYXV0aC5pZF9kX2lkIjoiMTM0NjM2NzUxMzgzMDI1NjYiLCJ1c2VyLnRlbmFudC5uYW1lIjoiT0F1dGhUZXN0VGVuYW50MTI1Iiwib3JhY2xlLm9hdXRoLnBybi5pZF90eXBlIjoiY2Fsb3JhcGFnZS04MiIsImlhdCI6MTQyNTQyODA2ODAwMCwiYXVkIjpbImh0dHA6Ly93d3cuZXhhbXBsZS5jb20iXSwicHJuIjoidGVuYW50QWRtaW5Vc2VyIiwianRpIjoiMGZmNTM4NzQtZTJhY2U0OTYwLTMzMDI1NzI5MiJ9
}
```

The JSON web token (JWT) obtained can be decoded and the claims in the access token can be viewed as follows:

**Access token:**

```json
```

3-8 Oracle Cloud Administering Oracle Cloud Identity Management
Audience and scope claims in the output:
The audience claim in an access token contains the API path of the resource. The `oracle.oauth.scope` claim contains the valid API path with the scope in the response. In the prior example, the incoming request has a scope of `http://www.example.com`. The client audience configuration also has a value of `http://www.example.com::*`. The OAuth token service validates the incoming request scope with the value in the client audience configuration values. Because this is a valid request, the OAuth token service sends a valid access token in the response. In this case, the audience claim has a value of `http://www.example.com` and the scope has a value of `http://www.example.com`.

**Client Credentials Grant Workflow**

Use the client credentials grant when the client itself owns the data and doesn’t need delegated access from a resource owner, or the delegated access has already been granted to the application outside of a typical OAuth workflow.

The client credentials grant provides a specific grant flow in which the resource owner (that is, the user) is not involved. When using this grant, the client application requests an access token only with its own credentials (the identifier and secret) or an assertion, and uses the access token on behalf of the client application itself. This grant flow is best-suited when a service provider wants to provide some API methods that are to be used by the client application in general, instead of methods that apply to a certain resource owner, for example, API methods for maintenance. This way of using an API is also referred to as *userless* access.

**Security Properties**

Depending on the use case for which you want to use the client credentials grant flow, a single set of credentials for a client could provide access to a large amount of data. The more data a single set of credentials has access to, the greater the risk if the
credentials become compromised. It’s critical that the credentials used to authenticate
the client are kept confidential. Ideally, these credentials would also be rotated
regularly.

**Key Characteristics of the Client Credentials Grant Type**

- It’s used by confidential clients.
- The flow is not redirection-based.
- It’s useful in cases where the client application communicates with the service
  provider directly and not on behalf of a resource owner.
- The resource owner isn’t part of the flow.

**Step-by-Step Workflow of the Client Credentials Grant**

The client credentials grant workflow relies on the client being able to properly
authenticate with the authorization server and the client’s authentication credentials
remaining confidential.

When using the client credentials grant workflow, only the client details are used for
authentication and there is no resource owner.

**Workflow of the Client Credentials Grant**

1. **Request an access token:** The client credentials are exchanged for an access token.
   The client application makes a request to the authorization server, including the
   HTTP basic authentication header and optionally a client assertion. The client
   application can use an already-generated client assertion or build a new assertion.
   The client assertion is a standard JSON web token (JWT), to be signed by a trusted
   or confidential client using its private key. Verify that the following claims are
   part of the client assertion:

   **Header**
   
   ```
   {
   "alg": "RS256",
   "typ": "JWT",
   "x5t": "<X5t of the certificate>"
   }
   ```

   **Body**
   
   ```
   {
   "exp": <Expiry Time in seconds>,
   "sub": "<clientId>",
   "aud": [ "oauth.idm.oracle.com" ],
   "iss": "<clientId>",
   "oracle.oauth.sub.id_type": "ClientID",
   "prn": "<clientId>",
   "jti": "<Globally Unique Id representing the token>",
   "iat": <Issued at Time in seconds>,
   "user.tenant.name": "<tenantname>",
   "oracle.oauth.prn.id_type": "ClientID"
   }
   ```

   Obtain an access token by using different scenarios in the client credentials
   workflow:

   - **Obtaining an Access Token Using the Client Authorization Header**
- **Obtaining an Access Token Using a Self-Signed Client Assertion**

2. **Receive an access token from the authorization server**: The authorization server authenticates the client based on the authorization header or assertion sent and makes a response. If the client is authenticated and the parameters supplied are valid, then the client gets an access token as the response. This is described in **Successful Authorization**

   If the authorization request fails for any reason, then the authorization server returns a response containing information about the error. This is described in **Authorization Error**

3. **Use the access token to make a service request**: The OAuth client makes a REST API call to the resource server using the access token to access the protected resource.

4. **Send a response**: The OAuth resource server sends a response to the service request.

5. **Grant access to the resource**: The client (enduser) or service gets access to the protected resource.

### Using REST API Calls for the Client Credentials Grant

Get an access token by using the client authorization header, or a self-signed client assertion.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain an access token by using a client authorization header.</td>
<td>The client application sends an authorization basic header to obtain the access token.</td>
<td>Obtaining an Access Token by Using a Client Authorization Header</td>
</tr>
<tr>
<td>Obtain an access token by using a self-signed client assertion.</td>
<td>The client application uses a self-signed client assertion as part of the request to obtain the access token.</td>
<td>Obtaining an Access Token by Using a Self-Signed Client Assertion</td>
</tr>
</tbody>
</table>

### Obtaining an Access Token by Using a Client Authorization Header

The client credentials workflow allows the client application to obtain an access token by using the basic authorization header.

In Oracle Cloud, all OAuth clients are confidential by default and so their credentials (client_id and password) are never exposed. The client_id and password credentials are encoded and sent in the basic authorization header. The format used to obtain the header value is `base64encoded(client_id:password)`.

**Parameters used in the access token request:**

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Authorization**: **Basic**: The basic authorization header. The client identifier and client secret of the client application are base64-encoded and sent in the header. For example, the authorization header has the value of `base64encoded(client_id:password)`.
- **Content-Type**: The type of content that’s sent in the request. It is a URL-encoded application.
• **Request:** The type of request that's sent. In the example that follows, a POST request is used to obtain an access token. This is followed by the authorization server URL, which provides tokens.

• **grant_type:** The grant type used to obtain the token. In the example that follows, the grant type is client credentials. The value of `client_credentials` is given for this grant type.

• **scope:** The limit of a particular scope for an access token.

The client identifier and password are encoded and sent in the basic authorization header. This is sent to obtain an access token.

To obtain an access token by providing the client credentials, use the following cURL command:

```
curl -i -H 'X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant125'
-H 'Authorization: Basic MzAzYTI0OTItZDY0Zi00ZTA0LWI3OGYtYjQzMzAwNDczMjI1O1wiLmNvbS5jaXJlZi5lbmNvZGluZw=='
-H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8'
--request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/tokens
-d 'grant_type=client_credentials &scope=http://www.example.com'
```

The output of the cURL command is:

```
{
"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
```

The JSON web token (JWT) obtained can be decoded and the claims in the access token can be viewed as follows:

```
Access token:
```

The output can be decoded and the claims in the access token can be viewed as follows:

```
Access token:
```

The JSON web token (JWT) obtained can be decoded and the claims in the access token can be viewed as follows:

**Access token:**
Using REST API Calls for the Client Credentials Grant

Obtaining an Access Token by Using a Self-Signed Client Assertion

The client application uses a self-signed client assertion as part of the request to obtain the access token.

Instead of sending the client credentials, send the client assertion as part of the request for greater security. In Oracle Cloud, all OAuth clients are confidential by default and so their credentials (client_id and password) are never exposed directly. A client assertion is generated before requesting an access token. See Step-by-Step Workflow of the Client Credentials Grant to identify the claims that need to be part of the client assertion.

In the client credentials workflow, you obtain an access token by using a client assertion.

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Content-Type**: The type of content that’s sent in the request. It is a URL-encoded application.
• **Request:** The type of request that’s sent. In the example that follows, a POST request is used to obtain an access token. This is followed by the authorization server URL, which provides tokens.

• **grant_type:** The grant type used to obtain the token. In the example that follows, the grant type is client credentials. The value of `client_credentials` is given for this grant type.

• **scope:** The limit of a particular scope for an access token.

• **client_assertion_type:** This specifies the type of client assertion that’s passed. In Oracle Cloud, it’s `jwt_bearer`.

• **client_assertion:** The value of the client token obtained.

The client credentials are available in the form of a self-signed JSON web token (JWT) client assertion. This is sent to obtain an access token.

To obtain an access token by using a client assertion, use the following **cURL** command:

```
curl -i -H "X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant150"
   -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
   --request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/tokens
   -d 'grant_type=client_credentials
   &client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
   &client_assertion=eyJ4NXQiOiJyb2NFQ2NaVD1heG5FDwpQMVpVQVo3ZGNyTmM1LCJraWQiOiJX0FNX0dPT0
Q1IJCj0eXai0iJKVQiIc1hbGci0iJSUuUxM1J9.eyJzdWIiOiJhbWQ1Mz11Yy15MDOmLzTQzZDYtOOGQ3Ny1hOD
YzYjhcuntezJcpc3M0iJX0FNX0dDPTQ1Ic1hvcmFjbgUub21jLnmRv2uLnr5GcGU0iJDTEI5T1R0U0tFTIIs
lmV4c
CI6NDU3OTI2NjA3NyVicil0j1YoYTJkNTM5ZWNMT0TA2Zi10M2ZLThkNzctYgt2M1I4ZDY3U0lwiawWFO1j
oxNDI
1njY2Jc3LCJvcmpjbgUub21jLnmRv2uLnr5Zy1jc1g4oiJ1aiAQ9YTJkNTM5ZWNMT0TA2Zi10M2ZLThkNzct
Yrg2M
214DZ3YU0LCjbj10ZM0X2IgdGvzDvGyLCBvdT10ZX0NCBvPW9yYWNsZSwgc3Q9Y2FsaWQ5YVwzYz
1lcYJ
9.0.MHC90c6fuaZGMrKAbmdnb36b-
nhk164Wq7Abyba3V3asHRM3.bq2Y_quM1A9H585SVhipm01R88N8NTINwstjzh
H629bATX6gJymSbyy8K7v35dKZ-awaCON90Ti2aPdApFkTiux9r0-1vZSWvMbwx6ZPSIUH58WvrpL58
&scope=http://www.example.com
```

The output of the **cURL** command is:

```
{
  "expires_in":3600,
  "token_type":"Bearer",
  "access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhNWQ1Mz11Yy15MDOmLzTQzZDYtOOGQ3Ny1hOD
YzYjahntezJcpc3M0iJX0FNX0dDPTQ1Ic1hvcmFjbgUub21jLnmRv2uLnr5GcGU0iJDTEI5T1R0U0tFTIIs
lmV4c
CI6NDU3OTI2NjA3NyVicil0j1YoYTJkNTM5ZWNMT0TA2Zi10M2ZLThkNzctYgt2M1I4ZDY3U0lwiawWFO1j
oxNDI
1njY2Jc3LCJvcmpjbgUub21jLnmRv2uLnr5Zy1jc1g4oiJ1aiAQ9YTJkNTM5ZWNMT0TA2Zi10M2ZLThkNzct
Yrg2M
214DZ3YU0LCjbj10ZM0X2IgdGvzDvGyLCBvdT10ZX0NCBvPW9yYWNsZSwgc3Q9Y2FsaWQ5YVwzYz
1lcYJ
9.0.MHC90c6fuaZGMrKAbmdnb36b-
nhk164Wq7Abyba3V3asHRM3.bq2Y_quM1A9H585SVhipm01R88N8NTINwstjzh
H629bATX6gJymSbyy8K7v35dKZ-awaCON90Ti2aPdApFkTiux9r0-1vZSWvMbwx6ZPSIUH58WvrpL58
&scope=http://www.example.com
}
```
The JWT obtained can be decoded, and the claims in the access token can be viewed as follows:

**Access token:**

```
{
  alg: "RS256",
  typ: "JWT",
  x5t: "kh6XrTN6WjzvhH8Lk6sKieP5hw",
  kid: "OAuthTestTenant150.cert"
}
```

```
{  
  sub: "a5d539ec-906f-43d6-8d77-a863b8d37ce4",
  iss: "OAuthTestTenant150",
  oracle.oauth.svc_p_n: "OAuthTestTenant150ServiceProfile",
  iat: 142566598000,
  oracle.oauth.prn.id_type: "ClientID",
  exp: 1425670198000,
  oracle.oauth.tk_context: "resource_access_tk",
  prn: "a5d539ec-906f-43d6-8d77-a863b8d37ce4",
  jti: "f90711e9-1981-48c2-8c09-d91534c0dbca",
  oracle.oauth.client_origin_id: "a5d539ec-906f-43d6-8d77-a863b8d37ce4",
  oracle.oauth.scope: "http://www.example.com",
  user.tenant.name: "OAuthTestTenant150",
  oracle.oauth.id_d_id: "30167455953447081"
}
```

**[signature]**

**Audience and scope claims in the output:**

The audience claim in an access token contains the API path of the resource. The `oracle.oauth.scope` claim contains the valid API path with the scope in the response. In the prior example, the incoming request has a scope of `http://www.example.com`. The client audience configuration has a value of `http://www.example.com::*`. The OAuth token service validates the incoming request scope with the value found in the client audience configuration. Because this is a valid request, the OAuth token service sends a valid access token in the response. In this case, the audience claim has a value of `http://www.example.com` and the scope has a value of `http://www.example.com`.

Securing Authorizations in Oracle Cloud 3-15
User Assertion Workflow

In this workflow the user provides the user assertion to obtain an access token from the OAuth token service.

The user assertion grant describes a flow where the client application, together with its client identifier and password, sends the user assertion in exchange for an access token. This flow has different security properties than the other OAuth flows. The primary difference is that the user’s credentials are never accessible to the application.

Key Characteristics of the User Assertion Workflow

The user assertion workflow:

- It’s used with confidential clients.
- It uses the assertion of the Resource Owner.
- It isn’t redirection-based; it takes a request only from the client application to the authorization server, and the user is not redirected between interfaces to authorize the request.

Workflow of the User Assertion Grant

1. Obtain user assertion: The user provides the assertion. The client application can use an already-generated user assertion or build a new assertion. The user assertion is a standard JSON web token (JWT), to be signed by a trusted client using its private key. Verify that the following claims are part of the user assertion:

   **Header**
   
   ```
   {
     "alg": "RS256",
     "typ": "JWT",
     "x5t": "<X5t of the certificate>"
   }
   ```

   **Body**
   
   ```
   {
     "exp": <Expiry Time in seconds>,
     "sub": "<username like john.doe@example.com>",
     "aud": [ "oauth.idm.oracle.com" ],
     "iss": "<clientid>",
     "oracle.oauth.sub.id_type": "LDAP_UID",
     "prn": "<username like john.doe@example.com>",
     "jti": "<Globally Unique Id representing the token>",
     "iat": <Issued at Time in seconds>,
     "user.tenant.name": "<tenantname>",
     "oracle.oauth.prn.id_type": "LDAP_UID"
   }
   ```

2. Request an access token: The user assertion is exchanged for an access token. The client application makes a request to the authorization server, including the user’s assertion and either the client’s credentials or the client assertion. The client application can use an already generated client assertion or build a new assertion.
Note:

In the regular flow the access token’s expiry claim is obtained from the configuration and the expiry time of the access token is by default 1 hour. However, in case of using the self-signed user assertion and client credentials flow, the expiry time of the access token can be modified to a value up to 90 days. The OAuth Server looks for the \( \text{exp} \) claim in the user assertion to determine the expiry claim of the resulting access token.

Obtain an access token by using different scenarios in the user assertion workflow:

- Obtaining an Access Token by Using a Self-Signed User Assertion and the Client Credentials
- Obtaining an Access Token by Using a Self-Signed User Assertion and a Client Assertion

3. Receive an access token from the authorization server: The authorization server authenticates the client based on the client identifier and secret, determines whether it is authorized for making this request, and verifies the user’s assertion and other parameters that are supplied. If everything is verified successfully, then the authorization server returns an access token in response. This is described in Successful Authorization.

If the authorization request fails for any reason, then the authorization server returns a response containing information about the error. This is described in Authorization Error.

4. Use the access token to make a service request: The OAuth client makes a REST API call to the resource server using the access token to access the protected resource.

5. Send a response: The OAuth resource server sends a response to the client application that made the request.

6. Grant access to the resource: The enduser or service gets access to the protected resource.

Using REST API Calls for the User Assertion Grant

In the user assertion flow, the user provides the user assertion to obtain an access token from the OAuth Service. In addition to the user assertion, the client provides an Authorization header, a self-signed client assertion, or a third-party generated client assertion in the access token request.

The table displays the different options to obtain an access token.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain an access token by using a self-signed user assertion and the client credentials.</td>
<td>The OAuth client can request an access token by providing a self-signed user assertion and the client credentials.</td>
<td>Obtaining an Access Token by Using a Self-Signed User Assertion and the Client Credentials</td>
</tr>
</tbody>
</table>
Obtaining an Access Token by Using a Self-Signed User Assertion and the Client Credentials

The OAuth client can request an access token by providing the user assertion and client credentials.

This workflow describes an access token request that uses the self-signed user assertion and a basic client authorization header. This is a more secure workflow than when the resource owner’s credentials (user name and password) are exposed.

The user assertion grant workflow allows you to obtain an access token by using a user assertion and the client credentials that are supplied in the form of a basic authorization header. See User Assertion Workflow to identify the claims that need to be part of the user assertion.

The client application makes a request to the authorization server that includes the HTTP basic authorization header. The basic authorization header is base64encoded(client_id:client_password).

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Content-Type**: The type of content that’s sent in the request. It’s a URL-encoded application.
- **Authorization: Basic**: The basic authorization header. The client id and client secret of the client application are base64-encoded and sent in the header. For example, the authorization header has a value of base64encoded(client_id:client_password).
- **Request**: The type of request that’s sent. In the example that follows, a POST request is used to obtain an access token. This is followed by the authorization server URL which provides tokens.
- **grant_type**: The grant type used to obtain the token. In the example that follows, the grant type is user assertion grant. The value of jwt-bearer is given for this grant type.
- **scope**: The limit of a particular scope for an access token.
- **assertion**: The value of the user token obtained.

The client identifier and password are encoded and sent in the basic authorization header. This is sent along with the self-signed user assertion to obtain an access token.

To obtain an access token by using the user assertion and the client credentials, use the following cURL command:

```
curl -i -H "Content-Type: application/x-www-form-urlencoded; charset=UTF-8" -H "X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant125"
```
The output of the `curl` command is:

```json
{
  "expires_in":3600,
  "token_type":"Bearer",
  "access_token":null,
  "token_url":null,
  "refresh_token":null,
  "token_info":null,
  "scope":null
}
```

The JSON web token (JWT) obtained can be decoded, and the claims in the access token can be viewed as follows:

**Access Token:**

```json
{
  "alg": "RS256",
  "typ": "JWT",
  "id": "abc123",
  "iss": "https://example.com",
  "iat": 1234567890,
  "exp": 12345678901,
  "aud": "example.com",
  "nbf": 12345678902,
  "jti": "12345678903",
  "sub": "12345678904",
  "scp": ["read", "write"]
}
```
x5t: "Wwrepu2das4PGR-AlVp8KUB6Jg",
kid: "OAuthTestTenant125.cert"
}

{
sub: "tenantAdminUser",
oracle.oauth.user_origin_id_type: "LDAP_UID",
oracle.oauth.user_origin_id: "tenantAdminUser",
iss: "OAuthTestTenant125",
oracle.oauth.svc_p_n: "OAuthTestTenant125ServiceProfile",
iat: 1425424318000,
oracle.oauth.prn_id_type: "LDAP_UID",
oracle.oauth.tk_context: "resource_access_tk",
exp: 1425427918000,
aud: [
  "http://www.example.com"
],
prn: "tenantAdminUser",
jti: "d385cc71-8f18-46a5-9ae4-6ab6f085badb",
oracle.oauth.client_origin_id: "303a2492-d64f-4e04-b78f-b4330047312b",
oracle.oauth.scope: "http://www.example.com",
user.tenant.name: "OAuthTestTenant125",
oracle.oauth.id_d_id: "13463675138302566"
}

Note:
In the regular flow the access token's expiry claim is obtained from the configuration and the expiry time of the access token is by default 1 hour. However for this use case the expiry time of the access token can be modified to a value up to 90 days. The OAuth Server looks for the exp claim in the user assertion to determine the expiry claim of the resulting access token. See User Assertion Workflow to determine the claims a self-signed user assertion should have.

Audience and scope claims in the output:
The audience claim in an access token contains the API path of the resource. The oracle.oauth.scope claim contains the valid API path with the scope in the response. In the prior example, the incoming request has a scope of http://www.example.com. The client audience configuration has a value of http://www.example.com::*. The OAuth token service validates the incoming request scope with the value found in the client audience configuration. Because this is a valid request, the OAuth token service sends a valid access token in the response. In this case, the audience claim has a value of http://www.example.com, and the scope has a value of http://www.example.com.

Obtaining an Access Token by Using a Self-Signed User Assertion and a Client Assertion

The OAuth client can request an access token by providing the user assertion and the client assertion.

This workflow has an access token request that uses a user assertion and a JSON web token (JWT) client assertion that is generated by a third party. This is a more secure workflow than when the resource owner's credentials (user name and password) are exposed.
The user assertion grant workflow allows you to obtain an access token by using a user assertion and a client assertion. See User Assertion Workflow to identify the claims that need to be part of the user assertion. See Step-by-Step Workflow of the Client Credentials Grant to identify the claims that need to be part of the client assertion.

Parameters used in the access token request:

- **X-USER-IDENTITY-DOMAIN-NAME**: The name of the identity domain.
- **Content-Type**: The type of content that’s sent in the request. It is a URL-encoded application.
- **Request**: The type of request that’s sent. In the example that follows, a POST request is used to obtain an access token. This is followed by the URL of the authorization server, which provides tokens.
- **grant_type**: The grant type used to obtain the token. In the example that follows, the grant type is a user assertion. The value of jwt-bearer is given for this grant type.
- **scope**: The limit of a particular scope for an access token.
- **client_assertion_type**: The type of client assertion that’s passed. In Oracle Cloud, it’s jwt_bearer.
- **client_assertion**: The value of the client token obtained.
- **assertion**: The value of the user token obtained.

The client credentials are available in the form of a third-party generated client assertion. This is sent along with a self-signed user assertion to obtain an access token.

To obtain an access token by using a self-signed user assertion and a client assertion, use the following cURL command:

```bash
curl -i -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
-H 'X-USER-IDENTITY-DOMAIN-NAME: OAuthTestTenant125'
--request POST https://<idm-domain>.identity.<data-center>.oraclecloud.com/oauth/tokens
-d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer'
-d 'client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer'
-d 'assertion=eyJraWQiOiJyb2NFQ2NaVDlheG5FdWpQMVVPQVo3ZGNyTmMiLCJraWQiOiJJX0FNX0dPT0QiLCJvcmFjbyI6IjkiLCJleHAiOjQ1Nzk4MTcwMzksImF1bSBpbiI6IjkiLCJpc3MiOiJJX0FNX0dPT0QiLCJvbmxvdyI6IjkiLCJ0aWQiOiJcb3Jlci50cmFjbyJd'.
```

Using REST API Calls for the User Assertion Grant

Securing Authorizations in Oracle Cloud 3-21
The output of the cURL command is:

```
{
  "expires_in":3600,
  "token_type":"Bearer",
  "access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0ZW5hbnRBZG1pblVzIiwiaXNzIjoiY2FsaWUifQ.Xw5zS21g7qDCdS8cW8Ow5wZw4V8dJ79fK1oq71w"
}
```

The JWT obtained can be decoded, and the claims in the access token can be viewed as follows:

**Access token:**

```
{
  
  alg: "RS256",
  typ: "JWT",
  x5t: "Wwrepu2dasaIpGR-AlVpHkUB6Jg",
  kid: "OAuthTestTenant125.cert"
}
```

**Claims:**

```
{  
  "sub": "tenantAdminUser",
  "oracle.oauth.user_origin_id_type": "LDAP_UID",
  "oracle.oauth.user_origin_id": "tenantAdminUser",
  "iss": "OAuthTestTenant125",
  "oracle.oauth.svc_p_n": "OAuthTestTenant125ServiceProfile",
  "iat": "1425424318000",
  "oracle.oauth.prn.id_type": "LDAP_UID",
  "oracle.oauth.tk_context": "resource_access_tk",
  "exp": "1425427918000",
  "aud": [  
```

The JWT obtained can be decoded, and the claims in the access token can be viewed as follows:

**Access token:**

```
{
  
  alg: "RS256",
  typ: "JWT",
  x5t: "Wwrepu2dasaIpGR-AlVpHkUB6Jg",
  kid: "OAuthTestTenant125.cert"
}
```

**Claims:**

```
{  
  "sub": "tenantAdminUser",
  "oracle.oauth.user_origin_id_type": "LDAP_UID",
  "oracle.oauth.user_origin_id": "tenantAdminUser",
  "iss": "OAuthTestTenant125",
  "oracle.oauth.svc_p_n": "OAuthTestTenant125ServiceProfile",
  "iat": "1425424318000",
  "oracle.oauth.prn.id_type": "LDAP_UID",
  "oracle.oauth.tk_context": "resource_access_tk",
  "exp": "1425427918000",
  "aud": [  
```
Note:
In the regular flow the access token’s expiry claim is obtained from the configuration and the expiry time of the access token is by default 1 hour. The OAuth Server looks for the `exp` claim in the user assertion to determine the expiry claim of the resulting access token. However, only if you are using a the self-signed user assertion and client credentials flow, the expiry time of the access token can be modified to a value up to 90 days.

**Audience and scope claims in the output:**

The audience claim in an access token contains the API path of the resource. The `oracle.oauth.scope` claim contains the valid API path with the scope in the response. In the prior example, the incoming request has a scope of `http://www.example.com`. The client audience configuration has a value of `http://www.example.com::*`. The OAuth token service validates the incoming request scope with the value found in the client audience configuration. Because this is a valid request, the OAuth token service sends a valid access token in the response. In this case, the audience claim has a value of `http://www.example.com` and the scope has a value of `http://www.example.com`.

**Successful Authorization**

During an authorization request, if the validations pass successfully, then the authorization server sends a response with an access token.

When the authorization server handles the request from the client, the following occur:

- Validation of the client assertion, also ensuring that the client is authorized to make the request. This is done by using the client certificate that was imported when the client was registered with the authorization server.

- Validation of the user, also ensuring that the user is authorized to make the request. Either the user's credentials or the user assertion is validated.

- Validation of the audience claim in the client profile in the OAuth service (information stored when the client is registered) against the scope in the incoming access token request.

If the validations pass successfully, the authorization server sends a response with an access token.

**Access Token in the Response**

In the body of the response, a JSON (or XML or other) object is included, representing the response, as shown in the following example:
The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that's returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```

The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that’s returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```

The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that’s returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```

The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that’s returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```

The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that’s returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```

The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that’s returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```

The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that’s returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```

The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that’s returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```

The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that’s returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```

The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that’s returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```

The fields that are a part of the response are:

- **expires_in**: An optional and recommended parameter that specifies the lifetime of the access token in seconds. In the prior example, the access token is valid for 1 hour (or 3600 seconds).
- **token_type**: A mandatory parameter that specifies the type of token that’s returned in the response. In the prior example, the token_type is Bearer.
- **access_token**: A mandatory parameter that has the actual access token as its value. This is the access token, in JSON web token (JWT) format, that the client application may store and use later. The token can be decoded to see the various claims in the access token response.

An example of a decoded JWT:

```
{"expires_in":3600,
"token_type":"Bearer",
"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
"}
```
Claims in the Access Token

An access token has a header, and standard and custom claims.

<table>
<thead>
<tr>
<th>Claim Name</th>
<th>Type</th>
<th>Description</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>alg</td>
<td>Header</td>
<td>The algorithm used to sign the token.</td>
<td>RS256</td>
</tr>
<tr>
<td>typ</td>
<td>Header</td>
<td>The classification type of the token. The default value is JWT. This indicates that this is a JSON web token (JWT).</td>
<td>JWT</td>
</tr>
<tr>
<td>x5t</td>
<td>Header</td>
<td>The X.509 certificate thumbprint (x5t) header parameter provides a base64url-encoded SHA-256 thumbprint of the DER encoding of an X.509 certificate that can be used to match a certificate.</td>
<td>_hVX9pXq7pUxkk5ry-8vK8qb8L8</td>
</tr>
<tr>
<td>kid</td>
<td>Header</td>
<td>The key ID (kid) header parameter is a hint indicating which specific key owned by the signer should be used to validate the signature. This allows signers to signal a change of the key to recipients explicitly. Omitting this parameter is equivalent to setting it to an empty string. The interpretation of the contents of the kid parameter is unspecified.</td>
<td>oauth_psrtenantx3.cert</td>
</tr>
<tr>
<td>sub</td>
<td>Standard</td>
<td>The subject (sub) claim identifies the principal that’s the subject of the JWT.</td>
<td><a href="mailto:MyAdmin@oracle1.com">MyAdmin@oracle1.com</a></td>
</tr>
<tr>
<td>prn</td>
<td>Standard</td>
<td>The principal (prn) claim identifies the principal that is the subject of the JWT.</td>
<td><a href="mailto:MyAdmin@oracle1.com">MyAdmin@oracle1.com</a></td>
</tr>
<tr>
<td>iss</td>
<td>Standard</td>
<td>The issuer (iss) claim identifies the principal that supplied the JWT.</td>
<td>oauth_psrtenantx3</td>
</tr>
<tr>
<td>iat</td>
<td>Standard</td>
<td>The issued at (iat) claim identifies the time at which the JWT was supplied.</td>
<td>1429128747000</td>
</tr>
<tr>
<td>exp</td>
<td>Standard</td>
<td>The expiration time (exp) claim identifies the expiration time on or after which the JWT must not be accepted for processing.</td>
<td>1429128747000</td>
</tr>
<tr>
<td>aud</td>
<td>Standard</td>
<td>The audience (aud) claim identifies the recipients for which the JWT is intended.</td>
<td>(a list of audiences)</td>
</tr>
<tr>
<td>Claim</td>
<td>Standard/Custom Claim Description</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>jti</td>
<td>The JWT ID (jti) claim provides a unique identifier for the JWT.</td>
<td>0565e04e-3823-404f-b950-e970ea17f41f</td>
<td></td>
</tr>
<tr>
<td>oracle.oauthsvc_p_n</td>
<td>IDM OAuth service profile name.</td>
<td>oauth_psrtenantx3ServiceProfile</td>
<td></td>
</tr>
<tr>
<td>oracle.oauthprn.id_type</td>
<td>Principal ID type. For user assertion, the value is always LDAP_UID.</td>
<td>LDAP_UID</td>
<td></td>
</tr>
<tr>
<td>oracle.oauthsub.id_type</td>
<td>Subject ID type. For user assertion, the value is always LDAP_UID.</td>
<td>LDAP_UID</td>
<td></td>
</tr>
<tr>
<td>oracle.oauthid_d_id</td>
<td>IDM OAuth server domain ID.</td>
<td>20625897169639935</td>
<td></td>
</tr>
<tr>
<td>oracle.oauthclient_origin_id</td>
<td>Subject ID for client used when user assertion is generated.</td>
<td>4457b326-fe88-4851-baad-b9488895e808</td>
<td></td>
</tr>
<tr>
<td>user.tenant.name</td>
<td>User tenancy for the OAuth token generated by IDM OAuth server.</td>
<td>oauth_psrtenantx3</td>
<td></td>
</tr>
</tbody>
</table>

### Authorization Error

If the access code request fails for any reason, or if one of the request parameters is invalid, then an error occurs.

The authorization server may return a response containing information about the error. This might be in JSON format (or XML or other) and may have the following format:

```json
{
    "error":"invalid_request",
    "error_description":"Username parameter missing"
}
```

### Possible Error Values

The `error` parameter can contain a number of values that describe the nature of the problem that occurred. These values and descriptions are as follows:

- **invalid_request**: The request is missing a parameter or value, a parameter is included multiple times, or a parameter has a malformed name.

- **invalid_client**: The authentication of the client fails. This can happen if authentication parameters are missing (for example, the client identifier and secret) or if the client tries to authenticate using an unsupported method.

- **invalid_grant**: The grant specified is invalid, expired, or revoked, or supplied to another client. For example, some services don’t allow a new access token to be requested until the currently issued token expires.

- **unauthorized_client**: The client was authenticated by the authorization server, but has no authorization to use the requested grant.

- **unsupported_grant_type**: The grant that was requested isn’t supported by the authorization server.
• **server_error**: The Authorization Server encountered an unexpected condition that prevented it from fulfilling the request.

• **temporarily_unavailable**: The authorization server is currently unable to handle the request because of a temporary overloading or maintenance of the server.

• **invalid_scope**: The scope specified in the request isn’t valid, is unknown, or is malformed. If this occurs, then read the developer documentation associated with the service provider to see which scopes are available and which can be used.

**Error Description**

Only the `error` parameter is mandatory. But the optional `error_description` parameter may contain a short message explaining the error, for example, indicating a missing user name.