Oracle® Fusion Middleware

Oracle API Gateway OAuth User Guide

11g Release 2 (11.1.2.3.0)

April 2014

Oracle API Gateway OAuth User Guide, 11g Release 2 (11.1.2.3.0)

30 April 2014

Contents

Introduction to API Gateway OAuth 2.0 server

Overview

OAuth is an open standard for authorization that enables client applications to access server resources on behalf of a specific Resource Owner. OAuth also enables Resource Owners (end users) to authorize limited third-party access to their server resources without sharing their credentials. For example, a Gmail user could allow LinkedIn or Flickr to have access to their list of contacts without sharing their Gmail user name and password.

The Oracle API Gateway can be used as an Authorization Server and as a Resource Server. An Authorization Server issues tokens to client applications on behalf of a Resource Owner for use in authenticating subsequent API calls to the Resource Server. The Resource Server hosts the protected resources, and can accept or respond to protected resource requests using access tokens.

Note

This guide assumes that you are familiar with the terms and concepts described in the OAuth 2.0 Authorization Framework [http://tools.ietf.org/html/rfc6749].

OAuth 2.0 concepts

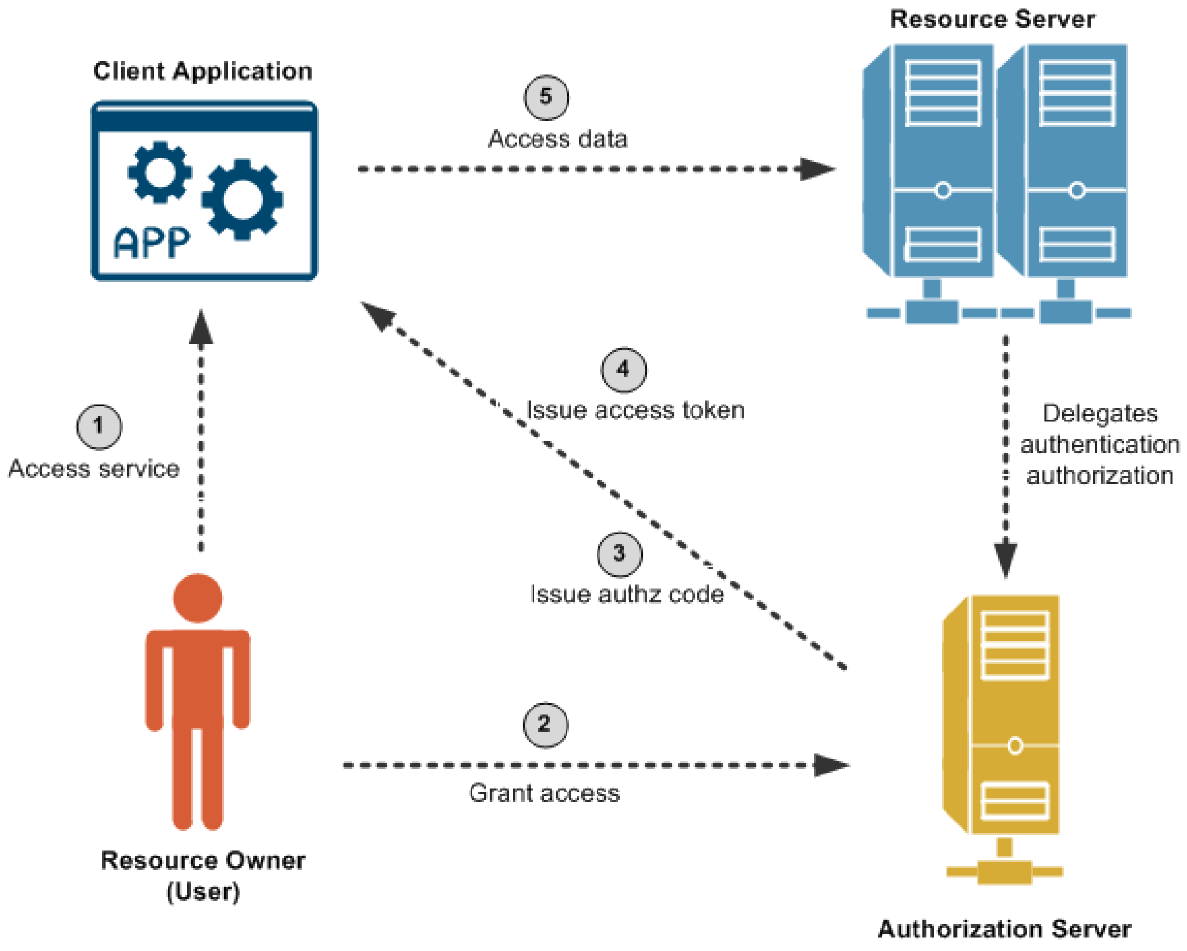
The API Gateway uses the following definitions of basic OAuth 2.0 terms:

- **Resource Owner**:
  An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end user.
- **Resource Server**:
  The server hosting the protected resources, and which is capable of accepting and responding to protected resource requests using access tokens. In this case, the API Gateway acts as a gateway implementing the Resource Server that sits in front of the protected resources.
- **Client Application**:
  A client application making protected requests on behalf of the resource owner and with its authorization.
- **Authorization Server**:
  The server issuing access tokens to the client application after successfully authenticating the Resource Owner and obtaining authorization. In this case, the API Gateway acts both as the Authorization Server and as the Resource Server.
- **Scope**:
  Used to control access to the Resource Owner's data when requested by a client application. You can validate the OAuth scopes in the incoming message against the scopes registered in the API Gateway. An example scope is `userinfo/readonly`.

OAuth 2.0 example workflow

Assume that you are using a image printing website such as Canon to print some of your photos. You also have some photos on your Flickr account that you would like to print. However, you must download all these locally, and then upload them again to the printing website, which is inconvenient. You would like to be able to sign into Flickr from your Canon printing profile, and print your photos directly.

This problem can be solved using the example OAuth 2.0 Web Server flow shown in the following diagram:



Out of band, the Canon printing client application pre-registers with Flickr and obtains a client ID/secret. The client application registers a callback URL to receive the authorization code from Flickr when you, as Resource Owner, allow Canon to access the photos from Flickr. The printing application has also requested access to an API named `/flickr/photos`, which has an OAuth scope of `photos`.

The steps in the diagram are described as follows:

1. You are using a mobile phone and are signed into the Canon image printing website. You click to print Flickr photos. The Canon client app redirects you to the Flickr OAuth Authorization Server. You must already have a Flickr account.
2. You log into your Flickr account, and the Flickr Authorization Server asks you "Do you want to allow the Canon printing app to access your photos?". You click **Yes** to authorize.
3. When successful, the printing app receives an authorization code at the callback URL that was pre-registered out of band.

Note

You have not shared your Flickr username and password with the printing app. At this point, you as Resource Owner are no longer involved in the process.

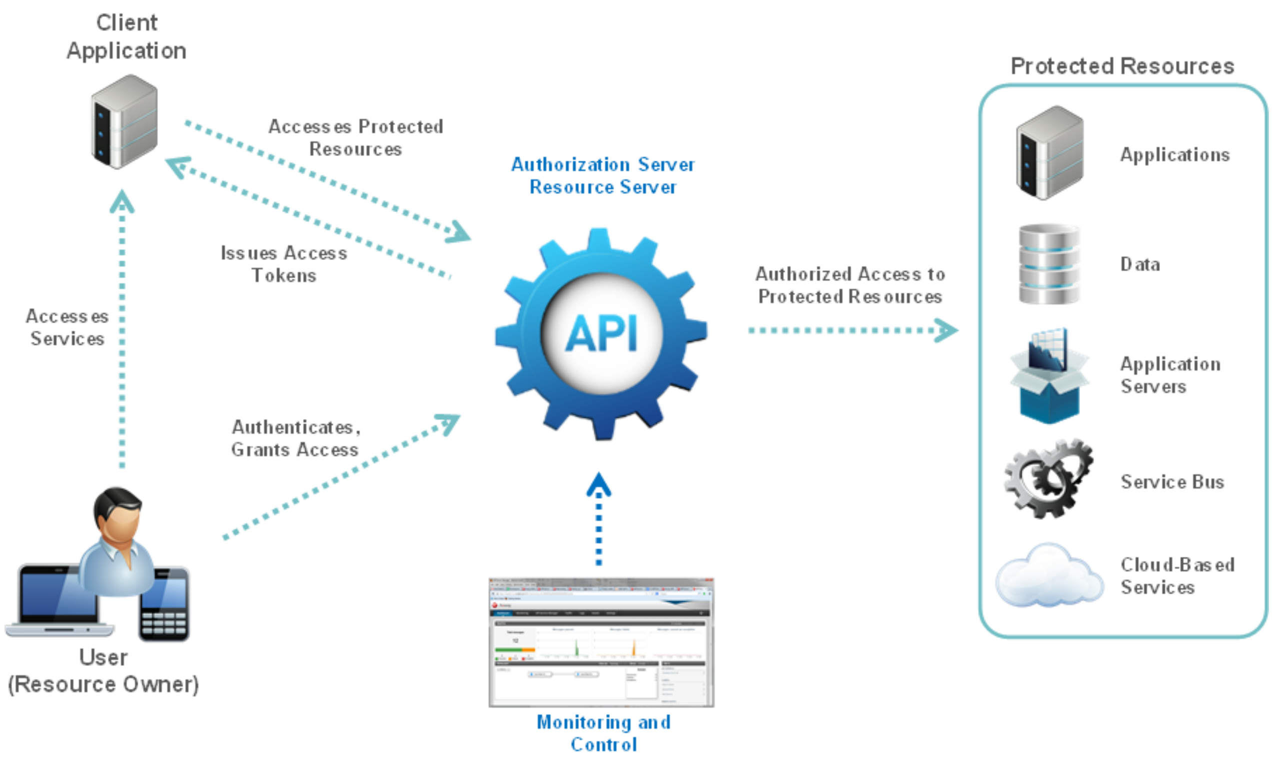
4. The client app gets the authorization code, and must exchange this short-lived code for an access token. The client app sends another request to the Authorization Server, saying it has a code that proves the user has authorized it to access their photos, and now issue the access token to be sent on to the API (Resource Server). The Authorization Server verifies the authorization code and returns an access token.
5. The client app sends the access token to the API (Resource Server), and receives the photos as requested.

API Gateway OAuth features

The API Gateway ships with the following features to support OAuth 2.0:

- Web-based client application registration
- Generation of authorization codes, access tokens, and refresh tokens
- Support for the following OAuth flows:
  - Authorization Code
  - Implicit Grant
  - Resource Owner Password Credentials
  - Client Credentials
  - JWT
  - Refresh Token
  - Revoke Token
  - Token Information Service
- Sample client applications for all supported flows

The following diagram shows the roles of the API Gateway as an OAuth 2.0 Resource Server and Authorization Server:

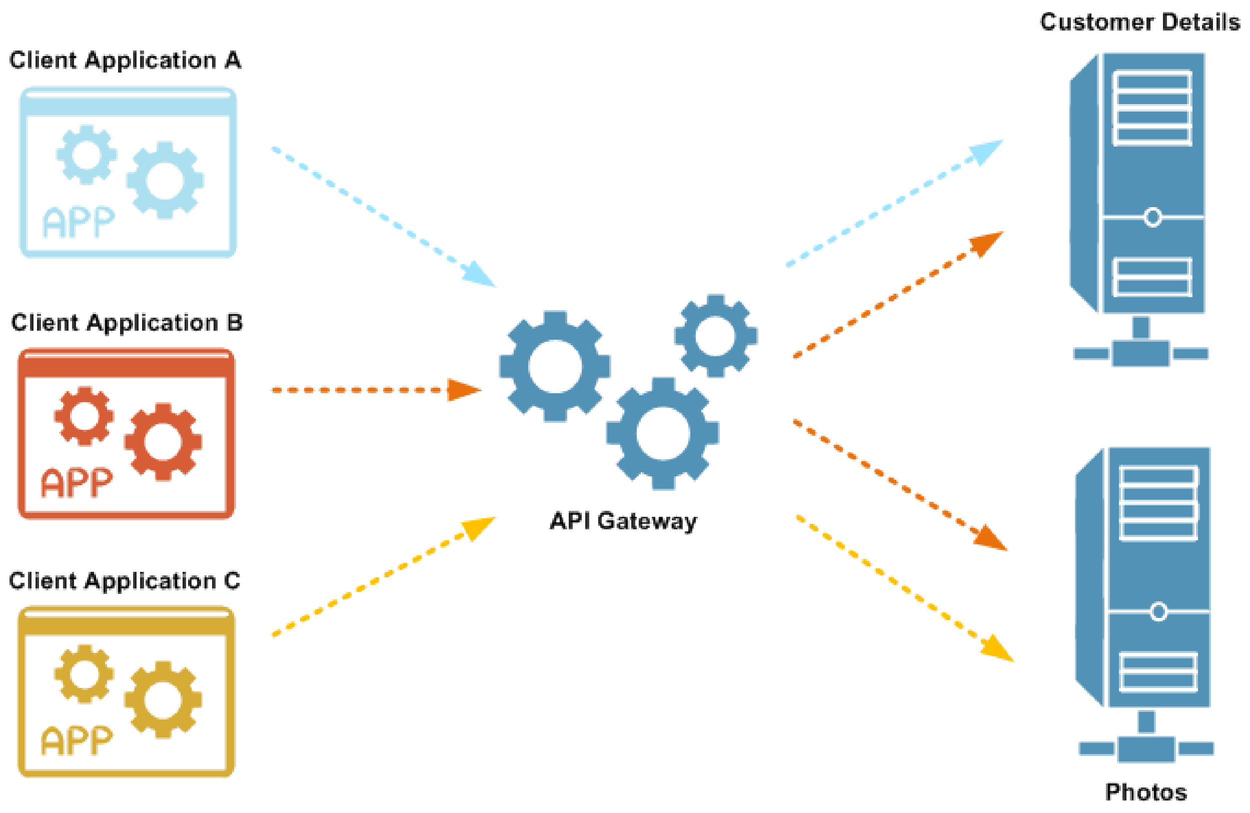


API Gateway OAuth scopes

An OAuth scope is a text string used to control access to protected resources. The resource that the scope is associated with determines the meaning of the scope. For example, if a `customer_details` scope is associated with a particular resource, and a client application is associated with the `customer_details` scope, the client application will have access to that resource. Client applications and resources can have multiple OAuth scopes.

For example, in the following overview diagram:

- Client application A can access the `customer_details` scope.
- Client application B can access the `customer_details` and `photos` scopes.
- Client application C can access the `photos` scope only.



You can configure the scopes that a client application can access in the Client Application Registry web interface. For more details, see *Manage OAuth 2.0 client applications*.

Tip

In general, good OAuth design involves a finite number of OAuth scopes. You should decide on the set of scopes to be used in your system instead of creating too many scopes later on.

OAuth 2.0 authentication flows

The API Gateway supports the following authentication flows:

- **OAuth 2.0 Authorization Code Grant (Web Server)**:
  The Web server authentication flow is used by applications that are hosted on a secure server. A critical aspect of the Web server flow is that the server must be able to protect the issued client application's secret.
- **OAuth 2.0 Implicit Grant (User-Agent)**:
  The user-agent authentication flow is used by client applications residing in the user's device. This could be implemented in a browser using a scripting language such as JavaScript or Flash. These client applications cannot keep the client application secret confidential.
- **OAuth 2.0 Resource Owner Password Credentials**:
  This username-password authentication flow can be used when the client application already has the Resource Owner's credentials.
- **OAuth 2.0 Client Credentials**:
  This username-password flow is used when the client application needs to directly access its own resources on the Resource Server. Only the client application's credentials are used in this flow. The Resource Owner's credentials are not required.
- **OAuth 2.0 JWT**:
  This flow is similar to OAuth 2.0 Client Credentials. A JSON Web Token (JWT) is a JSON-based security token encoding that enables identity and security information to be shared across security domains.
- **OAuth 2.0 Refresh Token**:
  After the client application has been authorized for access, it can use a refresh token to get a new access token. This is only done after the consumer already has received an access token using the Authorization Code Grant or Resource Owner Password Credentials flow.
- **OAuth 2.0 Revoke Token**:
  A revoke token request causes the removal of the client application permissions associated with the particular token to access the end-user's protected resources.
- **OAuth 2.0 Token Information Service**:
  The OAuth Token Info service responds to requests for information on a specified OAuth 2.0 access token.

Further information

For more details on the API Gateway OAuth 2.0 support, see the following topics:

- *Set up API Gateway OAuth 2.0*
- *Manage OAuth 2.0 client applications*
- *API Gateway OAuth 2.0 authentication flows*

For more details on OAuth 2.0, see the OAuth 2.0 Authorization Framework [http://tools.ietf.org/html/rfc6749].

Set up API Gateway OAuth 2.0

Overview

This chapter describes how to configure the OAuth 2.0 support provided with the API Gateway. It describes how to enable the OAuth 2.0 endpoints used to manage client applications, and how to import the preregistered examples provided with the API Gateway. It also explains how to migrate existing OAuth 2.0 applications.

Enable OAuth 2.0 management

The OAuth Service is not available in the basic installation. It must be deployed manually. However, there is a convenience script in $VDISTDIR/samples/scripts/oauth for deploying the OAuth 2.0 Services Listener, supporting policies and sample apps, this can be run from $VDISTDIR/samples/scripts with:

Linux:

```
./run.sh oauth/deployOAuthConfig.py --type=authzserver
```

Windows:

```
run.bat oauth\deployOAuthConfig.py --type=authzserver
```

The parameters for this script are as follows:

```
Usage: deployOAuthConfig.py [options]

Options:
  -h, --help            show this help message and exit
  -u USERNAME, --username=USERNAME
                        The user to connect to the topology (default 'admin')
  -p PASSWORD, --password=PASSWORD
                        The password for the user to connect to the topology
                        connect user (default 'changeme')
  --port=PORT           The port Client Application registry is listening on
                        (default 8089)
  --admin=ADMIN         The Client Application Registry admin name (default
                        regadmin)
  --adminpw=ADMINPW     The Client Application Registry admin password
                        (default changeme)
  --type=TYPE           The deployment type: "authzserver", "clientdemo" or
                        "all" (default all)
  -g GROUP, --group=GROUP
                        The group name
  -n SERVICE, --service=SERVICE
                        The service name
```

The API Gateway provides the following endpoints used to manage OAuth 2.0 client applications:

Description	**URL**
Authorization Endpoint (REST API)	`https://`*`GATEWAY`*`:8089/api/oauth/authorize`
Token Endpoint (REST API)	`https://`*`GATEWAY`*`:8089/api/oauth/token`

Description	URL
Token Info Endpoint (REST API)	`https://`*`GATEWAY`*`:8089/api/oauth/tokeninfo`
Revoke Endpoint (REST API)	`https://`*`GATEWAY`*`:8089/api/oauth/revoke`
Oracle Client Application Registry (HTML Interface)	`https://`*`GATEWAY`*`:8089`
Oracle Client Application Registry (REST API)	`https://`*`GATEWAY`*`:8089/api/kps/ClientApplicationRegistry`

In this table, *`GATEWAY`* refers to the machine on which the API Gateway is installed.

Important

You must first enable the OAuth listener port in the API Gateway before these endpoints are available.

Enable OAuth endpoints

To enable the OAuth management endpoints on your API Gateway, perform the following steps:

1. In the Policy Studio tree, select **Listeners** -> **API Gateway** -> **OAuth 2.0 Services** -> **Ports**.
2. Right-click the **OAuth 2.0 Interface** in the panel on the right, and select **Edit**.
3. Select **Enable Interface** in the dialog.
4. Click the **Deploy** button in the toolbar.
5. Enter a description and click **Finish**.

Note

On Linux-based systems, such as Oracle Enterprise Linux, you must open the firewall to allow external access to port `8089`. If you need to change the port number, set the value of the `env.PORT.OAUTH2.SERVICES` environment variable. For details on setting external environment variables for API Gateway instances, see the *API Gateway Deployment and Promotion Guide*.

Import client applications

The API Gateway ships with a number of preregistered sample client applications, if deploying with the deployOAuthConfig.py script these samples will already be imported. If the script is not used this section explains how to manually import these applications into the Client Application Registry.

Note

The sample client applications are for demonstration purposes only and should be removed before moving the Authorization Server into production.

For example, the default example client applications include the following:

Client ID	Client Secret
`SampleConfidentialApp`	`6808d4b6-ef09-4b0d-8f28-3b05da9c48ec`
`SamplePublicApp`	`3b001542-e348-443b-9ca2-2f38bd3f3e84`

Import the sample client applications

To import the preregistered example client applications, perform the following steps:

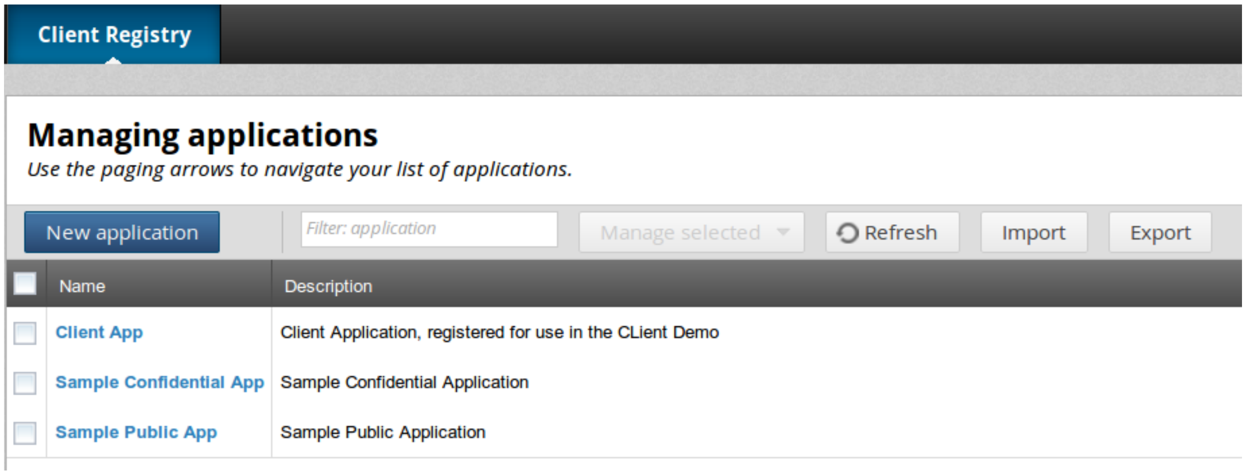
1. Access the **Client Application Registry** Web interface at the following URL:

   ```
   https://localhost:8089
   ```

2. Enter the default username/password of `admin/changeme`.
3. Click the **Import** button at the top right of the screen.
4. Select the following sample file in the dialog:

   ```
   $VDISTDIR/samples/scripts/oauth/sampleapps.dat
   ```

   `VDISTDIR` specifies the directory in which the API Gateway is installed.
5. You can also enter a **Decryption Secret** in the dialog. However, the `sampleapps.dat` file is in plaintext format, and does not require a password.
6. Click **OK** to import the two sample applications. The following screen shows these applications imported into the **Client Application Registry**:



Managing applications

Use the paging arrows to navigate your list of applications.

Name	Description
Client App	Client Application, registered for use in the CLient Demo
Sample Confidential App	Sample Confidential Application
Sample Public App	Sample Public Application

Alternatively, you can use the following script to import the sample client application data without using the Client Application Registry Web interface:

```
$VDISTDIR/samples/scripts/oauth/importSampleData.py
```

You can edit this script to configure your user credentials and file location.

Migrate client applications

If you are migrating from API Gateway version 11.1.2.0.x, you can use the following script to migrate your existing OAuth client applications:

```
$VDISTDIR/samples/scripts/oauth/migrateFrom71.py
```

This script enables you to first export your existing client application data, which you can then import as described in the section called “Import client applications”. This script has a `--password` parameter if you wish to encrypt the exported data for transport.

Migrate existing client applications

To migrate your existing client applications, perform the following steps:

1. After installing API Gateway 11.1.2.3.0, copy the `$VDISTIR/samples/oauth/migrateFrom71.py` file to the same location in your existing API Gateway 11.1.2.0.x installation:

   ```
   $VDISTIR/samples/oauth/migrateFrom71.py
   ```

2. In your existing API Gateway 11.1.2.0.x installation, ensure that `$VDISTIR/samples/scripts/common.py` has the correct `defServerName` and `defGroupName` variables set for your existing topology.
3. Run the `migrateFrom71.py` script against your running version 11.1.2.0.x Admin Node Manager and API Gateway. The script outputs the following file:

   ```
   $VDISTIR/samples/oauth/appregistry/encodedapps.dat
   ```

   Note

   If you wish to encrypt the data, run the script with the `--password` parameter.

4. Check the `encodedapps.dat` file to ensure that the export has been successful.
5. Import the `encodedapps.dat` output by the script into a running API Gateway 11.1.2.3.0 using the Client Application Registry web interface. For more details, see the section called “Import client applications”. When importing encrypted data, you must enter a password in the **Decryption Secret** field.

Upgrade API Gateway configuration

If you are migrating from a previous API Gateway version, you must upgrade your API Gateway configuration. To generate an upgraded API Gateway version 11.1.2.3.0 configuration, perform the following steps:

1. Run the following script from your version 11.1.2.3.0 installation directory:

   ```
   <11.1.2.3.0_install>/platform/bin/upgradeConfig --groups -d <previous-version-install>
     -o path/to/upgrade/output/
   ```

2. In Policy Studio, select **File** > **Open File**.
3. Specify the following file:

   ```
   path/to/upgrade/output/groups/group-2/conf/<guid>/configs.xml
   ```

4. In the open configuration in the Policy Studio tree, under **Key Property Stores**, delete **ApiKeyStore** and **ClientApplicationRegistry**.
5. Select **File** > **Save** > **Deployment Package** to export a `.fed` file.
6. Start the version 11.1.2.3.0 Admin Node Manager and API Gateway instance.
7. In Policy Studio, close the connection to the file, and connect to the now running 7.2 Admin Node Manager. Before connecting to the API Gateway instance, click **Deploy**.

8. Click **Browse for .fed**, and select the `.fed` file exported previously in step 4.
9. Import the client applications using the the web-based portal on `https://localhost:8089` by clicking **Import**, and browsing to the file created in the previous section:

   ```
   <11.1.2.3.0_install>/samples/oauth/appregistry/encodedapps.dat>
   ```

For more details on upgrading API Gateway configuration, see the *API Gateway Installation and Configuration Guide*.

Manage OAuth 2.0 client applications

Overview

Client applications that send OAuth requests to the API Gateway’s Authorization Server must be registered with the Authorization Server. This chapter describes the registry used to store these client applications, and how to manage them using a REST API-based HTML interface. This topic also includes details on the relational database schema, and SSL commands used for the example client applications.

Note

This topic assumes that you have already performed the steps described in *Set up API Gateway OAuth 2.0*. These include enabling the OAuth endpoints, importing sample applications, and migrating existing client applications.

Manage registered client applications

Every client application that sends OAuth requests to the API Gateway's OAuth Authorization Server must be registered with the Client Application Registry. The API Gateway provides the Client Application Registry Web-based HTML interface for managing registered client applications. If you have API Manager installed, the Client Application Registry is available in the API Manager web-based interface. The API Gateway also provides the Client Application Registry REST API to enable you to manage registered clients on the command line.

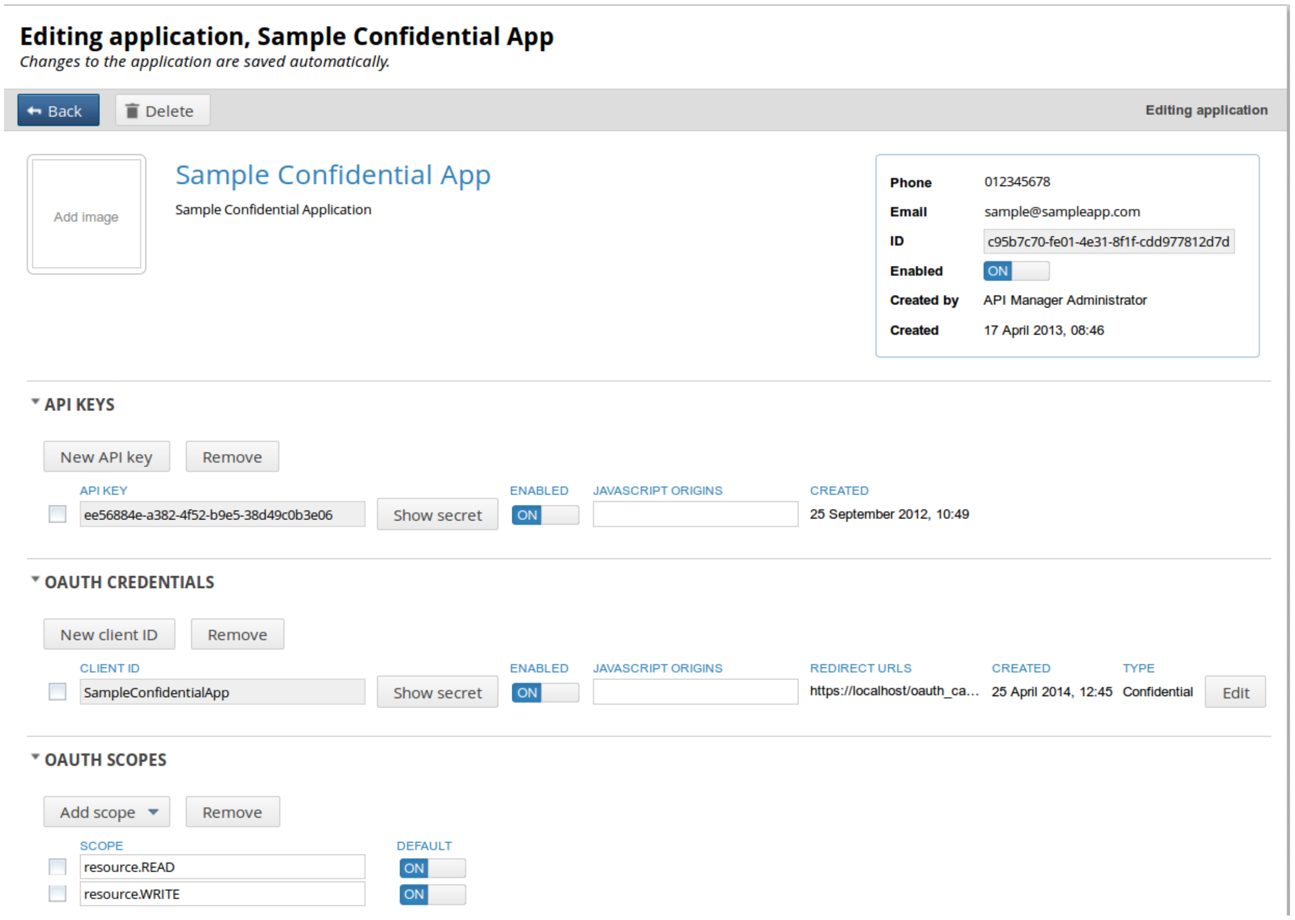
Access the Client Application Registry web interface

You can access the Client Application Registry Web interface at the following URL:

```
https://localhost:8089
```

The default username/password is `admin/changeme`.

You can select a client registration entry to update its details. For example, you can configure APIs, user sharing, API keys, credentials, quota plans, and scopes by expanding the appropriate link at the left:


Editing application, Sample Confidential App

Changes to the application are saved automatically.

Back | Delete | Editing application

Add image

Sample Confidential App

Sample Confidential Application

Phone	012345678
Email	sample@sampleapp.com
ID	c95b7c70-fe01-4e31-8f1f-cdd977812d7d
Enabled	ON
Created by	API Manager Administrator
Created	17 April 2013, 08:46

API KEYS

New API key | Remove

API KEY		ENABLED	JAVASCRIPT ORIGINS	CREATED
ee56884e-a382-4f52-b9e5-38d49c0b3e06	Show secret	ON		25 September 2012, 10:49

OAUTH CREDENTIALS

New client ID | Remove

CLIENT ID		ENABLED	JAVASCRIPT ORIGINS	REDIRECT URLS	CREATED	TYPE	
SampleConfidentialApp	Show secret	ON		https://localhost/oauth_ca...	25 April 2014, 12:45	Confidential	Edit

OAUTH SCOPES

Add scope | Remove

SCOPE	DEFAULT
resource.READ	ON
resource.WRITE	ON

By default, the Client Application Registry is backed by an embedded Apache Cassandra database.

Run the sample client applications

The API Gateway includes sample Jython client applications for all supported OAuth flows in the following directory your API Gateway installation:

```
INSTALL_DIR/samples/scripts/oauth
```

To run a sample script, open a UNIX shell or DOS command prompt in the following directory:

```
INSTALL_DIR/samples/scripts
```

Windows

For example, run the following command:

```
> run.bat oauth\implicit_grant.py
```

Linux/Solaris

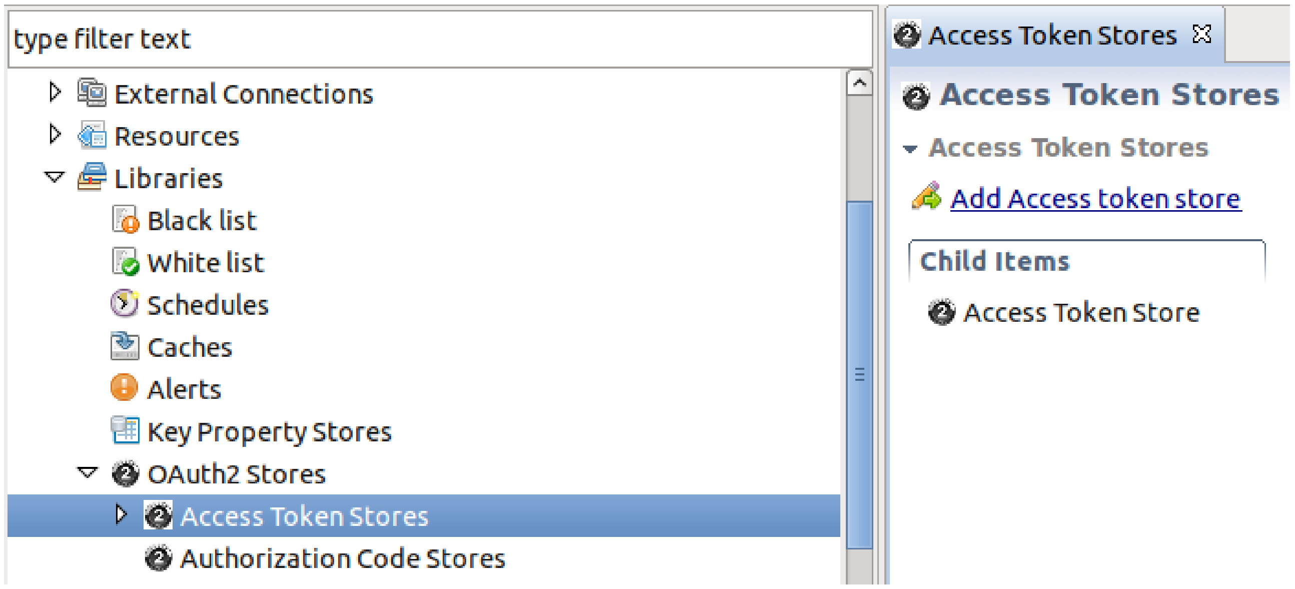
For example, run the following command:

```
> sh run.sh oauth/implicit_grant.py
```

Manage access tokens and authorization codes

API Gateway can store generated authorization codes and access tokens in its caches, in an embedded database, or in a relational database. The Authorization Server issues tokens to clients on behalf of a Resource Owner to use when authenticating subsequent API calls to the Resource Server. These issued tokens must be persisted so that subsequent client requests to the Authorization Server can be validated.

The following screen shows the OAuth stores in the Policy Studio:

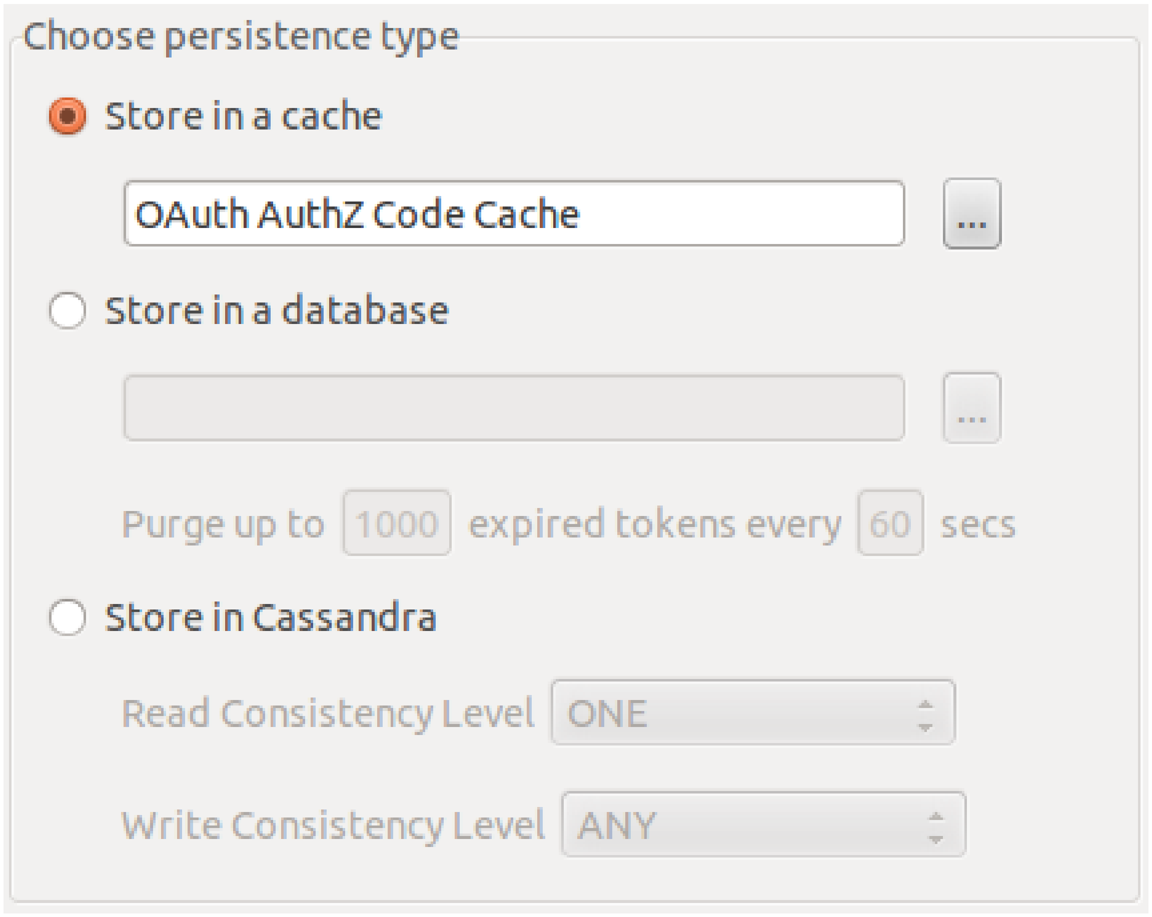


The Authorization Server can cache authorization codes and access tokens depending on the OAuth flow. The steps for adding an authorization code cache are similar to adding an access token cache.

The Authorization Server offers the following persistent storage options for access tokens and authorization codes:

- API Gateway cache (default)
- Relational Database Management System (RDBMS)
- Embedded Apache Cassandra database

The following screen shows these options in the Policy Studio:



[✓ Store in a cache] OAuth AuthZ Code Cache

Store in a database

Purge up to 1000 expired tokens every 60 secs

Store in Cassandra

Read Consistency Level ONE

Write Consistency Level ANY

The **Purge expired tokens every** 60 secs setting enables you to configure the time in seconds that a background process polls the cache or database looking for expired access/refresh tokens or authorization codes.

Store in a cache

Perform the following steps:

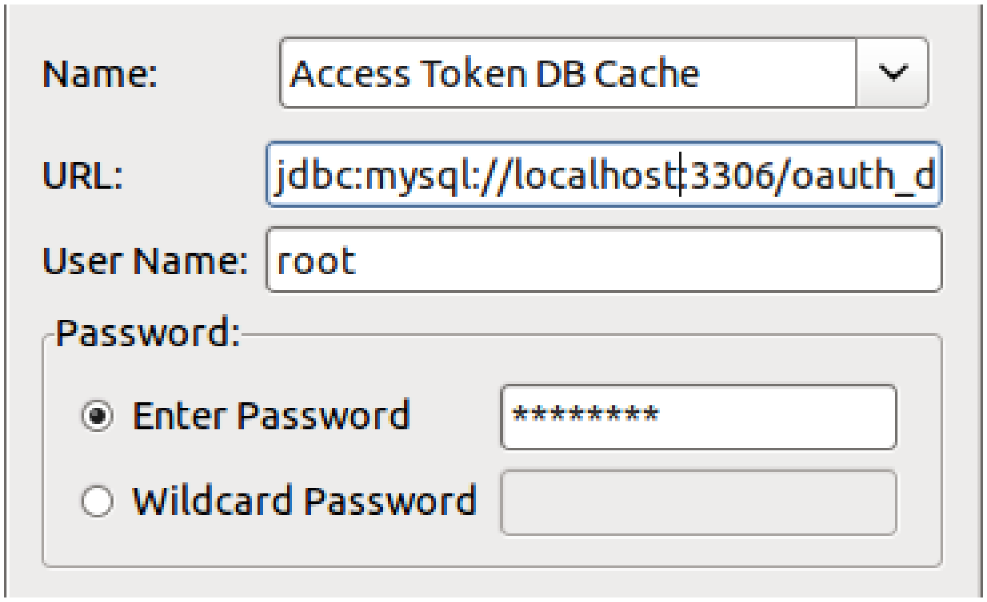
1. Right-click **Access Token Stores** in the Policy Studio tree, and select **Add Access Token Store**.
2. In dialog that enables you to choose the persistence type, select **Store in a cache**, and select the browse button to display the cache configuration dialog.
3. Add a new cache (for example, `OAuth Access Token Cache`). For more details, see the *API Gateway User Guide*.

Store in a relational database

Perform the following steps:

1. Create the supporting schema required for the storage of access tokens, refresh tokens, and authorization codes using the sql commands in $VDISTIR\system\conf\sql\<DBMS>\oauth-server.sql where <DBMS> is the Database Management System being used. Schema are provided for Microsoft SQLServer, MySQL, Oracle RDBMS and DB2
2. Right-click **Access Token Stores** in the Policy Studio tree, and select **Add Access Token Store**.

3. In the dialog that enables you to choose the persistence type, select **Store in a database**, and select the browse button to display a database configuration dialog.
4. Complete the database configuration details. The following example uses a MySQL instance named `oauth_db`. For more details, see the *API Gateway User Guide*.



Name: Access Token DB Cache
URL: jdbc:mysql://localhost:3306/oauth_d
User Name: root
Password:
(● Enter Password) ********
Wildcard Password

For more details, see the section called “Relational database-backed Client Application Registry”.

Store in Cassandra

Perform the following steps:

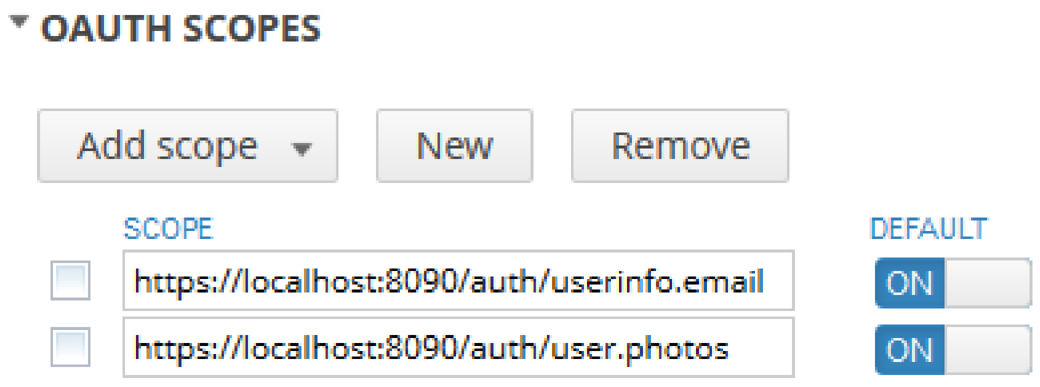
1. Right-click **Access Token Stores** in the Policy Studio tree, and select **Add Access Token Store**.
2. This displays the dialog that enables you to choose the persistence type. Select **Store in Cassandra**.
3. You can configure **Read** and **Write** consistency levels for the Cassandra database. These control how up-to-date and synchronized a row of data is on all of its replicas. The default **Read** setting of `ONE` means that the database returns a response from the closest replica. The default **Write** setting of `ANY` means that a write must be written to at least one replica node. For more details, see http://www.datastax.com/docs/0.8/dml/data_consistency.

Manage OAuth scopes

An OAuth scope is a text string used to control access to resources. The resource that the scope is associated with determines the meaning of the scope. For example, if a `vehicle_data` scope is associated with a particular resource, and a client application is associated with the `vehicle_data` scope, the client application will have access to that resource. Client applications and resources can have multiple OAuth scopes.

You can configure the scopes that a client application can access in the Client Application Registry web interface. You can specify scopes as free-form text or choose from a list of known configured scopes. You can also select a scope as a default scope for client applications. Default scopes are used when an authorization or token request does not contain scopes. The full list of scopes (default and non-default) represent the list of scopes that can be included in an authorization or token request.

You can manage scopes in the Client Application Registry web interface by expanding **OAUTH SCOPES**:



https://localhost:8090/auth/userinfo.email ON

https://localhost:8090/auth/user.photos ON

Note

The example default scopes provided with the API Gateway are URL-based. However, you can specify any text string for an OAuth scope (for example, `customer_details` or `readonly`).

When an authorization code or access token request is received from a client application, the API Gateway OAuth access token filters check that the scopes in the message match the scopes configured for the client application. If no scopes are provided in the message, the filter creates an access token for the scopes that are configured as default. The scope for which the access token was created is checked against the list of available scopes in the Client Application Registry web interface. This list is generated from the scopes defined in Validate Access Token filter in the server configuration. For more details on this filter, see *Validate access token*.

Important

You can also specify OAuth scopes using selectors (for example, use `${http.request.verb}` to map HTTP `GET` and `PUT` requests). However, the Client Application Registry web interface does not display selectorized scopes in the list of available scopes. This is because selectorized scopes in the Validate Access Token filter cannot be evaluated at registration time.

The administrator must therefore find out about any selectorized scopes to be applied to resources at runtime. If a scope must be conifigured using a selector, the administrator must find out exactly which selector to specify in the scope. For more details on selectors, see the *API Gateway User Guide*.

Relational database-backed Client Application Registry

By default, the Oracle Client Application Registry Key Property Store (KPS) is backed by an Apache Cassandra database. The Oracle Client Application Registry KPS can also be backed by a relational database such as Oracle, MySQL, DB2, or Microsoft MySQL Server. For more details, see the *Key Property Store User Guide*, available from Oracle Support.

OAuth relational database schemas

For example, the OAuth relational database schemas displayed by example `mysql` commands are as follows:

oauth_access_token schema
The following shows the result from the `show columns from oauth_access_token;` command:

```
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| id            | varchar(255) | NO   | PRI | NULL    |       |
| auth_request  | blob         | NO   |     | NULL    |       |
| client_id     | varchar(255) | NO   |     | NULL    |       |
| expiry_time   | datetime     | NO   |     | NULL    |       |
| token         | blob         | NO   |     | NULL    |       |
| refresh_token | varchar(255) | YES  |     | NULL    |       |
| user_auth     | varchar(255) | NO   |     | NULL    |       |
| user_name     | varchar(255) | NO   |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
```

oauth_refresh_token schema
The following shows the result from the `show columns from oauth_refresh_token;` command:

```
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| token_id     | varchar(255) | NO   | PRI | NULL    |       |
| auth_request | blob         | NO   |     | NULL    |       |
| expiry_time  | datetime     | NO   |     | NUL     |       |
| token        | blob         | NO   |     | NULL    |       |
| user_name    | varchar(255) | NO   |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

oauth_authz_code schema
The following shows the result from the `show columns from oauth_authz_code;` command:

```
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| id            | varchar(255) | NO   | PRI | NULL    |       |
| authorization | blob         | NO   |     | NULL    |       |
| expiry_time   | datetime     | NO   |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
```

Generate a certificate and private key for a client application

The following example `openssl` command shows generating a client application certificate and private key:

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mykey.pem
-out mycert.pem
Generating a 1024 bit RSA private key
...................................................................
...........++++++
.....++++++
writing new private key to 'mykey.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank.
For some fields there will be a default value.
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
```

```
State or Province Name (full name) [Some-State]:MA
Locality Name (eg, city) []:Newton
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Oracle
Organizational Unit Name (eg, section) []:API Gateway
Common Name (eg, YOUR name) []:SampleConfidentialApp
Email Address []:support@widgits.com
```

API Gateway OAuth 2.0 authentication flows

Overview

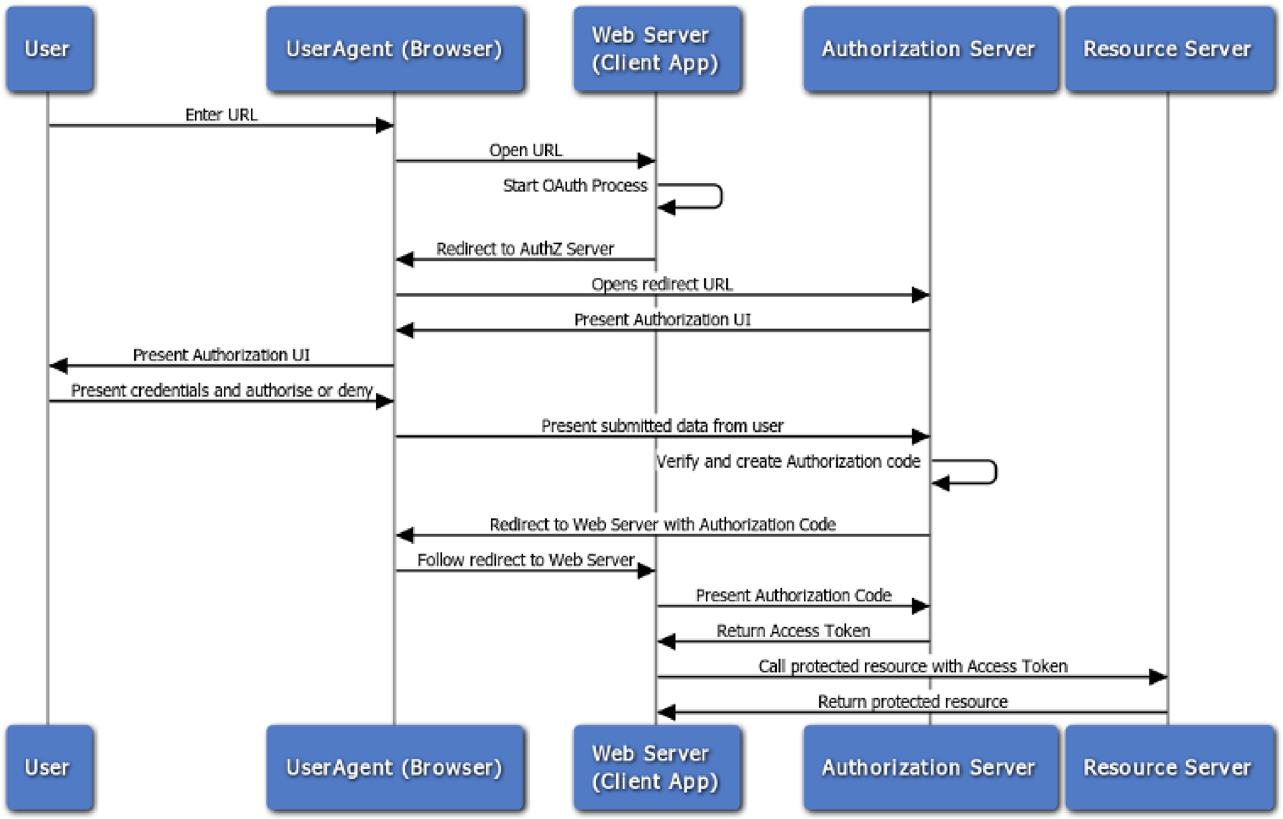
The API Gateway can use the OAuth 2.0 protocol for authentication and authorization. The API Gateway can act as an OAuth 2.0 Authorization Server and supports several OAuth 2.0 flows that cover common Web server, JavaScript, device, installed application, and server-to-server scenarios. This topic describes each of the supported OAuth 2.0 flows in detail, and shows how to run example client applications.

Authorization code (or web server) flow

The authorization code or web server flow is suitable for clients that can interact with the end-user’s user-agent (typically a Web browser), and that can receive incoming requests from the authorization server (can act as an HTTP server). The Authorization Code flow is also known as the *Three-Legged OAuth* flow.

The authorization code flow is as follows:

1. The web server redirects the user to the API Gateway acting as an authorization server to authenticate and authorize the server to access data on their behalf.
2. After the user approves access, the web server receives a callback with an authorization code.
3. After obtaining the authorization code, the web server passes back the authorization code to obtain an access token response.
4. After validating the authorization code, the API Gateway passes back a token response to the web server.
5. After the token is granted, the web server accesses their data.



Obtain an access token

The detailed steps for obtaining an access token are as follows:

1. Redirect the user to the authorization endpoint with the following parameters:

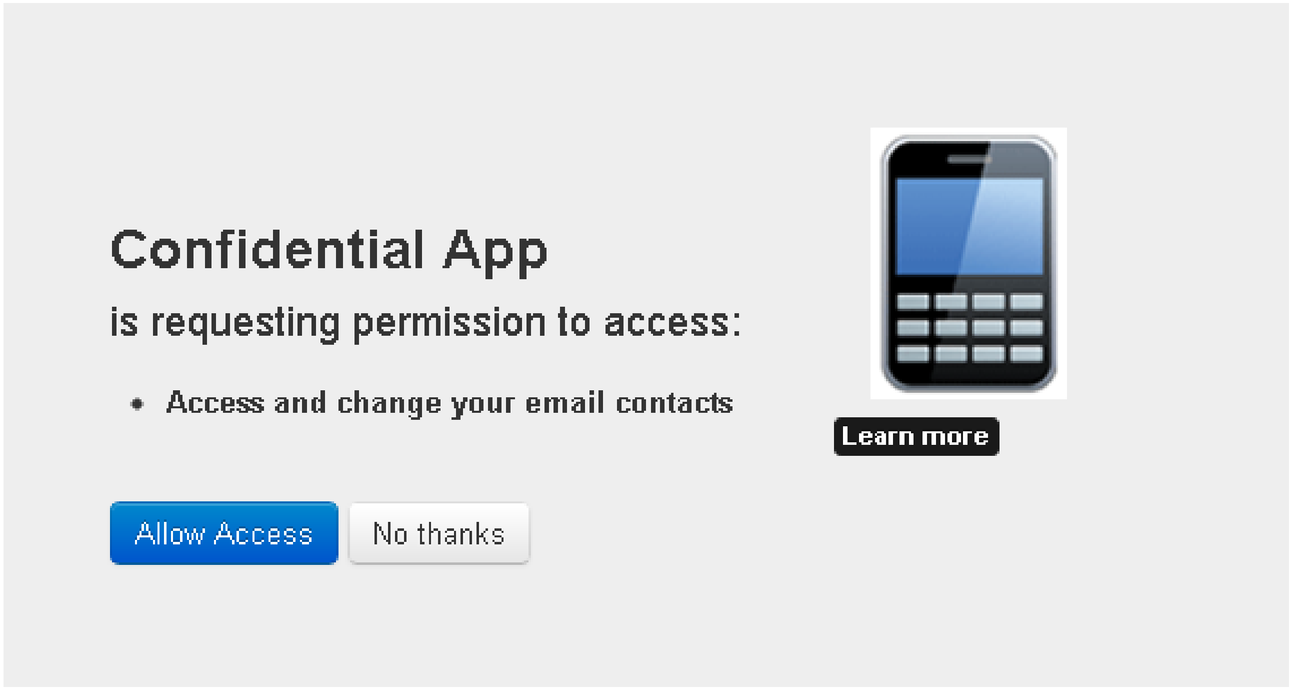
Parameter	Description
`response_type`	Required. Must be set to `code`.
`client_id`	Required. The Client ID generated when the application was registered in the Oracle Client Application Registry.
`redirect_uri`	Optional. Where the authorization code will be sent. This value must match one of the values provided in the Oracle Client Application Registry.
`scope`	Optional. A space delimited list of scopes, which indicate the access to the Resource Owner's data being requested by the application.
`state`	Optional. Any state the consumer wants reflected back to it after approval during the callback.

The following is an example URL:

```
https://apigateway/oauth/authorize?client_id=SampleConfidentialApp&
response_type=code&&redirect_uri=http%3A%2F%2Flocalhost%3A8090%2Fauth%2Fredirect.html&
scope=https%3A%2F%2Flocalhost%3A8090%2Fauth%2Fuserinfo.email
```

Note

During this step the Resource Owner user must approve access for the application Web server to access their protected resources, as shown in the following example screen.



Confidential App

is requesting permission to access:

- Access and change your email contacts

2. The response to the above request is sent to the `redirect_uri`. If the user approves the access request, the response contains an authorization code and the `state` parameter (if included in the request). If the user does not approve the request, the response contains an error message. All responses are returned to the Web server on the query string. For example:

```
https://localhost/oauth_callback&code=9srN6sqmjrvG5bWvNB42PCGju0TFVV
```

3. After the Web server receives the authorization code, it may exchange the authorization code for an access token and a refresh token. This request is an HTTPS `POST`, and includes the following parameters:

Parameter	Description
`grant_type`	Required. Must be set to `authorization_code`.
`code`	Required. The authorization code received in the redirect above.
`redirect_uri`	Required. The redirect URL registered for the application during application re-

Parameter	Description
	gistration.
`client_id*`	Optional. The `client_id` obtained during application registration.
`client_secret*`	Optional. The `client_secret` obtained during application registration.
`format`	Optional. Expected return format. The default is `json`. Possible values are: • `urlencoded` • `json` • `xml`

`*` If the `client_id` and `client_secret` are not provided as parameters in the HTTP `POST`, they must be provided in the HTTP Basic Authentication header (`Authorization base64Encoded(client_id:client_secret)`).

The following example HTTPS `POST` shows some parameters:

```
POST /api/oauth/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded

client_id=SampleConfidentialApp&client_secret=6808d4b6-ef09-4b0d-8f28-3b05da9c48ec
 &code=9srN6sqmjrvG5bWvNB42PCGju0TFVV&redirect_uri=http%3A%2F%2Flocalhost%3A809
 0%2Fauth%2Fredirect.html&grant_type=authorization_code&format=query
```

4. After the request is verified, the API Gateway sends a response to the client. The following parameters are in the response body:

Parameter	Description
`access_token`	The token that can be sent to the Resource Server to access the protected resources of the Resource Owner (user).
`refresh_token`	A token that may be used to obtain a new access token.
`expires`	The remaining lifetime on the access token.
`type`	Indicates the type of token returned. At this time, this field always has a value of `Bearer`.

The following is an example response:

```
HTTP/1.1 200 OK
Cache-Control: no-store
Content-Type: application/json
Pragma: no-cache{
    "access_token": “O91G451HZ0V83opz6udiSEjchPynd2Ss9......",
    "token_type": "Bearer",
    "expires_in": "3600",
}
```

5. After the Web server has obtained an access token, it can gain access to protected resources on the Resource Server by placing it in an `Authorization: Bearer` HTTP header:

```
GET /oauth/protected HTTP/1.1
Authorization: Bearer O91G451HZ0V83opz6udiSEjchPynd2Ss9
Host: apigateway.com
```

For example, the `curl` command to call a protected resource with an access token is as follows:

```
curl -H "Authorization: Bearer O91G451HZ0V83opz6udiSEjchPynd2Ss9"
  https://apigateway.com/oauth/protected
```

Run the sample client

The following Jython sample client creates and sends an authorization request for the authorization grant flow to the Authorization Server:

```
INSTALL_DIR/samples/scripts/oauth/authorization_code.py
```

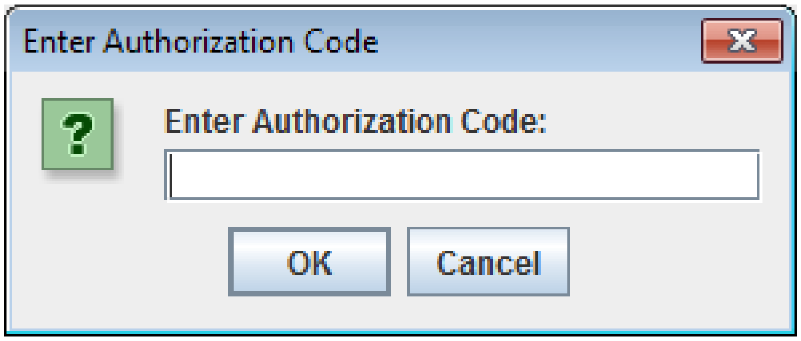
To run the sample, perform the following steps:

1. Open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

   ```
   > run oauth/authorization_code.py
   ```

   The script outputs the following:

   ```
   > Go to the URL here:
   http://127.0.0.1:8080/api/oauth/authorize?client_id=SampleConfidentialApp
     &response_type=code&scope=https%3A%2F%2Flocalhost%3A8090%2Fauth%2Fuserinfo.email
     &redirect_uri=https%3A%2F%2Flocalhost%2Foauth_callback
   Enter Authorization code in dialog
   ```



2. Copy the URL output to the command prompt into a browser, and perform the following steps as prompted:
   a. Provide login credentials to the authorization server. The default values are:
      - Username: `admin`
      - Password: `changeme`

   b. When prompted, grant access to the client application to access the protected resource.

3. After the Resource Owner has authorized and approved access to the application, the Authorization Server redirects a fragment containing the authorization code to the redirection URI. For example:

   ```
   https://localhost/oauth_callback&code=AaI5Or3RYB2uOgiyqVsLs1ATIY0ll0
   ```

   In this example, the authorization code is:

   ```
   AaI5Or3RYB2uOgiyqVsLs1ATIY0ll0
   ```

   Enter this value into the **Enter Authorization Code** dialog. The script will exchange the authorization code for an access token, and then access the protected resource using the access token. For example:

   ```
   Enter Authorization code in dialog
   AuthZ code: AaI5Or3RYB2uOgiyqVsLs1ATIY0ll0
   Exchange authZ code for access token
   Sending up access token request using grant_type set to authorization_code
   Response from access token request: 200
   Parsing the json response
   **********************ACCESS TOKEN RESPONSE***********************************
   Access token received from authorization server icPgKP2uVUD2thvAZ5ENhsQb66ffnZEC
    XHyRQEz5zP8aGzcobLV3AR
   Access token type received from authorization server Bearer
   Access token expiry time: 3599
   Refresh token:  NpNbzIVVvj8MhMmcWx2zsawxxJ3YADfc0XIxlZvw0tIhh8
   ******************************************************************************
   Now we can try access the protected resource using the access token
   Executing get request on the protected url
   Response from protected resource request is: 200
   <html>Congrats! You've hit an OAuth protected resource</html>
   ```

Further information

For details on API Gateway filters that support this flow, see the following topics:

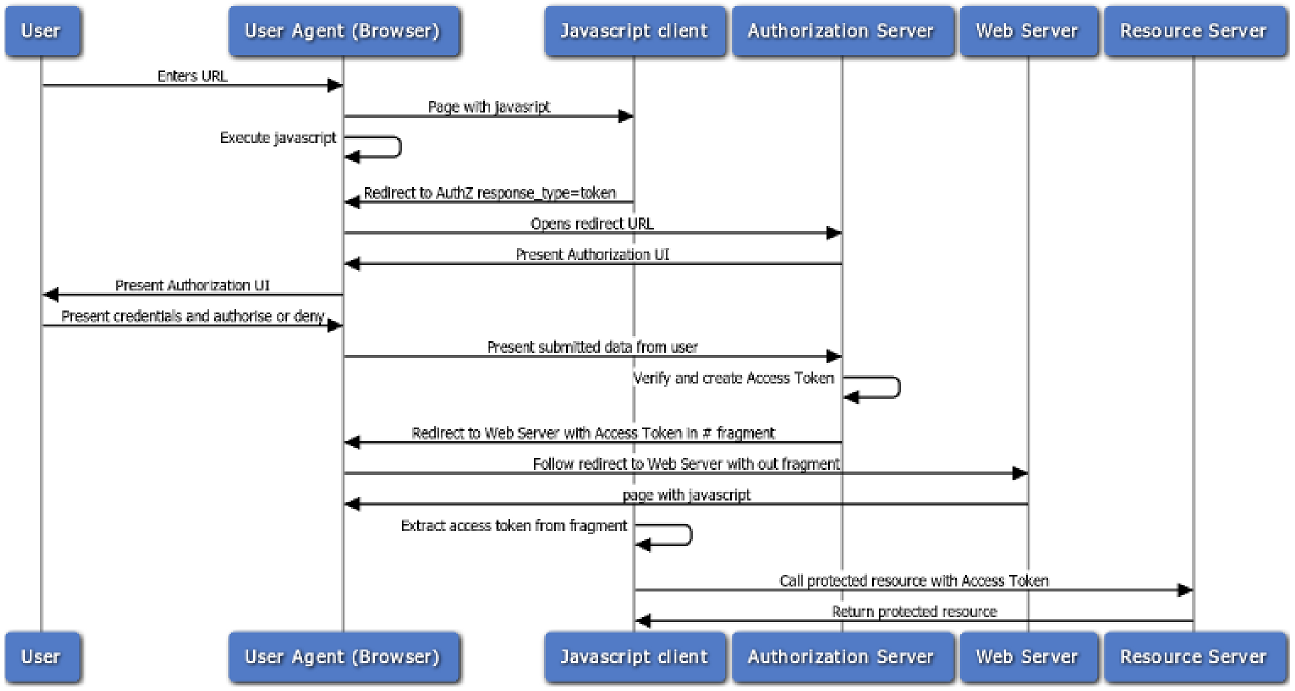
- *Get access token using authorization code*
- *Consume authorization requests*
- *Authorize transaction*

Implicit grant (or user agent) flow

The implicit grant (user-agent) authentication flow is used by client applications (consumers) residing in the user's device. This could be implemented in a browser using a scripting language such as JavaScript, or from a mobile device or a desktop application. These consumers cannot keep the client secret confidential (application password or private key).

The user agent flow is as follows:

1. The web server redirects the user to the API Gateway acting as an authorization server to authenticate and authorize the server to access data on their behalf.
2. After the user approves access, the web server receives a callback with an access token in the fragment of the redirect URL.
3. After the token is granted, the application can access the protected data with the access token.


User
User Agent (Browser)
Javascript client
Authorization Server
Web Server
Resource Server
Enters URL
Page with javasript
Execute javascript
Redirect to AuthZ response_type=token
Opens redirect URL
Present Authorization UI
Present Authorization UI
Present credentials and authorise or deny
Present submitted data from user
Verify and create Access Token
Redirect to Web Server with Access Token in # fragment
Follow redirect to Web Server with out fragment
page with javascript
Extract access token from fragment
Call protected resource with Access Token
Return protected resource

Obtain an access token

The detailed steps for obtaining an access token are as follows:

1. Redirect the user to the authorization endpoint with the following parameters:

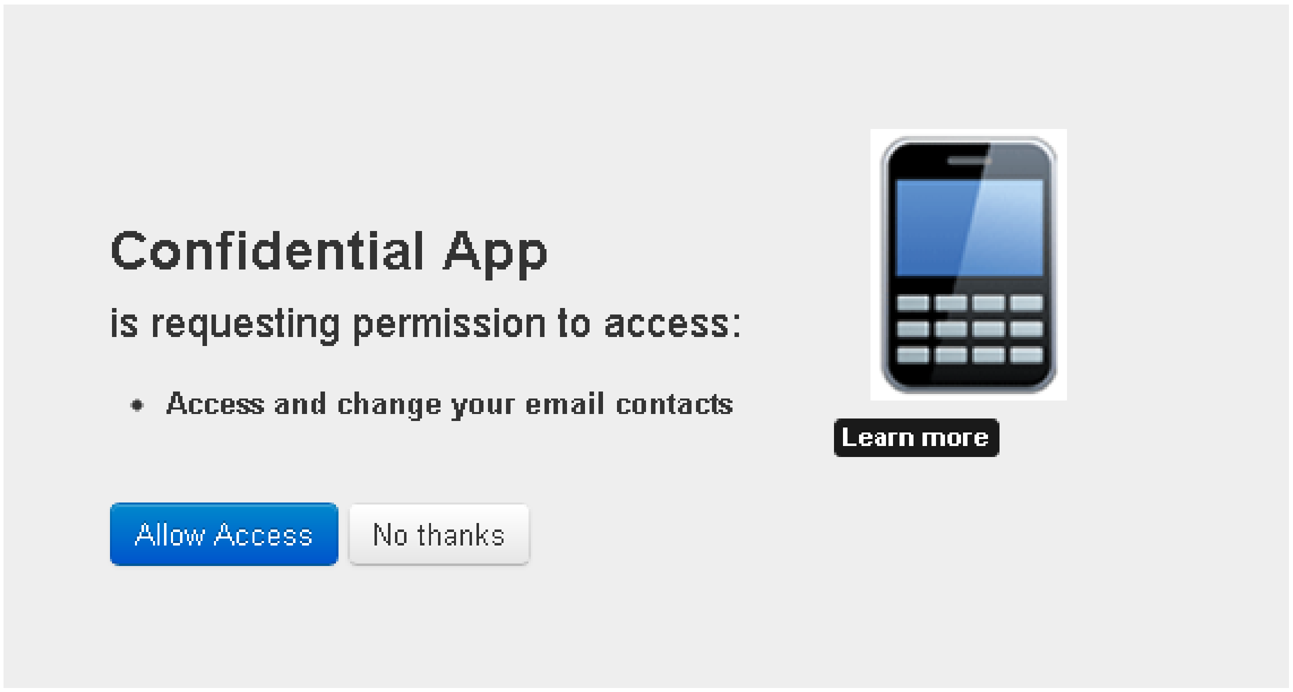
Parameter	Description
`response_type`	Required. Must be set to token.
`client_id`	Required. The Client ID generated when the application was registered in the Oracle Client Application Registry.
`redirect_uri`	Optional. Where the access token will be sent. This value must match one of the values provided in the Oracle Client Application Registry.
`scope`	Optional. A space delimited list of scopes, which indicates the access to the Resource Owner's data requested by the application.
`state`	Optional. Any state the consumer wants reflected back to it after approval during the callback.

The following is an example URL:

```
https://apigateway/oauth/authorize?client_id=SampleConfidentialApp&response_type=
  token&&redirect_uri=http%3A%2F%2Flocalhost%3A8090%2Fauth%2Fredirect.html&scope=
  https%3A%2F%2Flocalhost%3A8090%2Fauth%2Fuserinfo.email
```

Note

During this step the Resource Owner user must approve access for the application (Web server) to access their protected resources, as shown in the following example screen.



Confidential App

is requesting permission to access:

- Access and change your email contacts

2. The response to the above request is sent to the `redirect_uri`. If the user approves the access request, the response contains an access token and the state parameter (if included in the request). For example:

```
https://localhost/oauth_callback#access_token=19437jhj2781FQd44AzqT3Zg
 &token_type=Bearer&expires_in=3600
```

If the user does not approve the request, the response contains an error message.

3. After the request is verified, the API Gateway sends a response to the client. The following parameters are contained in the fragment of the redirect:

Parameter	Description
`access_token`	The token that can be sent to the Resource Server to access the protected resources of the Resource Owner (user).
`expires`	The remaining lifetime on the access token.
`type`	Indicates the type of token returned. At this time, this field will always have a value of `Bearer`.
`state`	Optional. If the client application sent a value for state in the original authorization request, the state parameter is populated with this value.

4. After the application has obtained an access token, it can gain access to protected resources on the Resource Server by placing it in an `Authorization: Bearer` HTTP header:

```
GET /oauth/protected HTTP/1.1
Authorization: Bearer O91G451HZ0V83opz6udiSEjchPynd2Ss9
Host: apigateway.com
```

For example, the `curl` command to call a protected resource with an access token is as follows:

```
curl -H "Authorization: Bearer O91G451HZ0V83opz6udiSEjchPynd2Ss9"
https://apigateway.com/oauth/protected
```

Run the sample client

The following Jython sample client creates and sends an authorization request for the implicit grant flow to the Authorization Server:

```
INSTALL_DIR/samples/scripts/oauth/implicit_grant.py
```

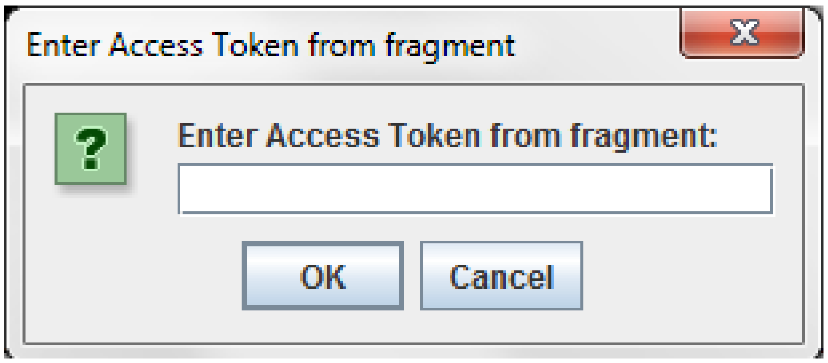
To run the sample, perform the following steps:

1. Open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

   ```
   > run oauth/implicit_grant.py
   ```

   The script outputs the following:

   ```
   > Go to the URL here:
    http://127.0.0.1:8080/api/oauth/authorize?client_id=SampleConfidentialApp&
     response_type=token&scope=https%3A%2F%2Flocalhost%3A8090%2Fauth%2Fuserinfo.email&
     redirect_uri=https%3A%2F%2Flocalhost%2Foauth_callback&state=1956901292
    Enter Access Token code in dialog
   ```


Enter Access Token from fragment
Enter Access Token from fragment:
OK
Cancel

2. After the Resource Owner has authorized and approved access to the application, the Authorization Server redirects to the redirection URI a fragment containing the access token. For example:

   ```
   https://localhost/oauth_callback#access_token=
    4owzGyokzLLQB5FH4tOMk7Eqf1wqYfENEDXZ1mGvN7u7a2Xexy2OU9&expires_in=
    3599&state=1956901292&token_type=Bearer
   ```

   In this example, the access token is:

```
4owzGyokzLLQB5FH4tOMk7Eqf1wqYfENEDXZ1mGvN7u7a2Xexy2OU9
```

Enter this value into the **Enter Access Token from fragment** dialog, and the script attempts to access the protected resource using the access token. For example:

```
*********************ACCESS TOKEN RESPONSE******************************
Access token received from authorization server 4owzGyokzLLQB5FH4tOMk7Eqf1wqYfEN
EDXZ1mGvN7u7a2Xexy2OU9
***********************************************************************
Now we can try access the protected resource using the access token
Executing get request on the protected url
Response from protected resource request is: 200
<html>Congrats! You've hit an OAuth protected resource</html>
```

Further information

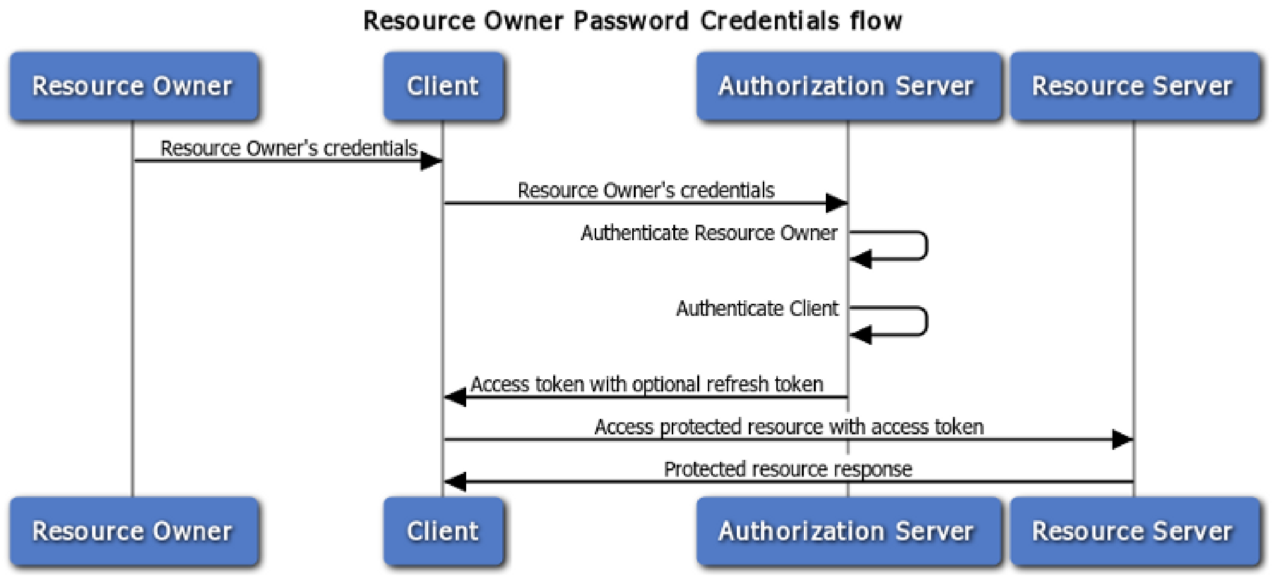
For details on the API Gateway filter that supports this flow, see the *Consume authorization requests* filter.

Resource owner password credentials flow

The resource owner password credentials flow is also known as the username-password authentication flow. This flow can be used as a replacement for an existing login when the consumer already has the user’s credentials.

The Resource Owner password credentials grant type is suitable in cases where the Resource Owner has a trust relationship with the client (for example, the device operating system or a highly privileged application). The Authorization Server should take special care when enabling this grant type, and only allow it when other flows are not viable.

This grant type is suitable for clients capable of obtaining the Resource Owner's credentials (username and password, typically using an interactive form). It is also used to migrate existing clients using direct authentication schemes such as HTTP Basic or Digest authentication to OAuth by converting the stored credentials to an access token.


Resource Owner Password Credentials flow
Resource Owner
Client
Authorization Server
Resource Server
Resource Owner's credentials
Resource Owner's credentials
Authenticate Resource Owner
Authenticate Client
Access token with optional refresh token
Access protected resource with access token
Protected resource response
Resource Owner
Client
Authorization Server
Resource Server

Request an access token

The client token request should be sent in an HTTP `POST` to the token endpoint with the following parameters:

Parameter	Description
`grant_type`	Required. Must be set to `password`
`username`	Required. The Resource Owner's user name.
`password`	Required. The Resource Owner's password.
`scope`	Optional. The scope of the authorization.
`format`	Optional. Expected return format. The default is `json`. Possible values are: • `urlencoded` • `json` • `xml`

The following is an example HTTP `POST` request:

```
POST /api/oauth/token HTTP/1.1
Content-Length: 424
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Host: 192.168.0.48:8080
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JWgrant_type=password&username=
 johndoe&password=A3ddj3w
```

Handle the response

The API Gateway will validate the resource owner's credentials and authenticate the client against the Oracle Client Application Registry. An access token, and optional refresh token, is sent back to the client on success. For example, a valid response is as follows:

```
HTTP/1.1 200 OK
 Cache-Control: no-store
 Content-Type: application/json
 Pragma: no-cache
 {
     "access_token": “O91G451HZ0V83opz6udiSEjchPynd2Ss9......",
     "token_type": "Bearer",
     "expires_in": "3600",
     “refresh_token”: “8722gffy2229220002iuueee7GP..........”
 }
```

Run the sample client

The following Jython sample client sends a request to the Authorization Server using the Resource Owner password credentials flow:

```
INSTALL_DIR/samples/scripts/oauth/resourceowner_password_credentials.py
```

To run the sample, open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

```
> run oauth/resourceowner_password_credentials.py
```

The script outputs the following:

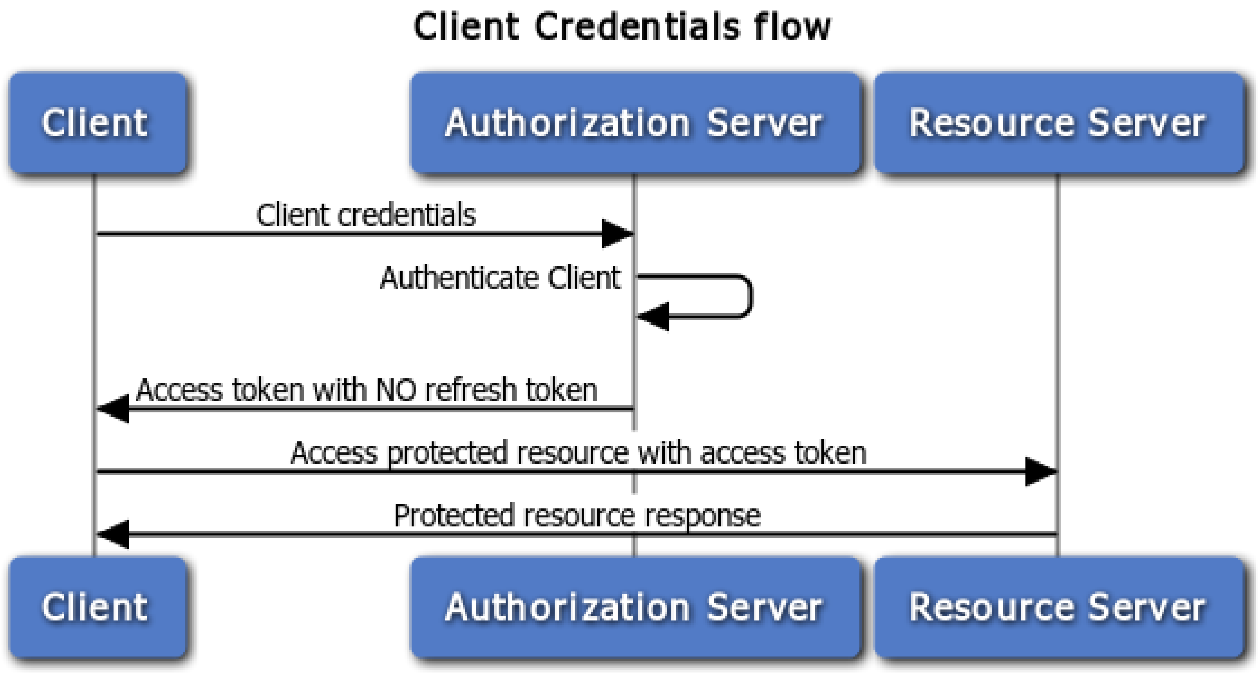
```
Sending up access token request using grant_type set to password
Response from access token request: 200
Parsing the json response
*********************ACCESS TOKEN RESPONSE***********************************
Access token received from authorization server lrGHhFhFwSmycXStIza1jjvXlSaac9
 JNIgviF7oPiV8OnxlSIsrxVA
Access token type received from authorization server Bearer
Access token expiry time: 3600
*****************************************************************************
Now we can try access the protected resource using the access token
Executing get request on the protected url
Response from protected resource request is: 200
<html>Congrats! You've hit an OAuth protected resource</html>
```

Further information

For details on the API Gateway filter that supports this flow, see *Get access token using resource owner credentials*.

Client credentials grant flow

The client credentials grant type must only be used by confidential clients. The client can request an access token using only its client credentials (or other supported means of authentication) when the client is requesting access to the protected resources under its control. The client can also request access to those of another Resource Owner that has been previously arranged with the Authorization Server (the method of which is beyond the scope of the specification).



Request an access token

The client token request should be sent in an HTTP POST to the token endpoint with the following parameters:

Parameter	Description
`grant_type`	Required. Must be set to `client_credentials`.
`scope`	Optional. The scope of the authorization.
`format`	Optional. Expected return format. The default is `json`. Possible values are: • `urlencoded` • `json` • `xml`

The following is an example POST request:

```
POST /api/oauth/token HTTP/1.1
Content-Length: 424
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Host: 192.168.0.48:8080
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
grant_type=client_credentials
```

Handle the response

The API Gateway authenticates the client against the Oracle Client Application Registry. An access token is sent back to the client on success. A refresh token is not included in this flow. An example valid response is as follows:

```
HTTP/1.1 200 OK
Cache-Control: no-store
Content-Type: application/json
Pragma: no-cache
{    "access_token": “O91G451HZ0V83opz6udiSEjchPynd2Ss9......",
     "token_type": "Bearer",
     "expires_in": "3600"
}
```

Run the sample client

The following Jython sample client sends a request to the Authorization Server using the client credentials flow:

```
INSTALL_DIR/samples/scripts/oauth/client_credentials.py
```

To run the sample, open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

```
> run oauth/client_credentials.py
```

The outputs the following:

```
Sending up access token request using grant_type set to client_credentials
Response from access token request: 200
Parsing the json response
**********************ACCESS TOKEN RESPONSE***********************************
Access token received from authorization server
```

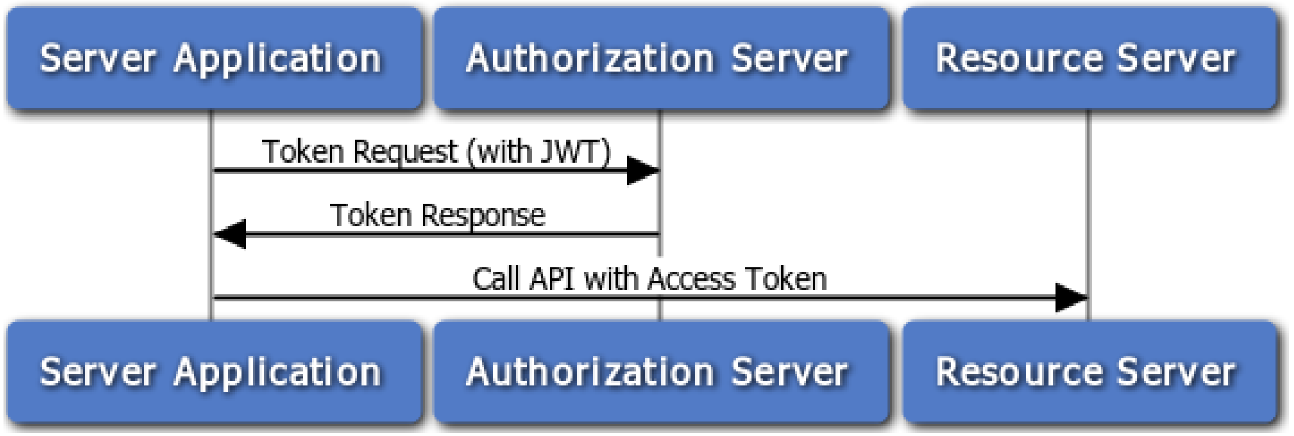
```
OjtVvNusLg2ujy3a6IXHhavqdEPtK7qSmIj9fLl8qywPyX8bKEsjqF
Access token type received from authorization server Bearer
Access token expiry time: 3599
******************************************************************************
Now we can try access the protected resource using the access token
Response from protected resource request is: 200
<html>Congrats! You've hit an OAuth protected resource</html>
```

Further information

For details on the API Gateway filter that supports this flow, see *Get access token using client credentials*.

JWT flow

A JSON Web Token (JWT) is a JSON-based security token encoding that enables identity and security information to be shared across security domains.



In the OAuth 2.0 JWT flow, the client application is assumed to be a confidential client that can store the client application's private key. The X.509 certificate that matches the client's private key must be registered in the Oracle Client Application Registry. The API Gateway uses this certificate to verify the signature of the JWT claim. For information on creating a private key and certificate, see the section called "Generate a certificate and private key for a client application".

For more details on the OAuth 2.0 JWT flow, see
http://self-issued.info/docs/draft-ietf-oauth-jwt-bearer-00.html

Create a JWT bearer token

To create a JWT bearer token, perform the following steps:

1. Construct a JWT header in the following format:

   ```
   {"alg":"RS256"}
   ```

2. Base64url encode the JWT Header as defined here, which results in the following:

   ```
   eyJhbGciOiJSUzI1NiJ9
   ```

3. Create a JWT Claims Set, which conforms to the following rules:
   - The issuer (`iss`) must be the OAuth `client_id` or the remote access application for which the developer registered their certificate.

- The audience (`aud`) must match the value configured in the JWT filter. By default, this value is as follows:

  ```
  http://apigateway/api/oauth/token
  ```

- The validity (`exp`) must be the expiration time of the assertion, within five minutes, expressed as the number of seconds from `1970-01-01T0:0:0Z` measured in UTC.
- The time the assertion was issued (`iat`) measured in seconds after `00:00:00` UTC, January 1, 1970.
- The JWT must be signed (using RSA SHA256).
- The JWT must conform with the general format rules specified here: http://tools.ietf.org/html/draft-jones-json-web-toke.

For example:

```
{
"iss": "SampleConfidentialApp",
    "aud": "http://apigateway/api/oauth/token",
    "exp": "1340452126",
    "iat": "1340451826"
}
```

4. Base64url encode the JWT Claims Set, resulting in:

```
eyJpc3MiOiJTYW1wbGVDb25maWRlbnRpYWxBcHAiLCJhdWQiOiJodHRwOi8vYXBpc2VydmV
  yL2FwaS9vYXV0aC90b2tlbiIsImV4cCI6IjEzNDA0NTIxMjYiLCJpYXQiOiIxMzQwNDUxODI2In0=
```

5. Create a new string from the encoded JWT header from step 2, and the encoded JWT Claims Set from step 4, and append them as follows:

```
Base64URLEncode(JWT Header) + . + Base64URLEncode(JWT Claims Set)
```

This results in a string as follows:

```
eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiAiU2FtcGxlQ29uZmlkZW50aWFsQXBwIiwgImF1ZCI6ICJodHRw
 Oi8vYXBpc2VydmVyL2FwaS9vYXV0aC90b2tlbiIsICJleHAiOiAiMTM0MTM1NDYwNSIsICJpYXQiOiAi
 MTM0MTM1NDMwNSJ9
```

6. Sign the resulting string in step 5 using SHA256 with RSA. The signature must then be Base64url encoded. The signature is then concatenated with a `.` character to the end of the Base64url representation of the input string. The result is the following JWT (line breaks added for clarity):

```
{Base64url encoded header}.
{Base64url encoded claim set}.
```

This results in a string as follows:

```
eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiAiU2FtcGxlQ29uZmlkZW50aWFsQXBwIiwgImF1ZCI6ICJodHR
wOi8vYXBpc2VydmVyL2FwaS9vYXV0aC90b2tlbiIsICJleHAiOiAiMTM0MTM1NDYwNSIsICJpYXQiOiA
iMTM0MTM1NDMwNSJ9.ilWR8O8OlbQtT5zBaGIQjveOZFIWGTkdVC6LofJ8dN0akvvD0m7IvUZtPp4dx3
KdEDj4YcsyCEAPhfopUlZO3LE-iNPlbxB5dsmizbFIc2oGZr7Zo4IlDf92OJHq9DGqwQosJ-s9GcIRQk
-IUPF4lVy1Q7PidPWKR9ohm3c2gt8
```

Request an access token

The JWT bearer token should be sent in an HTTP `POST` to the Token Endpoint with the following parameters:

Parameter	Description
`grant_type`	Required. Must be set to `urn:ietf:params:oauth:grant-type:jwt-bearer`.

Parameter	Description
`assertion`	Required. Must be set to the JWT bearer token, base64url-encoded.
`format`	Optional. Expected return format. The default is `json`. Possible values are: • `urlencoded` • `json` • `xml`

The following is an example `POST` request:

```
POST /api/oauth/token HTTP/1.1
Content-Length: 424
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Host: 192.168.0.48:8080
grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer&assertion=eyJhbGciOiJS
 UzI1NiJ9.eyJpc3MiOiAiU2FtcGxlQ29uZmlkZW50aWFsQXBwIiwgImF1ZCI6ICJodHRwOi8vYXBpc2Vy
 dmVyL2FwaS9vYXV0aC90b2tlbiIsICJleHAiOiAiMTM0MTM1NDYwNSIsICJpYXQiOiAiMTM0MTM1NDMwN
 SJ9.ilWR8O8OlbQtT5zBaGIQjveOZFIWGTkdVC6LofJ8dN0akvvD0m7IvUZtPp4dx3KdEDj4YcsyCEAPh
 fopUlZO3LE-iNPlbxB5dsmizbFIc2oGZr7Zo4IlDf92OJHq9DGqwQosJ-s9GcIRQk-IUPF4lVy1Q7PidP
 WKR9ohm3c2gt8
```

Handle the response

The API Gateway returns an access token if the JWT claim and access token request are properly formed, and the JWT has been signed by the private key matching the registered certificate for the client application in the Oracle Client Application Registry.

For example, a valid response is as follows:

```
HTTP/1.1 200 OK
Cache-Control: no-store
Content-Type: application/json
Pragma: no-cache
{
    "access_token": “O91G451HZ0V83opz6udiSEjchPynd2Ss9......",
    "token_type": "Bearer",
    "expires_in": "3600",
}
```

Run the sample client

The following Jython sample creates and sends a JWT Bearer token to the Authorization Server:

```
INSTALL_DIR/samples/scripts/oauth/jwt.py
```

To run the sample, open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

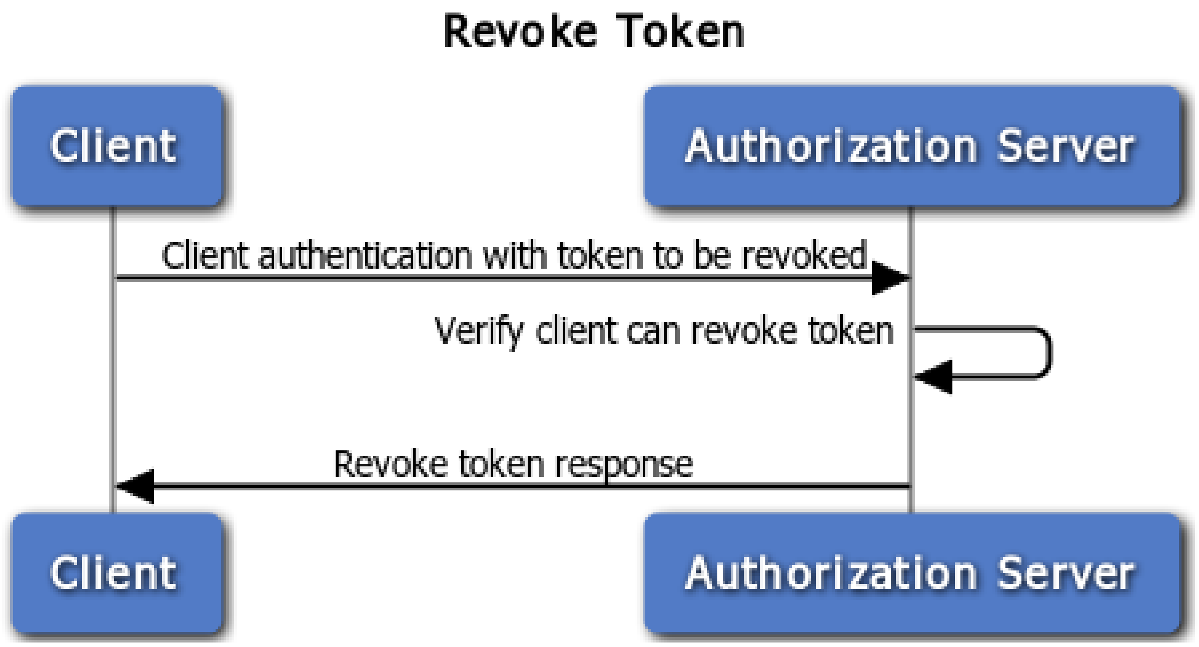
```
> run oauth/jwt.py
```

Further information

For details on the API Gateway filter that supports this flow, see *Get access token using JWT*.

Revoke token

In some cases a user may wish to revoke access given to an application. An access token can be revoked by calling the API Gateway revoke service and providing the access token to be revoked. A revoke token request causes the removal of the client permissions associated with the particular token to access the end-user's protected resources.



The endpoint for revoke token requests is as follows:

```
https://<API Gateway>:8089/api/oauth/revoke
```

The token to be revoked should be sent to the revoke token endpoint in an HTTP `POST` with the following parameter:

Parameter	Description
`token`	Required. A token to be revoked (for example, `4eclEUX1N6oVIOoZBbaDTI977SV3T9KqJ3ayOvs4gqhGA4`).

The following is an example POST request:

```
POST /api/oauth/revoke HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Host: 192.168.0.48:8080
Authorization: Basic U2FtcGxlQ29uZmlkZW50aWFsQXBwOjY4MDhkNGI2LWVmMDktNGIwZC04ZjI4LT
NiMDVkYTljNDhlYw==token=4eclEUX1N6oVIOoZBbaDTI977SV3T9KqJ3ayOvs4gqhGA4
```

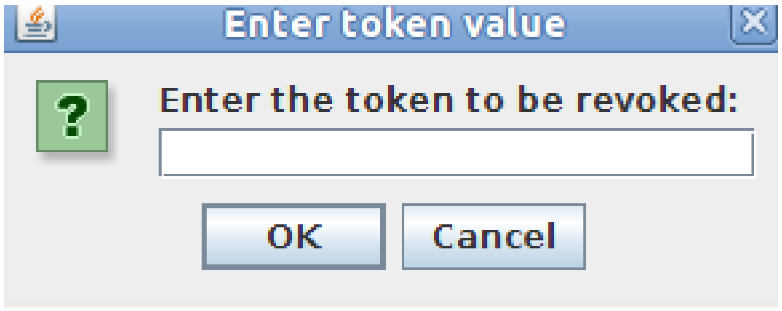
Run the sample client

The following Jython sample client creates a token revoke request to the Authorization Server:

```
INSTALL_DIR/samples/scripts/oauth/revoke_token.py
```

To run the sample, open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

```
> run oauth/revoke_token.py
```



When the Authorization Server receives the token revocation request, it first validates the client credentials and verifies whether the client is authorized to revoke the particular token based on the client identity.

Note

Only the client that was issued the token can revoke it.

The Authorization Server decides whether the token is an access token or a refresh token:

- If it is an access token, this token is revoked.
- If it is a refresh token, all access tokens issued for the refresh token are invalidated, and the refresh token is revoked.

Response codes

The following HTTP status response codes are returned:

- HTTP 200 if processing is successful.
- HTTP 401 if client authentication failed.
- HTTP 403 if the client is not authorized to revoke the token.

The following is an example response:

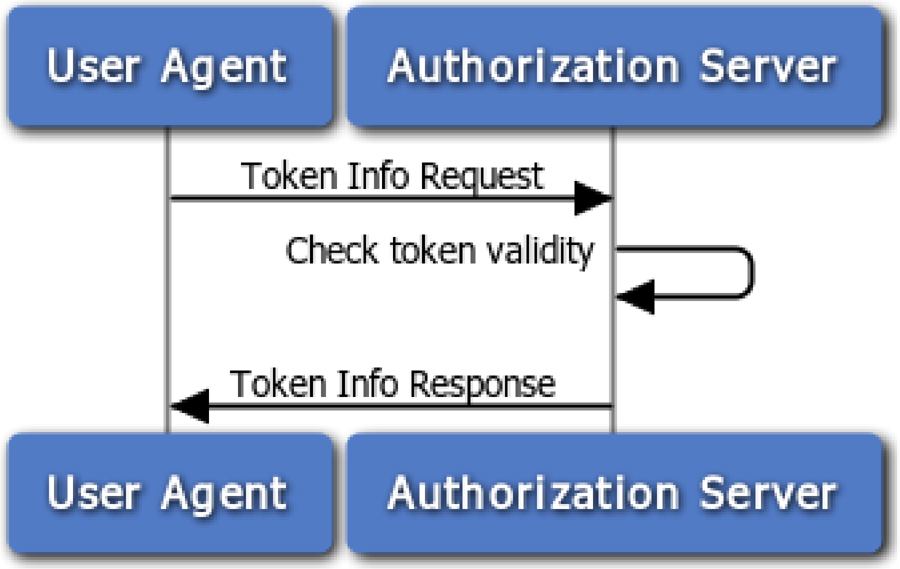
```
Token to be revoked: 3eXnUZzkODNGb9D94Qk5XhiV4W4gu9muZ56VAYoZiot4WNhIZ72D3
Revoking token...............
Response from revoke token request is: 200
Successfully revoked token
```

Further information

For details on the API Gateway filter that supports this flow, see *Revoke token*.

Token information service

You can use the token information service to validate that an access token was issued by the API Gateway. A request to the `tokenInfo` service is an HTTP `GET` request for information in a specified OAuth 2.0 access token.



The endpoint for the token information service is as follows:

```
https://<apigateway>:8089/api/oauth/tokeninfo
```

Getting information about a token from the Authorization Server only requires a `GET` request to the tokeninfo endpoint. For example:

```
GET /api/oauth/tokeninfo HTTP/1.1
Host: 192.168.0.48:8080
access_token=4eclEUX1N6oVIOoZBbaDTI977SV3T9KqJ3ayOvs4gqhGA4
```

This request includes the following parameter:

Parameter	Description
`access_token`	Required. A token that you want information about (for example: `4eclEUX1N6oVIOoZBbaDTI977SV3T9KqJ3ayOvs4gqhGA4`)

The following example uses this parameter:

```
https://apigateway/api/oauth/tokeninfo?access_token=4eclEUX1N6oVIOoZBba
 DTI977SV3T9KqJ3ayOvs4gqhGA4
```

Run the sample client

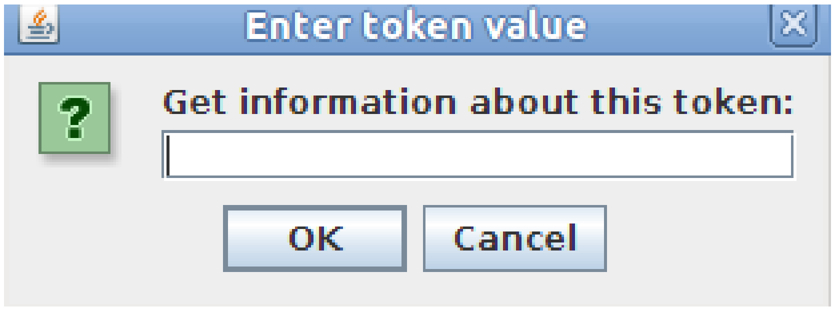
The following Jython sample client creates a token revoke request to the Authorization Server:

```
INSTALL_DIR/samples/scripts/oauth/token_info.py
```

To run the sample, open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

```
> run oauth/token_info.py
```

This displays the following dialog:



When the Authorization Server receives the Token Info request, it first ensures the token is in its cache (EhCache or Database), and ensures the token is valid and has not expired.

The following is an example response:

```
Get token info for this token: BcYGjPOQSCrtbEc1F0ag8zf6OT9rCaMLiI1dYjFLT5zhxz3x5ScrdN
Response from token info request is: 200
**********************TOKEN INFO RESPONSE***********************************
Token audience received from authorization server: SampleConfidentialApp
Scopes user consented to: https://localhost:8090/auth/userinfo.email
Token expiry time: 3566
User id : admin
*****************************************************************************
```

Response codes

The following HTTP Status codes are returned:

- 200 if processing is successful
- 400 on failure

The response is sent back as a JSON message. For example:

```
{
 "audience" : "SampleConfidentialApp",
   "user_id" : "admin",
   "scope" : "https://localhost:8090/auth/userinfo.email",
   "expires_in" : 2518
 }
```

You can get additional information about the access token using message attributes. For more details, see *OAuth 2.0 server message attributes*.

Further information

For details on the API Gateway filter that supports this flow, see *Get access token information*.

Get access token information

Overview

The OAuth 2.0 **Access Token Information** filter is used to return a JSON description of the specified OAuth 2.0 access token. OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions.

An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes. For details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

Token settings

Configure the following fields on the **Access Token Info Settings** tab:

Token to verify can be found here:
Click the browse button to select the location of the access token to verify (for example, in the default **OAuth Access Token Store**). To add a store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called “Manage access tokens and authorization codes”.

Where to get access token from:
Select one of the following:

- **In Query String**:
  This is the default setting. Defaults to the `access_token` parameter.
- **In a selector**:
  Defaults to the `${http.client.getCgiArgument('access_token')}` selector. For more details on API Gateway selectors, see the *API Gateway User Guide*.

Monitoring settings

The settings on the **Monitoring** tab configure service-level monitoring options such as whether to store usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the API Gateway Analytics tool to produce reports on how the service is used.

- **Monitor service usage**:
  Select this option if you want to store message metrics for this service.
- **Monitor service usage per client**:
  Select this option if you want to generate reports monitoring which authenticated clients are calling which services.
- **Monitor client usage**:
  If you want to generate reports on authenticated clients, but are not interested in which services they are calling, select this option and deselect **Monitoring service usage per client**.
- **Which attribute is used to identify the client?**:
  Enter the message attribute to use to identify authenticated clients. The default is `authentication.subject.id`, which stores the identifier of the authenticated user (for example, the username or user's X.509 Distinguished Name).
- **Composite Context**:
  This setting enables you to select a service context as a composite context in which multiple service contexts are monitored during the processing of a message. This setting is not selected by default.

  For example, the API Gateway receives a message, and sends it to `serviceA` first, and then to `serviceB`. Monitoring is performed separately for each service by default. However, you can set a composite service context before `serviceA` and `serviceB` that includes both services. This composite service passes if both services complete successfully, and monitoring is also performed on the composite service context.

Advanced settings

The settings on the **Advanced** tab include the following:

Return additional Access Token parameters:
Click **Add** to return additional access token parameters, and enter the **Name** and **Value** in the dialog. For example, you could enter `Department` in **Name**, and the following selector in **Value**:

```
${accesstoken.getAdditionalInformation().get("Department")
```

Get access token using authorization code

Overview

The OAuth 2.0 **Access Token using Authorization Code** filter is used to get a new access token using the authorization code. This supports the OAuth 2.0 Authorization Code Grant or Web server authentication flow, which is used by applications that are hosted on a secure server. A critical aspect of this flow is that the server must be able to protect the issued client application's secret. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

Application validation settings

Configure the following fields on this tab:

Use this store to validate the Authorization Code:
Click the browse button to select the store in which to validate the authorization code (for example, in the default **Authz Code Store**). To add a store, right-click **Authorization Code Stores**, and select **Add Authorization Code Store**. You can store codes in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called "Manage access tokens and authorization codes".

Find client application information from message:
Select one of the following:

- **In Authorization Header**
  This is the default setting.
- **In Query String**:
  The **Client Id** defaults to `client_id`, and **Client Secret** defaults to `client_secret`.

Access token settings

Configure the following fields on the this tab:

Access Token will be stored here:
Click the browse button to select where to store the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called "Manage access tokens and authorization codes".

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is selected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to `43200` (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to `46`.

Store additional Access Token parameters:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department`, `Engineering`).

Monitoring settings

The settings on this tab configure service-level monitoring options such as whether to store usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the section called “Monitoring settings” in *Get access token information*.

Get access token using client credentials

Overview

The OAuth 2.0 **Access Token using Client Credentials** filter enables an OAuth client to request an access token using only its client credentials. This supports the OAuth 2.0 Client Credentials flow, which is used when the client application needs to directly access its own resources on the Resource Server. Only the client application's credentials or public/private key pair are used in the this flow. The Resource Owner's credentials are not required. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

Application validation settings

Configure the following fields on this tab:

Find client application information from message:
Select one of the following:

- **In Authorization Header**:
  This is the default setting.
- **In Query String**:
  The **Client Id** defaults to `client_id`, and **Client Secret** defaults to `client_secret`.

Access token settings

Configure the following fields on the this tab:

Access Token will be stored here:
Click the browse button to select where to store the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called “Manage access tokens and authorization codes”.

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is selected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to

`43200` (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to `46`.

Store additional Access Token parameters:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department` and `Engineering`).

Generate Token Scopes:
When requesting a token from the Authorization Server, you can specify a parameter for the OAuth scopes that you wish to access. When scopes are sent in the request, you can select whether the access token is generated only if the scopes in the request match all or any scopes registered for the application. Alternatively, for extra flexibility you can get the scopes by calling out to a policy.

Select one of the following options to configure how access tokens are generated based on specified scopes:

- **Get scopes from a registered application**:
  Select whether the scopes must match **Any** or **All** of the scopes registered for the application in the Client Application Registry. Defaults to **Any**. If no scopes are sent in the request, the token is generated with the scopes registered for the application.
- **Get scopes by calling policy**:
  Select a pre-configured policy to get the scopes, and enter the attribute that stores the scopes in the **Scopes approved for token are stored in the attribute** textbox. Defaults to `scopes.for.token`. The configured filter requires the scopes as set of strings on the message whiteboard.

Monitoring settings

The settings on this tab configure service-level monitoring options such as whether to store usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the section called “Monitoring settings” in *Get access token information*.

Get access token using JWT

Overview

The OAuth 2.0 **Access Token using JWT** filter enables an OAuth client to request an access token using only a JSON Web Token (JWT). This supports the OAuth 2.0 JWT flow, which is used when the client application needs to directly access its own resources on the Resource Server. Only the client JWT token is used in this flow, the Resource Owner's credentials are not required. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

A JWT is a JSON-based security token encoding that enables identity and security information to be shared across security domains. JWTs represent a set of claims as a JSON object. For more information on JWT, go to http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

Application validation settings

Configure the following fields on this tab:

Audience (aud) must contain the following URI:
Enter the JWT `aud` (intended audience). The JWT must contain an `aud` URI that identifies the Authorization Server, or service provider domain, as an intended audience. The Authorization Server must also verify that it is an intended audience for the JWT. Defaults to `http://apigateway/api/oauth/token`.

Clock skew in seconds for JWT Claim:
When creating the JWT, an OAuth client can set certain claims relating to time (for example, `iat`, `exp`, or `nbf`). This field allows you to enter a number of seconds to allow for clock skew when dealing with these claims.

If the `iat` claim is present, the OAuth token service asserts that the current time is greater than the issued at time. If the `exp` claim is present, the OAuth token service asserts that the current time is less than or equal to the expiry time (plus skew seconds if configured). If the `nbf` claim is present, the OAuth token service asserts that the current time is greater than or equal to expiry time (minus skew seconds if configured).

Access token settings

Configure the following fields on this tab:

Access Token will be stored here:
Click the browse button to select where to cache the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called “Manage access tokens and authorization codes”

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protec-

ted resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is unselected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to `43200` (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to `46`.

Store additional Access Token parameters:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department` and `Engineering`).

Generate Token Scopes:
When requesting a token from the Authorization Server, you can specify a parameter for the OAuth scopes that you wish to access. When scopes are sent in the request, you can select whether the access token is generated only if the scopes in the request match all or any scopes registered for the application. Alternatively, for extra flexibility, you can get the scopes by calling out to a policy.

Select one of the following options to configure how access tokens are generated based on specified scopes:

- **Get scopes from a registered application**:
  Select whether the scopes must match **Any** or **All** of the scopes registered for the application in the Client Application Registry. Defaults to **Any**. If no scopes are sent in the request, the token is generated with the scopes registered for the application.
- **Get scopes by calling policy**:
  Select a preconfigured policy to get the scopes, and enter the attribute that stores the scopes in the **Scopes approved for token are stored in the attribute** field. Defaults to `scopes.for.token`. The configured filter requires the scopes as set of strings on the message whiteboard.

Monitoring settings

The settings on this tab configure service-level monitoring options such as whether to store usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the section called “Monitoring settings” in *Get access token information*.

Get access token using SAML assertion

Overview

The OAuth 2.0 **Access Token using SAML Assertion** filter enables an OAuth client to request an access token using a SAML assertion. This supports the OAuth 2.0 SAML flow, which is used when a client wishes to utilize an existing trust relationship, expressed through the semantics of the SAML assertion, without a direct user approval step at the authorization server. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

For more information on SAML, see the IETF draft document SAML 2.0 Profile for OAuth 2.0 [http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-18].

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

SAML assertion validation settings

Configure the following fields on this tab:

Audience and Recipient within SAML Assertion must contain the following URI:
Enter a URI that must be contained in the SAML assertion's intended audience and recipient. The SAML assertion must contain a URI that identifies the authorization server as an intended audience, and that identifies the token endpoint URL of the authorization server as a recipient. Defaults to `http://apigateway/api/oauth/token`.

Drift time (seconds):
Enter a drift time in seconds to allow for clock skew.

Call the following policy to verify SAML Assertion signature:
Click the browse button to select a policy to verify the SAML assertion signature.

Access token settings

Configure the following fields on this tab:

Access Token will be stored here:
Click the browse button to select where to cache the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called “Manage access tokens and authorization codes”

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Refresh Token Details:
Select one of the following options:

- **Generate a new refresh token:**
  Select this option to generate a new access token and refresh token pair. The old refresh token passed in the request is removed. This option is selected by default.

  Enter the number of seconds before the refresh token expires in the **Refresh Token Expiry (in secs)** field, and enter the number of characters in the refresh token in the **Refresh Token Length** field. The expiry defaults to `43200` (12 hours), and the length defaults to `46`.
- **Do not generate a refresh token:**
  Select this option to generate a new access token only. The old refresh token passed in the request is removed.

Store additional meta data with the access token which can subsequently be retrieved:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department` and `Engineering`).

Generate Token Scopes:
When requesting a token from the Authorization Server, you can specify a parameter for the OAuth scopes that you wish to access. When scopes are sent in the request, you can select whether the access token is generated only if the scopes in the request match all or any scopes registered for the application. Alternatively, for extra flexibility, you can get the scopes by calling out to a policy.

Select one of the following options to configure how access tokens are generated based on specified scopes:

- **Get scopes from a registered application**:
  Select whether the scopes must match **Any** or **All** of the scopes registered for the application in the Client Application Registry. Defaults to **Any**. If no scopes are sent in the request, the token is generated with the scopes registered for the application.
- **Get scopes by calling policy**:
  Select a preconfigured policy to get the scopes, and enter the attribute that stores the scopes in the **Scopes approved for token are stored in the attribute** field. Defaults to `scopes.for.token`. The configured filter requires the scopes as set of strings on the message whiteboard.

Monitoring settings

The settings on this tab configure service-level monitoring options such as whether to store usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the section called “Monitoring settings” in *Get access token information*.

Consume authorization requests

Overview

The OAuth 2.0 **Authorization Code Flow** filter is used to consume OAuth authorization requests, and is also known as the **Authorization Request** filter. This filter supports the OAuth 2.0 Authorization Code Grant (Web server) authentication flow, which is used by applications hosted on a secure server. A critical aspect of this flow is that the server must be able to protect the issued client application's secret. The Web server flow is suitable for clients capable of interacting with the end-user’s user-agent (typically a Web browser), and capable of receiving incoming requests from the Authorization Server (acting as an HTTP server). The Authorization Code Grant flow is also known as the *Three-Legged OAuth Flow*.

The OAuth 2.0 Authorization Code Grant flow is as follows:

1. The Web server redirects the user to the API Gateway acting as an Authorization Server to authenticate and authorize the server to access data on their behalf.
2. After the user approves access, the Web server receives a callback with an authorization code.
3. After obtaining the authorization code, the Web server passes back the authorization code to obtain an access token response.
4. After validating the authorization code, the API Gateway passes back a token response to the Web server.
5. After the token is granted, the Web server accesses their data.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

The OAuth 2.0 **Authorization Request** filter also supports the Implicit Grant (User Agent) flow. This is used by client applications (consumers) residing in the user's device (for example, in a browser using JavaScript, or from a mobile device or desktop application). These consumers cannot keep the client secret confidential (application password or private key).

For more details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

Validation settings

Configure the following fields on the **Validation/Templates** tab:

Authorize Resource Owner:
Select one of the following:

- **Use internal flow**
  Uses the internal API Gateway flow to authorize the Resource Owner. This is the default setting. The internal flow authenticates the user against the API Gateway user store, and redirects the user to the **Authorize Transaction** filter to use sample template files for login and Resource Owner scope authorization.

  Note

  If you wish to store additional information with the authorization code (for Authorization Code flow), or with an access token (for Implicit Grant flow), you must set additional parameters in the **Authorize Transaction** flow filter.

- **Call this policy**
  Click the browse button to select a policy to authorize the Resource Owner. You can use the **Policy will store sub-**

ject in selector text box to specify where the policy is stored. Defaults to the `${authentication.subject.id}` message attribute. For more details on selectors, see the *API Gateway User Guide*.

Note

If you wish to store additional information with the authorization code (for Authorization Code flow), or with an access token (for Implicit Grant flow), you must set additional parameters in the **Authorization Code Flow** filter.

Authorization code settings

Configure the following fields on the **Authz Code Details** tab:

Authorization Code will be stored here:
Click the browse button to select where to cache the access token (for example, in the default **Authz Code Store**). To add an access token store, right-click **Authorization Code Stores**, and select **Add Authorization Code Store**. You can store codes in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called “Manage access tokens and authorization codes”.

Location of Access Code redirect page:
Enter the full path to the HTML page used for the access code HTTP redirect. Defaults to the following:

```
${environment.VDISTDIR}/samples/oauth/templates/showAccessCode.html
```

`VDISTDIR` specifies the directory in which the API Gateway is installed.

Length:
Enter the number of characters in the authorization code. Defaults to `30`.

Expiry (in secs):
Enter the number of seconds before the authorization code expires. Defaults to `600` (ten minutes).

Additional parameters to store for this Authorization Code:
If you wish to store additional metadata with the authorization code, click **Add**, and enter the **Name** and **Value** in the dialog (for example, `Department` and `Engineering`). When additional data is set, it is then available in the **Access Token using Authorization Code** filter when the authorization code is exchanged for an access token. You can also specify the fields in this table using selectors. For more details, see the *API Gateway User Guide*.

Note

If you entered parameters for the authorization code and parameters for the access token, the data will be merged. Data in the **Access Token using Authorization Code** filter may overwrite parameters stored with the authorization code. For example, if you set `Name:John` and `Department:Engineering` in the **Authorization Request** filter, and set `Department:HR` in the **Access Token using Authorization Code** filter, the token is created with `Name:John` and `Department:HR`.

Access token settings

Configure the following fields on the **Access Token Details** tab:

Access Token will be stored here:
Click the browse button to select where to cache the access token (for example, in the default `OAuth Access Token Store`). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the

section called "Manage access tokens and authorization codes".

Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Length:
Enter the number of characters in the access token. Defaults to `54`.

Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Additional parameters to store for this Access Token:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department`, `Engineering`).

Generate Token Scopes:
When requesting a token from the Authorization Server, you can specify a parameter for the OAuth scopes that you wish to access. You can select whether the access token is generated only if the scopes in the request match all or any scopes registered for the application. Alternatively, for extra flexibility you can get the scopes by calling out to a policy.

Select one of the following options to configure how access tokens are generated based on specified scopes:

- **Get scopes from a registered application**:
  Select whether the scopes must match **Any** or **All** of the scopes registered for the application in the Client Application Registry. Defaults to **Any**. If no scopes are sent in the request, the token is generated with the scopes registered for the application.
- **Get scopes by calling policy**:
  Select a pre-configured policy to get the scopes, and enter the attribute that stores the scopes in the **Scopes approved for token are stored in the attribute** textbox. Defaults to `scopes.for.token`. The configured filter requires the scopes as set of strings on the message whiteboard.

Monitoring settings

The settings on this tab configure service-level monitoring options such as whether to store usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the API Gateway Analytics tool to produce reports on how the service is used.

Monitoring Options
For details on the **Monitoring Options** fields on this tab, see the section called "Monitoring settings" in *Get access token information*.

Record Outbound Transactions
Select whether to record outbound message traffic. You can use this setting to override the **Record Outbound Transactions** setting on the **System Settings** > **Traffic Monitor** screen. This setting is selected by default.

Authorize transaction

Overview

The OAuth 2.0 **Authorize Transaction** filter is used to authorize the Resource Owner and grant (allow/deny) client access to the resources. This supports the OAuth 2.0 Authorization Code Grant or Web server authentication flow, which is used by applications hosted on a secure server. A critical aspect of this flow is that the server must be able to protect the issued client application's secret. The Web server flow is suitable for clients capable of interacting with the end-user’s user-agent (typically a Web browser), and capable of receiving incoming requests from the Authorization Server (acting as an HTTP server).

For more details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

Template settings

Configure the following fields on the **Validation/Templates** tab:

HTML Templates:
Specify the following templates for HTML forms:

- **Login Form**:
  Enter the full path to the HTML form that the Resource Owner can use to log in. Defaults to the following:

  ```
  ${environment.VDISTDIR}/samples/oauth/templates/login.html
  ```

- **Authorization Form**:
  Enter the full path to the HTML form that the Resource Owner can use to grant (allow/deny) client access to the resources. Defaults to the following:

  ```
  ${environment.VDISTDIR}/samples/oauth/templates/requestAccess.html
  ```

`VDISTDIR` specifies the directory in which the API Gateway is installed.

Authorization code settings

Configure the following fields on the **Authz Code Details** tab:

Authorization Code will be stored here:
Click the browse button to select where to cache the access token (for example, in the default `Authz Code Store`). To add an access token store, right-click **Authorization Code Stores**, and select **Add Authorization Code Store**. You can store codes in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called “Manage access tokens and authorization codes”.

Location of Access Code redirect page:
Enter the full path to the HTML page used for the access code HTTP redirect. Defaults to the following:

```
${environment.VDISTDIR}/samples/oauth/templates/showAccessCode.html
```

Length:
Enter the number of characters in the authorization code. Defaults to `30`.

Expiry (in secs):
Enter the number of seconds before the authorization code expires. Defaults to `600` (ten minutes).

Additional parameters to store for this Authorization Code:
If you wish to store additional metadata with the authorization code, click **Add**, and enter the **Name** and **Value** in the dialog (for example, `Department` and `Engineering`). When additional data is set, it is then available in the **Access Token using Authorization Code** filter when the authorization code is exchanged for an access token. You can also specify the fields in this table using selectors. For more details, see the *API Gateway User Guide*.

Note

If you entered parameters for the authorization code and parameters for the access token, the data will be merged. Data in the **Access Token using Authorization Code** filter may overwrite parameters stored with the authorization code. For example, if you set `Name:John` and `Department:Engineering` in the **Authorize Transaction** filter, and set `Department:HR` in the **Access Token using Authorization Code** filter, the token is created with `Name:John` and `Department:HR`.

Access token settings

Configure the following fields on the **Access Token Details** tab:

Access Token will be stored here:
Click the browse button to select where to cache the access token (for example, in the default `OAuth Access Token Store`). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called “Manage access tokens and authorization codes”.

Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Length:
Enter the number of characters in the access token. Defaults to `54`.

Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Additional parameters to store for this Access Token:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department`, `Engineering`).

Generate Token Scopes:
When requesting a token from the Authorization Server, you can specify a parameter for the OAuth scopes that you wish to access. When scopes are sent in the request, you can select whether the access token is generated only if the scopes in the request match all or any scopes registered for the application. Alternatively, for extra flexibility you can get the scopes by calling out to a policy.

Select one of the following options to configure how access tokens are generated based on specified scopes:

- **Get scopes from a registered application**:
  Select whether the scopes must match **Any** or **All** of the scopes registered for the application in the Client Application Registry. Defaults to **Any**. If no scopes are sent in the request, the token is generated with the scopes registered for the application.
- **Get scopes by calling policy**:
  Select a pre-configured policy to get the scopes, and enter the attribute that stores the scopes in the **Scopes approved for token are stored in the attribute** textbox. Defaults to `scopes.for.token`. The configured filter requires the scopes as set of strings on the message whiteboard.

Monitoring settings

The settings on this tab configure service-level monitoring options such as whether to store usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the section called “Monitoring settings” in *Get access token information*.

Refresh access token

Overview

The OAuth 2.0 **Refresh Access Token** filter enables an OAuth client to get a new access token using a refresh token. This filter supports the OAuth 2.0 Refresh Token flow. After the client consumer has been authorized for access, they can use a refresh token to get a new access token (session ID). This is only done after the consumer already has received an access token using either the Web Server or User-Agent flow. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

Application validation settings

Configure the following fields on this tab:

Find client application information from message:
Select one of the following:

- **In Authorization Header**:
  This is the default setting.
- **In Query String**:
  The **Client Id** defaults to `client_id`, and **Client Secret** defaults to `client_secret`.

Access token settings

Configure the following fields on this tab:

Access Token will be stored here:
Click the browse button to select where to cache the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called “Manage access tokens and authorization codes”.

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Refresh Token Details:
Select one of the following options:

- **Generate a new refresh token:**
  Select this option to generate a new access token and refresh token pair. The old refresh token passed in the request is removed. This option is selected by default.

  Enter the number of seconds before the refresh token expires in the **Refresh Token Expiry (in secs)** field, and enter the number of characters in the refresh token in the **Refresh Token Length** field. The expiry defaults to `43200` (12 hours), and the length defaults to `46`.
- **Do not generate a refresh token:**
  Select this option to generate a new access token only. The old refresh token passed in the request is removed.

- **Preserve the existing refresh token:**
  Select this option to generate a new access token and preserve the existing refresh token. The refresh token passed in the request is sent back with the access token response.

Store additional meta data with the access token which can subsequently be retrieved:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department` and `Engineering`).

Monitoring settings

The settings on this tab configure service-level monitoring options such as whether to store usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the section called “Monitoring settings” in *Get access token information*.

Get access token using resource owner credentials

Overview

The OAuth 2.0 **Resource Owner Credentials** filter is used to directly obtain an access token and an optional refresh token. This supports the OAuth 2.0 Resource Owner Password Credentials flow, which can be used as a replacement for an existing login when the consumer client already has the user's credentials. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

Application validation settings

Configure the following fields on this tab:

Authenticate Resource Owner
Select one of the following:

- **Authenticate credentials using this repository**:
  Select one of the following from the list:
  - `Simple Active Directory Repository`
  - `Local User Store`
- **Call this policy**:
  Click the browse button to select a policy to authenticate the Resource Owner. You can use the **Policy will store subject in selector** text box to specify where the policy is stored. Defaults to the `${authentication.subject.id}` message attribute. For more details on selectors, see the *API Gateway User Guide*.

Find client application information from message:
Select one of the following:

- **In Authorization Header**:
  This is the default setting.
- **In Query String**:
  The **Client Id** defaults to `client_id`, and **Client Secret** defaults to `client_secret`.

Access token settings

Configure the following fields on the this tab:

Access Token will be stored here:
Click the browse button to select where to cache the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called "Manage access tokens and authorization codes".

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is selected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to `43200` (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to `46`.

Store additional Access Token parameters:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department` and `Engineering`).

Generate Token Scopes:
When requesting a token from the Authorization Server, you can specify a parameter for the OAuth scopes that you wish to access. You can select whether the access token is generated only if the scopes in the request match all or any scopes registered for the application. Alternatively, for extra flexibility you can get the scopes by calling out to a policy.

Select one of the following options to configure how access tokens are generated based on specified scopes:

- **Get scopes from a registered application**:
  Select whether the scopes must match **Any** or **All** of the scopes registered for the application in the Client Application Registry. Defaults to **Any**. If no scopes are sent in the request, the token is generated with the scopes registered for the application.
- **Get scopes by calling policy**:
  Select a pre-configured policy to get the scopes, and enter the attribute that stores the scopes in the **Scopes approved for token are stored in the attribute** textbox. Defaults to `scopes.for.token`. The configured filter requires the scopes as set of strings on the message whiteboard.

Monitoring settings

The settings on this tab configure service-level monitoring options such as whether to store usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the API Gateway Analytics tool to produce reports on how the service is used.

Monitoring Options
For details on the **Monitoring Options** fields on this tab, see the section called “Monitoring settings” in *Get access token information*.

Record Outbound Transactions
Select whether to record outbound message traffic. You can use this setting to override the **Record Outbound Transactions** setting on the **System Settings** > **Traffic Monitor** screen. This setting is selected by default.

Revoke token

Overview

The OAuth 2.0 **Revoke a Token** filter is used to revoke a specified OAuth 2.0 access or refresh token. A revoke token request causes the removal of the client permissions associated with the specified token used to access the user's protected resources. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. OAuth refresh tokens are tokens issued by the Authorization Server to the client that can be used to obtain a new access token.

Revoke token settings

Configure the following fields on this tab:

Token to be revoked can be found here:
Click the browse button to select the cache to revoke the token from (for example, the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called “Manage access tokens and authorization codes”.

Find client application information from message:
Select one of the following:

- **In Authorization Header**:
  This is the default setting.
- **In Query String**:
  The **Client Id** defaults to `client_id`, and **Client Secret** defaults to `client_secret`.

Monitoring settings

The settings on this tab configure service-level monitoring options such as whether to store usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the section called “Monitoring settings” in *Get access token information*.

Validate access token

Overview

The OAuth 2.0 **Validate Access Token** filter is used to validate a specified access token contained in persistent storage. OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions.

For more details on supported OAuth flows, see *API Gateway OAuth 2.0 authentication flows*.

General settings

Configure the following fields:

Name:
Enter a suitable name for this filter.

Verify access token is in cache:
Click the browse button to select the cache in which to verify access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can store tokens in a cache, in a relational database, or in an embedded Cassandra database. For more details, see the section called "Manage access tokens and authorization codes".

Location of access token:
Select one of the following:

- **In Authorization Header with prefix**:
  The access token is in the Authorization header with the selected prefix. Defaults to `Bearer`. This is the default option.
- **In query string/form body field named**:
  The access token is in the HTTP query string with the name specified in the text box.
- **In Attribute**:
  The access token is in the API Gateway message attribute specified in the text box.

Validate Scopes:
Select whether scopes match **Any** or **All** of the configured scopes in the table, and click **Add** to add an OAuth scope. The default scopes are found in `${http.request.uri}`.

For example, the default scopes used in the OAuth demos are `resource.READ` and `resource.WRITE`.

Response codes

The **Validate Access Token** filter performs a number of checks to determine if the token is valid. If any of the checks fail, the response can be examined to determine the reason for the failure.

The filter performs the following sequence of steps to determine if the token is valid:

1. Locate the token in the incoming request. The token can be in the Authorization header, in a query string, or in a message attribute.
   - If the filter is configured to find the token in a message attribute and no token is found, the following response is sent:

     ```
     HTTP/1.1 400 Bad Request
     WWW-Authenticate: Bearer realm="DefaultRealm",
     ```

```
error="invalid_request",
error_description="Unable to find token in the message."
```

- If the filter is configured to find the token in the Authorization Bearer header and no token is found (or the Authorization header is not found or does not contain the Bearer header), the following response is sent:

  ```
  HTTP/1.1 401 Unauthorized
  WWW-Authenticate: Bearer realm="DefaultRealm"
  ```

2. If the token is found in the incoming request, next verify that the token can be found in the API Gateway persistent storage mechanism. If it cannot be found, the following response is sent:

   ```
   HTTP/1.1 401 Unauthorized
   WWW-Authenticate: Bearer realm="DefaultRealm",
   error="invalid_token",
   error_description="Unable to find the access token in persistent storage."
   ```

3. If the token is found in persistent storage, next verify the authenticity of the token. This includes checking the token's expiry, client identifier, and required scopes.
   - Check if the token has expired. An expired token must not be able to allow access to a resource. If the token has expired, the following response is sent:

     ```
     HTTP/1.1 401 Unauthorized
     WWW-Authenticate: Bearer realm="DefaultRealm",
     error="invalid_token",
     error_description="The access token expired."
     ```

   - Check the client ID in the token and ensure it is the same as a client ID stored in the API Gateway client registry. (To use OAuth you need a client application and the client application must have OAuth credentials.) Check that the application is still enabled. If either checks fail, the following response is sent:

     ```
     HTTP/1.1 401 Unauthorized
     WWW-Authenticate: Bearer realm="DefaultRealm",
     error="invalid_token",
     error_description="The client app was not found or is disabled."
     ```

   - Validate the scopes in the token against the scopes configured in Policy Studio. In Policy Studio you can specify that scopes should match **Any** or **All** of the scopes listed in the table. If **All** is selected, the token scopes must match all of the scopes listed in Policy Studio. If **Any** is selected, the token scopes intersection with the scopes listed in Policy Studio must not be empty. If the scopes do not match, the following response is sent:

     ```
     HTTP/1.1 403 Forbidden
     WWW-Authenticate: Bearer realm="DefaultRealm",
     error="insufficient_scope",
     error_description="scope(s) associated with access token are not valid to access
     this resource.",
     scope="Scopes must match Any of these scopes:resource.WRITE"
     ```

     The message includes a further string listing the scopes required to access the resource.
4. If the token is authentic, allow access to the resource.

OAuth 2.0 server message attributes

Overview

Most of the OAuth 2.0 server policy filters in the API Gateway generate message attributes that can be queried further using the API Gateway selector syntax. For example, the message attributes generated by the OAuth server filters include the following:

- `accesstoken`
- `accesstoken.authn`
- `authzcode`
- `authentication.subject.id`
- `oauth.client.details`
- `scope` attributes

For more details on selectors, see the *API Gateway User Guide*.

accesstoken methods

The following methods are available to call on the `accesstoken` message attribute:

```
${accesstoken.getValue()}
${accesstoken.getExpiration()}
${accesstoken.getExpiresIn()}
${accesstoken.isExpired()}
${accesstoken.getTokenType()}
${accesstoken.getRefreshToken()}
${accesstoken.getOAuth2RefreshToken().getValue()}
${accesstoken.getOAuth2RefreshToken().getExpiration()}
${accesstoken.getOAuth2RefreshToken().getExpiresIn()}
${accesstoken.getOAuth2RefreshToken().hasExpired()}
${accesstoken.hasRefresh()}
${accesstoken.getScope()}
${accesstoken.getAdditionalInformation()}
```

The following example shows output from querying each of the `accesstoken` methods:

```
so0HlJYASrnXqn2fL2VWgiunaLfSBhWv6W7JMbmOa131HoQzZB1rNJ
Fri Oct 05 17:16:54 IST 2012
3599
false
Bearer
xif9oNHi83N4ETQLQxmSGoqfu9dKcRcFmBkxTkbc6yHDfK
xif9oNHi83N4ETQLQxmSGoqfu9dKcRcFmBkxTkbc6yHDfK
Sat Oct 06 04:16:54 IST 2012
43199
false
true
https://localhost:8090/auth/userinfo.email
{department=engineering}
```

accesstoken.authn methods

The following methods are available to call on the `accesstoken.authn` message attribute:

```
${accesstoken.authn.getUserAuthentication()}
```

```
${accesstoken.authn.getAuthorizationRequest().getScope()}
${accesstoken.authn.getAuthorizationRequest().getClientId()}
${accesstoken.authn.getAuthorizationRequest().getState()}
${accesstoken.authn.getAuthorizationRequest().getRedirectUri()}
${accesstoken.authn.getAuthorizationRequest().getParameters()}
```

The following example shows output from querying each of the `accesstoken.authn` methods:

```
admin
[https://localhost:8090/auth/userinfo.email]
SampleConfidentialApp
343dqak32ksla
https://localhost/oauth_callback
{client_secret=6808d4b6-ef09-4b0d-8f28-3b05da9c48ec,
  scope=https://localhost:8090/auth/userinfo.email, grant_type=authorization_code,
  redirect_uri=https://localhost/oauth_callback, state=null,
  code=FOT4nudbglQouujRl8oH3EOMzaOlQP, client_id=SampleConfidentialApp}
```

authzcode methods

The following methods are available to call on the `authzcode` message attribute:

```
${authzcode.getCode()}
${authzcode.getState()}
${authzcode.getApplicationName()}
${authzcode.getExpiration()}
${authzcode.getExpiresIn()}
${authzcode.getRedirectURI()}
${authzcode.getScopes()}
${authzcode.getUserIdentity()}
${authzcode.getAdditionalInformation()}
```

The following example shows output from querying each of the `authzcode` methods:

```
F8aHby7zctNRknmWlp3voe61H20Md1
sds12dsd3343ddsd
SampleConfidentialApp
Fri Oct 05 15:47:39 IST 2012
599 (expiry in secs)
https://localhost/oauth_callback
[https://localhost:8090/auth/userinfo.email]
admin
{costunit=hr}
```

oauth.client.details methods

The following methods are available to call on the `oauth.client.details` message attribute:

```
${authzcode.getCode()}
${authzcode.getState()}
${authzcode.getApplicationName()}
${authzcode.getExpiration()}
${authzcode.getExpiresIn()}
${authzcode.getRedirectURI()}
${authzcode.getScopes()}
${authzcode.getUserIdentity()}
```

The following example shows output from querying each of the `oauth.client.details` methods:

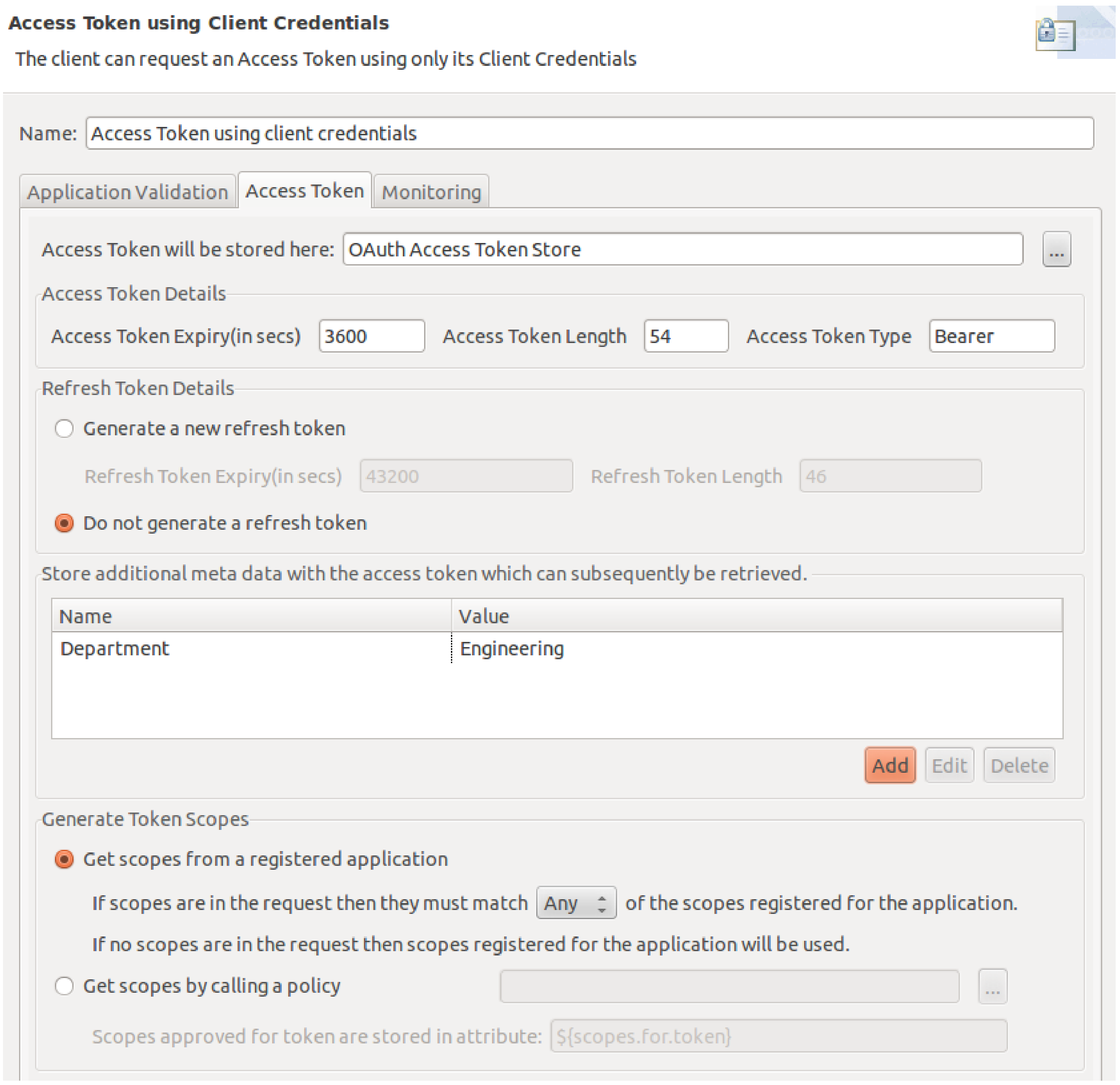
```
F8aHby7zctNRknmWlp3voe61H20Md1
sds12dsd3343ddsd
SampleConfidentialApp
Fri Oct 05 15:47:39 IST 2012
599 (expiry in secs)
https://localhost/oauth_callback
[https://localhost:8090/auth/userinfo.email]
admin
```

Example of querying a message attribute

If you add additional access token parameters to the OAuth 2.0 **Access Token Info** filter, you can return a lot of additional information about the token. For example:

```
{
   "audience" : "SampleConfidentialApp",
   "user_id" : "admin",
   "scope" : "https://localhost:8090/auth/userinfo.email",
   "expires_in" : 3567,
   "Access Token Expiry Date" : "Wed Aug 15 11:19:19 IST 2012",
   "Authentication parameters" : "{username=admin,
    client_secret=6808d4b6-ef09-4b0d-8f28-3b05da9c48ec,
    scope=https://localhost:8090/auth/userinfo.email, grant_type=password,
    redirect_uri=null, state=null, client_id=SampleConfidentialApp,
    password=changeme}",
   "Access Token Type:" : "Bearer"
```

You also have the added flexibility to add extra name/value pair settings to access tokens upon generation.The OAuth 2.0 access token generation filters provide an option to store additional parameters for an access token. For example, if you add the name/value pair `Department/Engineering` to the **Client Credentials** filter:



Access Token using Client Credentials

The client can request an Access Token using only its Client Credentials

Name: Access Token using client credentials

Application Validation | Access Token | Monitoring

Access Token will be stored here: OAuth Access Token Store

Access Token Details

Access Token Expiry(in secs)	3600	Access Token Length	54	Access Token Type	Bearer

Refresh Token Details

- ○ Generate a new refresh token
  - Refresh Token Expiry(in secs) 43200 Refresh Token Length 46
- ● Do not generate a refresh token

Store additional meta data with the access token which can subsequently be retrieved.

Name	Value
Department	Engineering

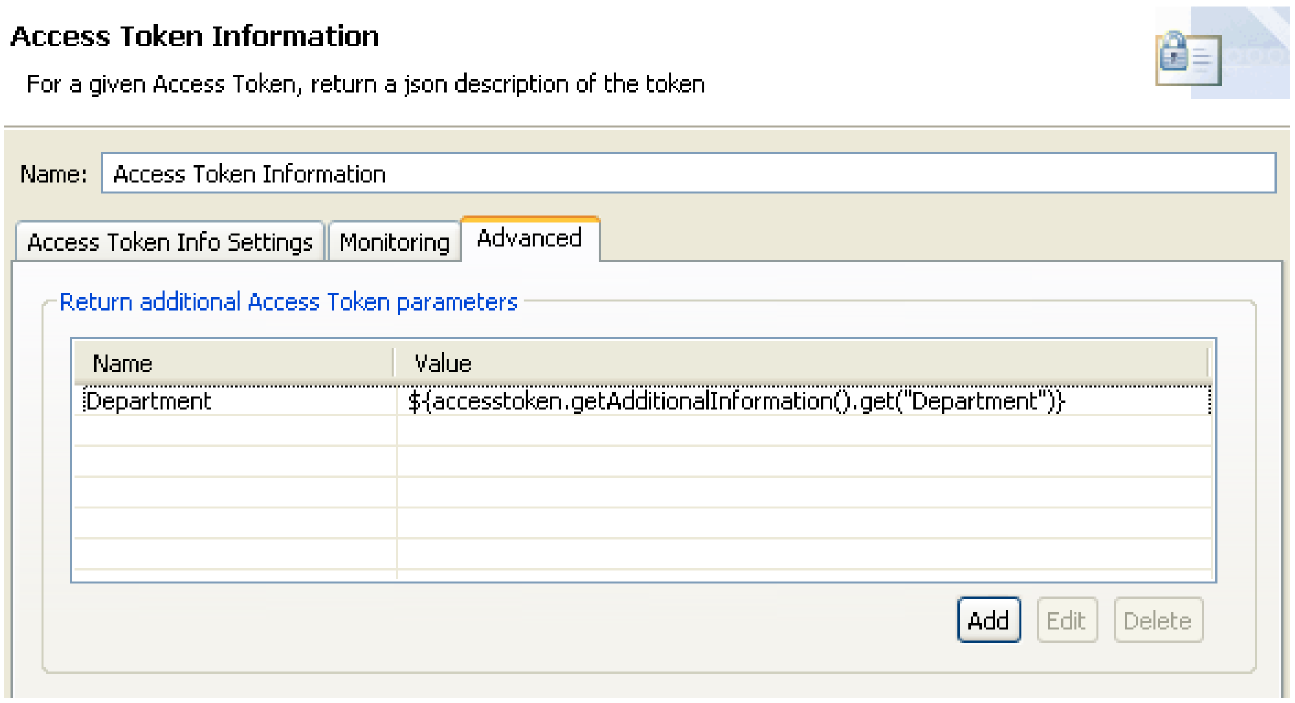
Generate Token Scopes

- ● Get scopes from a registered application
  - If scopes are in the request then they must match Any of the scopes registered for the application.
  - If no scopes are in the request then scopes registered for the application will be used.
- ○ Get scopes by calling a policy
  - Scopes approved for token are stored in attribute: ${scopes.for.token}

You can then update the **Access Token Info** filter to add a name/value pair using a selector to get the following value:

```
Department/${accesstoken.getAdditionalInformation().get("Department")}
```

For example:


Access Token Information
For a given Access Token, return a json description of the token
Access Token Info Settings
Monitoring
Advanced
Return additional Access Token parameters
Add
Edit
Delete

Name: Access Token Information

Name	Value
Department	${accesstoken.getAdditionalInformation().get("Department")}

Then the JSON response is as follows:

```
{
  "audience" : "SampleConfidentialApp",
  "user_id" : "SampleConfidentialApp",
  "scope" : "https://localhost:8090/auth/userinfo.email",
  "expires_in" : 3583,
  "Access Token Type:" : "Bearer",
  "Authentication parameters" :
  "{client_secret=6808d4b6-ef09-4b0d-8f28-3b05da9c48ec,
    scope=https://localhost:8090/auth/userinfo.email, grant_type=client_credentials,
    redirect_uri=null, state=null, client_id=SampleConfidentialApp}",
  "Department" : "Engineering",
  "Access Token Expiry Date" : "Wed Aug 15 12:10:57 IST 2012"
```

You can also use API Gateway selector syntax when storing additional information with the token. For more details on selectors, see the *API Gateway User Guide*.

OAuth scope attributes

In addition, the following message attributes are used by the OAuth filters to manage OAuth scopes. The scopes are stored as a set of strings (for example, `resource.READ` and `resource.WRITE`):

- `scopes.in.token`
  Stores the OAuth scopes that have been sent in to the Authorization Server when requesting the access token.
- `scopes.for.token`
  Stores the OAuth scopes that have been granted for the access token request.
- `scopes.required`
  Used by the **Validate Access Token** filter only. If there is a failure accessing an OAuth resource due to incorrect scopes in the access token, an `insufficent_scope` exception is sent back in the `WWW-Authenticate` header. When **Get scopes by calling a policy** is set, the configured policy can set the `scopes.required` message attrib-

ute. This enables the OAuth Resource Server to properly interact with client applications and provide useful error response messages. For example:

```
WWW-Authenticate  Bearer realm="DefaultRealm",error="insufficient_scope",
   error_description="scope(s) associated with access token are not valid
   to access this resource", scope="Scopes must match All of these scopes:
   https://localhost:8090/auth/user.photos https://localhost:8090/auth/userinfo.email"
```

Introduction to API Gateway OAuth 2.0 Client

Overview

OAuth is an open standard for authorization that enables client applications to access server resources on behalf of a specific Resource Owner. OAuth also enables Resource Owners (end users) to authorize limited third-party access to their server resources without sharing their credentials.

The API Gateway can act as the client application in an OAuth 2.0 scenario, as such the API Gateway can instigate the authorization process, handle redirects and request OAuth tokens from an authorization server. Received tokens are stored securely and subsequently used to access protected resources on behalf of users. These features provide the benefits that the oauth client burden is moved to the gateway, the resource owner’s credentials are never shared with the client application, and the access token is never shared with the resource owner’s user agent.

Note

This guide assumes that you are familiar with both the terms and concepts described in the OAuth 2.0 Authorization Framework [http://tools.ietf.org/html/rfc6749] and the OAuth Server features of the API Gateway see Chapter 1, API Gateway as an OAuth server.

API Gateway OAuth Client features

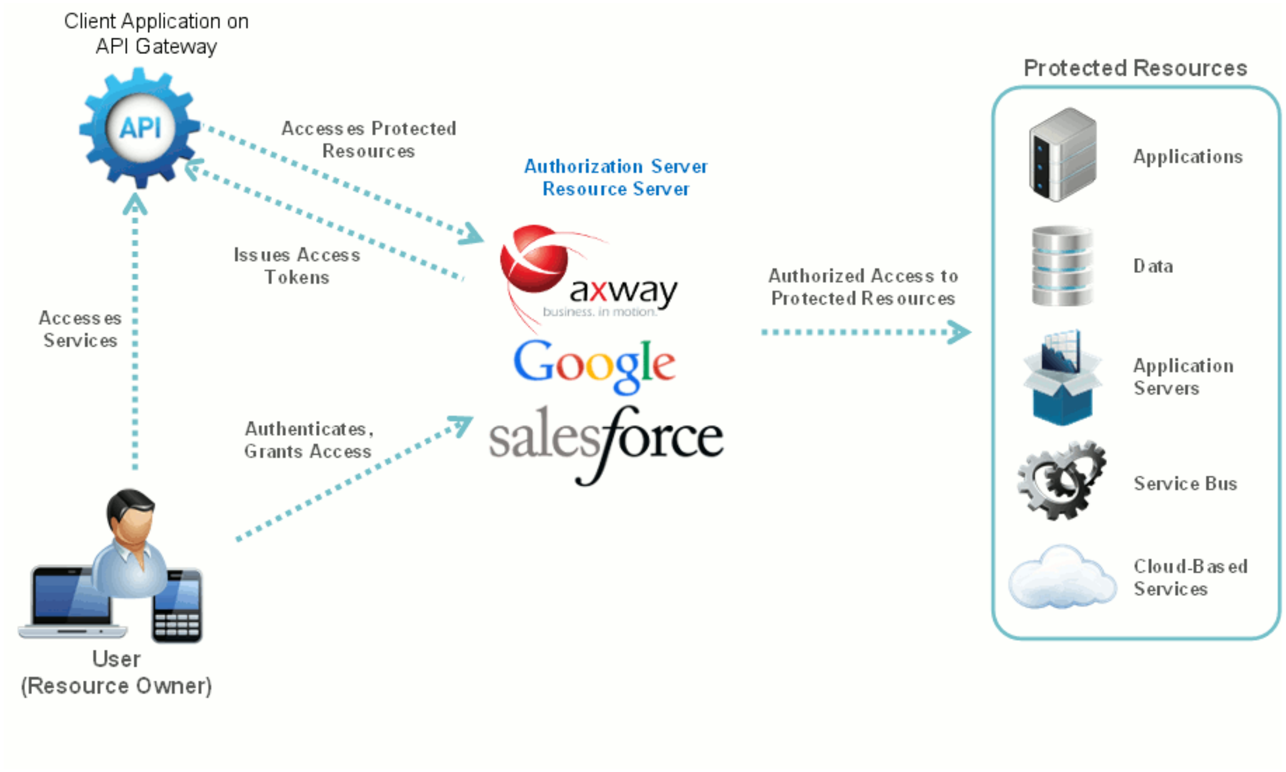
The API Gateway ships with the following features to support OAuth 2.0 Client functionality:

- Provider Profiles for defining OAuth Service providers and the applications registered therein.
- A set of pre-configured sample Provider Profiles for use with Axway, Google and Salesforce OAuth services
- Storage of received tokens
- Support for the following OAuth flows:
  - Authorization Code
  - Resource Owner Password Credentials
  - Client Credentials
  - JWT
  - SAML

Note

The Implicit grant type is not supported as it is designed to support client applications that do not have a secure server component and as such is not applicable for the API Gateway acting as an OAuth client.

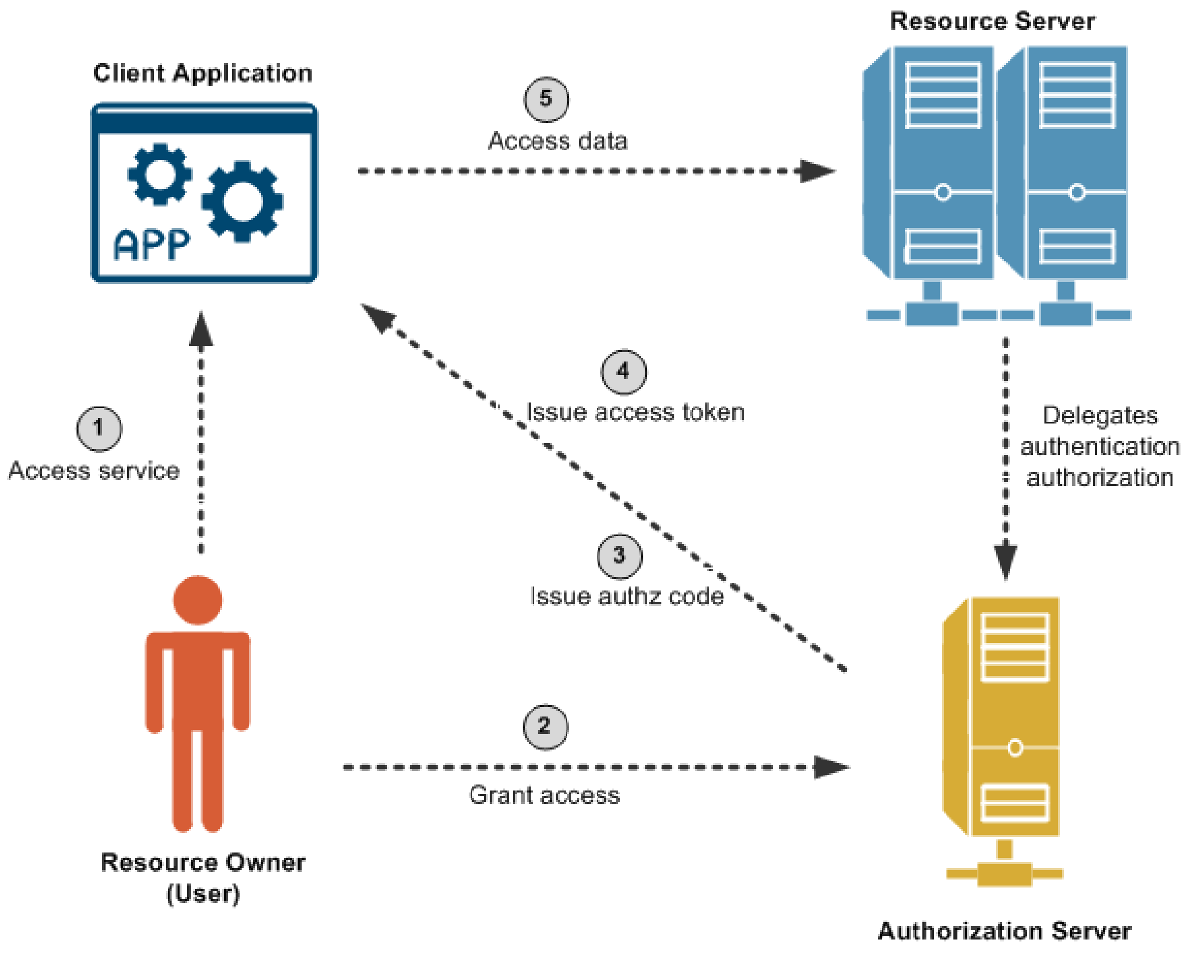
The following diagram shows the roles of the API Gateway as an OAuth 2.0 Client Application accessing OAuth services provided By the Axway API Gateway, Google and Salesforce:


Client Application on API Gateway
API
Accesses Protected Resources
Issues Access Tokens
Accesses Services
Authenticates, Grants Access
User (Resource Owner)
Authorization Server Resource Server
axway business. in motion.
Google
salesforce
Authorized Access to Protected Resources
Protected Resources
Applications
Data
Application Servers
Service Bus
Cloud-Based Services

OAuth 2.0 example client workflow

The OAuth 2.0 client is responsible for accessing the OAuth 2.0 protected resources of other servers. It is useful to consider an example workflow to understand how the API Gateway fits the role of OAuth Client, for this we use a similar example to the one in the section called “OAuth 2.0 example workflow” but in this context the API Gateway assumes the role of client, and the service provider is Google. As an oauth client the API Gateway wishes to access a calendar of a local user who maintains a personal account on Google, but it is required that the user does not reveal their Google credentials to the API Gateway.

This problem can be solved using the example OAuth 2.0 Web Server flow shown in the following diagram:



Out of band, the API Gateway pre-registers with Google and obtains a client ID/secret. The API Gateway also registers a callback URL to receive the authorization code from Google when the Resource Owner authorizes access to their calendar. The printing application has also requested access to an API named `/google/calendar`, which has an OAuth scope of `calendar`. The credentials received from Google are added to the Google Provider Profile on the API Gateway via PolicyStudio, see *Configure OAuth 2.0 client applications*. The Provider Profile is also configured with the Authorization end point and Token endpoint of the Google Authorization server. The callback URL is also created as a HTTP Listener on the API Gateway with a filter for receiving the authorisation code.

The steps in the diagram are described as follows:

1. With a User Agent (UA), such as a browser or mobile phone, the user accesses a service defined on the API Gateway. This service must access Google on the users behalf and so instigates the authorization flow by redirecting the UA to the authorization endpoint defined in the Google Provider Profile.
2. After following the redirect the user logs into their Google account and authorizes the application for the requested scope.

Note

The user has not shared their Google username and password with the API Gateway app. At this point, the Resource Owner is no longer involved in the process.

3. The authorization server then redirects the users UA to the callback url on the API Gateway along with an authorization code.
4. On receiving the authorization code the API Gateway client app can exchange this short-lived code for an access token. The client app sends another request to the Authorization Server, this time to the token endpoint, saying it has a code that proves the user has authorized it to access their calendar, and now issue the access token to be sent on to the API (Resource Server). The Authorization Server verifies the authorization code and returns an access token. The API Gateway then stores the access token in persistent storage.
5. The client app then attaches the access token to the API (Resource Server) requests, and receives the calendar information as requested.

Set up API Gateway OAuth 2.0

Overview

This chapter describes how to deploy the OAuth 2.0 client sample provided with the API Gateway. The sample demonstrate

Enable OAuth 2.0 management

The OAuth Services are not available in the basic installation. They must be deployed manually. However, there is a convenience script in $VDISTDIR/samples/scripts/oauth for deploying the OAuth 2.0 Client demo, supporting policies and sample Provider Profiles, this can be run from $VDISTDIR/samples/scripts with: Linux:

```
        ./run.sh oauth/deployOAuthConfig.py --type=clientdemo
```

Windows:

```
        run.bat oauth\deployOAuthConfig.py --type=clientdemo
```

The parameters for this script are as follows:

```
          Usage: deployOAuthConfig.py [options]

          Options:
            -h, --help            show this help message and exit
            -u USERNAME, --username=USERNAME
                                  The user to connect to the topology (default 'admin')
            -p PASSWORD, --password=PASSWORD
                                  The password for the user to connect to the topology
                                  connect user (default 'changeme')
            --port=PORT           The port Client Application registry is listening on
                                  (default 8089)
            --admin=ADMIN         The Client Application Registry admin name (default
                                  regadmin)
            --adminpw=ADMINPW     The Client Application Registry admin password
                                  (default changeme)
            --type=TYPE           The deployment type: "authzserver", "clientdemo" or
                                  "all" (default all)
            -g GROUP, --group=GROUP
                                  The group name
            -n SERVICE, --service=SERVICE
                                  The service name
```

Configure OAuth 2.0 client applications

Overview

OAuth 2.0 client credential profiles enable you to globally configure authentication settings for OAuth 2.0 as a client. An OAuth 2.0 credential profile is the combination of OAuth service provider details and a specific OAuth client application. An OAuth service provider defines the authorization and token endpoints. API Gateway includes three preconfigured OAuth providers:

- API Gateway
- Google
- SalesForce

Client applications must be registered with the service provider to obtain a client ID and secret as well as to register additional details like the OAuth flow type and redirect URL (where required). Google applications can be registered at https://cloud.google.com/console, SalesForce applications can be registered at https://www.salesforce.com, and API Gateway applications can be registered in the client application registry or API Manager (ports 8089 and 8075 respectively).

The API Gateway provider represents OAuth services running on an API Gateway. For more information on setting up the OAuth server on API Gateway, see *Set up API Gateway OAuth 2.0*. The API Gateway provider uses the existing OAuth server samples for authorization and token endpoints (`https://127.0.0.1:8089/api/oauth/authorize` and `https://127.0.0.1:8089/api/oauth/token`). The Google and SalesForce provider settings ship with the current public endpoints.

You can also add new OAuth providers. See the section called “Add OAuth 2.0 provider” for more information.

Add application

Each OAuth 2.0 provider can have multiple client application credentials. Each set of credentials represents an application that has been registered with the provider. Upon registering, the application is assigned a client ID and secret and can designate a redirect URL for receiving access codes.

To add an application for an existing OAuth 2.0 provider, click an OAuth 2.0 client credential node (for example, **Google**), and click the **Add** button on the **OAuth2 Credentials** tab of the **OAuth2 Credential Profile** window. Complete the following fields on the **Add OAuth2 Application** dialog:

Name:
Enter a suitable name for this client application.

Client ID:
This identifies the client responsible for the OAuth request. This ID is assigned by the OAuth provider.

Client Secret:
This is a confidential secret key used for authentication. This secret is assigned by the OAuth provider.

OAuth Flow Type:
Select an OAuth flow type. The options are:

- Authz Code
- Client Credentials
- JWT
- Resource Owner
- SAML

For more details on the authentication flows that API Gateway supports, see the *API Gateway OAuth 2.0 authentication flows* topic.

Redirect URL:
Enter the URL of the client's redirect endpoint (for example, `https://localhost:8088/oauth_callback`). This is the URL registered with the provider for receiving access codes via a redirect from the authorization server. This must match a listener configured on API Gateway (see Redirect URL Listener).

To configure client scopes, SAML bearer settings, JWT settings, or other advanced settings, click the appropriate tabs.

Configure scopes

You can configure the scopes that a client application can access on the **Scopes** tab. Click **Add** to add a scope. This is the set of scopes required by the application, and this list must match, or be a subset of, the required scopes registered with the OAuth provider. For more information on scopes, see the section called “Manage OAuth scopes”.

Configure SAML bearer

You can configure SAML bearers on the **SAML Bearer** tab. According to the IETF draft document SAML 2.0 Profile for OAuth 2.0 [http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-18], a SAML assertion can be used to request an access token when a client wishes to utilize an existing trust relationship, expressed through the semantics of the SAML assertion, without a direct user approval step at the authorization server. When a client application is configured to use the SAML grant type a SAML assertion must be either configured/generated or made available on the message board.

To generate an assertion select the **Generate assertion using following configuration** option and complete the following fields:

Use private key to sign SAML assertion:
Click **Signing Key** to select a private key to use to sign the assertion. This will be the private key certificate registered with the OAuth provider.

Resource Owner ID:
Enter the identity of the resource owner as expected by the resource server. This can be specified using a selector (for example, `${authentication.subject.id}`).

Assertion expires in:
Enter the time duration that the assertion is valid for. Expressed in days, hours, minutes, and seconds.

Drift time (secs):
Enter a drift time in seconds to allow for clock skew.

Alternatively, you can generate the assertion through other means and take it from the message board by selecting the option **Get assertion from message attribute named:** and entering the name of the attribute (for example, `${oauth.saml.assertion}`).

Important

The IETF draft document also describes how to use SAML 2.0 for client authentication. This is *not* supported in API Gateway.

Configure JWT

You can configure JWT on the **JWT** tab. This enables you to configure JWT for authorization grant, as defined by the IETF draft document JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants [http://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer-07].

Important

API Gateway only supports the use of JWT as authorization grant and does not support JWT for client authentication.

Configure the following fields:

Sign using private key:
Select this option and click **Signing Key** to select a private key certificate that has been registered with the OAuth provider, and use it to sign the JWT claim.

Sign using client secret:
Select this option to sign the JWT claim using a client secret issued by the OAuth provider.

JWT expiry (in secs):
Enter the expiry time for the JWT claim, in seconds.

Add additional JWT claims:
Click the **Add** button to add additional JWT claims. You can also **Edit** or **Delete** existing claims.

By default a JWT is generated with the following claim set:

Claim	Dafault Value
`iss`	The application client ID
`aud`	The token endpoint of the provider
`exp`	The expiry time from the field **JWT expiry (in secs)**
`iat`	The issued assertion time, the time the assertion was issued measured in seconds since 00:00:00 UTC, January 1, 1970.

These claims can be overridden by adding additional claims. It is also possible to add claims like `scope` to define scopes, and `prn` (for SalesForce), or `sub` (as defined in the IETF draft doc) to identify the resource owner for whom a token is being requested.

Note

Scopes must be added to a claim on this tab if they are required by the provider to be present in a claim. The scopes defined on the **Scopes** tab are added to the query string of the token request, but for flexibility they are not automatically added to the claim. The reason for this is because JWT authorization grants are non-normative and claim sets must be agreed in advance with individual OAuth providers. For example, SalesForce does not allow the addition of scopes to a JWT claim, whereas Google requires a scope claim. Automatically adding scopes from the **Scopes** tab to a claim could preclude a JWT grant flow where scopes must be present in the request but not the claim.

Configure advanced settings

You can use the following options to specify where to add the client credentials in token requests. The Authorization Header or The Query String. This option applies to all standard grant types excluding JWT and SAML.

In Authorization Header:
Select this option to add the client credentials to the authorization header.

In Query String:
Select this option to add the client credentials to the query string.

Use the following options to specify where to find resource owner credentials, for the resource owner grant type.

Resource Owner ID:
Enter the resource owner ID. This can be specified as a selector.

Resource Owner Password:
Enter the resource owner password. This can be specified as a selector.

Finally, in the **Properties** table you can add additional properties to pass with authorization or token requests. These properties can be used to set up provider-specific options, for example, Google authorization requests require the parameter `access_type=offline` to issue a refresh token.

After you have configured your OAuth 2.0 client credentials globally, you can select the client credential profile to use for authentication on the **Authentication** tab of your filter (for example, in the **Connection** and **Connect To URL** filters). For more information, see the *API Gateway User Guide*.

Add OAuth 2.0 provider

To configure a new OAuth 2.0 provider, right-click **OAuth2**, and select **Add OAuth2 Client Credential**. Complete the following fields on the **OAuth2 Provider Configuration** dialog:

Profile Name:
Enter a suitable name for this OAuth provider configuration (for example, `Google` or `Microsoft`).

Authorization Endpoint:
Enter the URL of the OAuth provider's authorization endpoint (for example, `https://accounts.google.com/o/oauth2/auth`). This is a public URL where a resource owner is directed to authorize a client application. This is used in the authorization code flow.

Token Endpoint:
Enter the URL of the OAuth provider's token endpoint (for example, `https://accounts.google.com/o/oauth2/token`). This is a public URL where a client application can request a token.

Token Store:
Click the browse button to choose an access token store. This is where received tokens are persisted.

You can configure OAuth access token stores globally under the **Libraries** node in the Policy Studio tree. These can then be selected in the **Access Token Store** field. For more details on configuring access token stores, see the section called “Manage access tokens and authorization codes”.

Store OAuth State in this Cache:
Click the browse button to choose a cache. This is where the state of an authorization request is stored. This is used in the authorization code flow to maintain state when the user is directed to the authorization server for authorization.

Tip

To change the configuration of an existing OAuth 2.0 provider, click the OAuth client credential node, and edit the settings on the **OAuth2 Provider Settings** tab of the **OAuth2 Credential Profile** window.

Creating a Callback URL listener

The callback url that is registered with an OAuth Provider is implemented very simply by creating a matching relative path in a HTTP Listener. The policy for this path needs only to add an Authorize client with server filter (see *Authorize client with server*). The filter must be configured with a reference to the relevant Provider Profile for this callback url.

Retrieve OAuth client access token from token storage

Overview

You can use the **Retrieve OAuth Client Access Token from Token Storage** filter to retrieve a stored access token from a client access token store.

Tokens received from OAuth providers are stored in a **Client Access Token Store**. You can configure client access token stores under the **Libraries > OAuth2 Stores** node in the Policy Studio tree view. Similar to an **Access Token Store**, this store can be backed by an API Gateway cache (default), a Relational Database Management System (RDBMS), or the embedded Apache Cassandra database. (For more details, see the section called “Manage access tokens and authorization codes”.)

A configured token store is associated with an OAuth provider (see the section called “Add OAuth 2.0 provider”) and is shared by all client applications registered with that provider.

These stored tokens can be retrieved by this filter by specifying the OAuth2 provider profile (the client application registered with a provider). Stored tokens are indexed by the client ID of the the client application and the authentication subject id of the current user. If `authentication.subject.id` is not available, the client ID is used for both indexes. This is valid for grant types that treat the client application as the resource owner, that is, client credentials, JWT, and SAML (when no resource owner is specified).

If a valid token is found by this filter it is placed on the message board as `oauth.client.accesstoken`, and the filter passes. If the token is expired, or there is no token found, the filter fails (expired tokens are still placed on the message board). The fail path can be used to refresh an expired token or start the process of requesting a token. The client application is also placed on the message board, under the attribute name `oauth.client.application`, for use in subsequent filters.

General settings

Configure the following general settings for the **Retrieve OAuth Client Access Token from Token Storage** filter:

Name:
Enter a suitable name for this filter.

Choose profile to be used for token request:
Click the browse button to select an OAuth 2.0 client credential profile.

Authorize client with server

Overview

You can use the **Authorize Client with Server** filter to request a token.

Depending on the grant type this filter either makes a direct request to the OAuth provider for a token (two-legged flow), or redirects the user to the provider's authorization server to authorize the client application (three-legged flow).

The two-legged flow covers all but the authorization code flow type and if successful results in a token being placed on the message board and stored in the configured provider's token store. The filter passes and the token can be used to make resource requests with the **Connect to URL** filter.

In the three-legged flow (authorization code flow) the filter redirects the user and the authorization completes when the user is directed back to the client application redirect URL that was registered with the OAuth provider. For more information, see the section called “Creating a Callback URL listener”.

If there is a token already stored for the user and client application, this filter sets the token on the message board and passes. If the token has expired but has a refresh token this filter attempts to refresh the token instead of requesting a new token or redirecting the user.

General settings

Configure the following general settings for the **Authorize Client with Server** filter:

Name:
Enter a suitable name for this filter.

Optionally use an explicit profile:
Select this option and click the browse button to explicitly select an OAuth2 client credential profile. This can be used if no preceding filter has set the application profile on the message board, or to override the existing application profile.

SSL settings

You can configure SSL settings, such as trusted certificates, client certificates, and ciphers on the **SSL** tab. For details on the fields on this tab, see the **Connect to URL** filter in the *API Gateway User Guide*.

Additional settings

The **Settings** tab allows you to configure the following additional settings:

- **Retry**
- **Failure**
- **Proxy**
- **Redirect**
- **Headers**

By default, these sections are collapsed. Click a section to expand it.

For details on the fields on this tab, see the **Connect to URL** filter in the *API Gateway User Guide*.

Refresh an OAuth client access token

Overview

OAuth 2.0 client tokens are designed to be short lived and have an expiry time, however, tokens can be issued with refresh tokens. If a token has expired and it has a refresh token, you can use the **Refresh an OAuth Client Access Token** filter to explicitly refresh the token.

General settings

Configure the following general settings for the **Refresh an OAuth Client Access Token** filter:

Name:
Enter a suitable name for this filter.

Optionally use an explicit profile:
Select this option and click the browse button to explicitly select an OAuth2 client credential profile. This can be used if no preceding filter has set the application profile on the message board, or to override the existing application profile.

SSL settings

You can configure SSL settings, such as trusted certificates, client certificates, and ciphers on the **SSL** tab. For details on the fields on this tab, see the **Connect to URL** filter in the *API Gateway User Guide*.

Additional settings

The **Settings** tab allows you to configure the following additional settings:

- **Retry**
- **Failure**
- **Proxy**
- **Redirect**
- **Headers**

By default, these sections are collapsed. Click a section to expand it.

For details on the fields on this tab, see the **Connect to URL** filter in the *API Gateway User Guide*.

OAuth 2.0 client message attributes

Overview

The OAuth 2.0 client policy filters in API Gateway generate message attributes that can be queried further using the API Gateway selector syntax. The message attributes generated by the OAuth 2.0 client filters include the following:

- `oauth.client.accesstoken`
- `oauth.client.application`
- `oauth.callback.state`

For more details on selectors, see the *API Gateway User Guide*.

oauth.client.accesstoken methods

The following methods are available to call on the `oauth.client.accesstoken` message attribute:

```
${oauth.client.accesstoken.getAuthentication()}
${oauth.client.accesstoken.getClientId()}
${oauth.client.accesstoken.getCreated()}
${oauth.client.accesstoken.isExpired()}
${oauth.client.accesstoken.hasRefresh()}
${oauth.client.accesstoken.getRefreshToken()}
${oauth.client.accesstoken.getExpiresIn()}
${oauth.client.accesstoken.getExpiryDate()}
${oauth.client.accesstoken.getParams()}
${oauth.client.accesstoken.getTokenType()}
```

The following example shows output from querying each of the `oauth.client.accesstoken` methods:

```
regadmin
 ClientConfidentialApp
 Thu Mar 06 12:34:44 GMT 2014
 false
 true
 GokdAuu706ydZtNkl92UEPmnJRNmVBJPiPVGGrEwXKz5Uh
 3599
 Thu Mar 06 13:34:43 GMT 2014
 {state=9a388d14-a0e9-4b32-9003-e322c93279dd, scope=resource.WRITE}
```

oauth.client.application methods

This attribute represents the provider profile selected in the filter. It contains the provider details, such as token and authorization endpoints, and the token store, as well as the specifics of the client application including the client ID and secret. The following methods are available to call on the `oauth.client.application` message attribute:

```
${oauth.client.application.getTokenURL()}
${oauth.client.application.getAuthentication()}
${oauth.client.application.getProviderName()}
${oauth.client.application.getAppName()}
${oauth.client.application.getClientID()}
${oauth.client.application.getFlow()}
${oauth.client.application.getClientSecret()}
${oauth.client.application.getExtraTokenRequestProps()}
${oauth.client.application.getScopes()}
${oauth.client.application.getLocationOfClientDetails()}
```

```
${oauth.client.application.getClientIdHeaderName()}
${oauth.client.application.getClientSecretHeaderName()}
${oauth.client.application.getTokenStore()}
${oauth.client.application.getToken()}
${oauth.client.application.getTokenFromStore()}
${oauth.client.application.getProvider()}
```

The following example shows output from querying each of the `oauth.client.application` methods:

```
https://127.0.0.1:8089/api/oauth/token
regadmin
API Gateway
Sample Client Authzcode App
ClientConfidentialApp
authorization_code
9cb76d80-1bc2-48d3-8d31-edeec0fddf6c
{}
[resource.WRITE]
QueryString
client_id
client_secret
an object of type com.vordel.circuit.oauth.persistence.SynchronizedClientTokenStore
an object of type com.vordel.oauth.client.store.OAuth2ClientAccessToken
an object of type com.vordel.oauth.client.store.OAuth2ClientAccessToken
an object of type com.vordel.oauth.client.providers.BaseOAuth2Provider
```

oauth.callback.state

This property is a map of string to string containing the state set before entering into a three-legged authorization code flow. The state map is stored before directing the resource owner to the provider's authorization server and is retrieved when the user is returned to the redirect URL of the client application. In its basic form it contains the authentication subject id of the local user and the client ID of the client application being authorized by the user.

This attribute is only set by the **Authorize Client with Server** filter.