Oracle® Fusion Middleware

Oracle API Gateway OAuth User Guide

11g Release 2 (11.1.2.4.0)

July 2015

ORACLE

Oracle API Gateway OAuth User Guide, 11g Release 2 (11.1.2.4.0)

27 July 2015

Contents

Preface

This document describes how to use the OAuth 2.0 and OpenID Connect features of API Gateway. It describes how to configure API Gateway as an OAuth server and as an OAuth client. It also describes the OpenID Connect support provided by API Gateway.

Who should read this document

The intended audience for this document is policy developers and system integrators who are responsible for configuring OAuth and OpenID Connect flows.

Before configuring OAuth or OpenID Connect flows in API Gateway you should understand exactly what message filters are, and how they are chained together to create a message policy. These concepts are documented in detail in the *API Gateway Policy Developer Guide*. You should also have an understanding of API Gateway concepts and features. For more information, see the *API Gateway Concepts Guide*.

How to use this document

This document should be used in conjunction with the other documents in the API Gateway documentation set.

Before you begin using the OAuth features of API Gateway, review this document thoroughly. The following is a brief description of the contents of each chapter:

OAuth and OpenID Connect concepts on page 11 – Describes OAuth 2.0 and OpenID Connect concepts.

Introduction to API Gateway OAuth 2.0 server on page 13 – Describes the features of API Gateway as an OAuth server.

API Gateway OAuth 2.0 authentication flows on page 17 – Describes the OAuth flows supported by API Gateway.

Set up API Gateway as an OAuth 2.0 server on page 43 – Describes how to set up API Gateway as an OAuth server.

API Gateway as an OAuth 2.0 authorization server on page 48 – Describes the OAuth authorization server features of API Gateway.

OAuth 2.0 authorization server filters on page 51 – Describes how to configure the OAuth authorization server filters.

API Gateway as an OAuth 2.0 resource server on page 75 – Describes the OAuth resource server features of API Gateway.

OAuth 2.0 resource server filters on page 80 – Describes how to configure the OAuth resource server filters.

API Gateway as an OAuth 2.0 client on page 83 – Describes the OAuth client features of API Gateway.

OAuth 2.0 client filters on page 93 – Describes how to configure the OAuth client filters.

API Gateway and OpenID Connect on page 100 – Describes how to use the OpenID Connect features of API Gateway.

OpenID Connect filters on page 110 – Describes how to configure the OpenID Connect filters.

OAuth 2.0 message attributes on page 113 – Describes the message attributes used in the OAuth filters.

What's new

This release of the API Gateway OAuth User Guide contains the following changes:

- New concepts section – See *OAuth and OpenID Connect concepts* on page 11.
- New section on OpenID Connect features – See *API Gateway and OpenID Connect* on page 100.
- Restructuring of the section on API Gateway as an OAuth server into two parts:
  - *API Gateway as an OAuth 2.0 authorization server* on page 48
  - *API Gateway as an OAuth 2.0 resource server* on page 75
- Changes to the OAuth authorization server filters section – The **Authorize Transaction** filter has been removed and the functionality moved to the **Authorization Code Flow** filter. See *Consume authorization requests* on page 64
- Changes to the OAuth client filters section – The **Redirect resource owner to Authz Server** and **Get OAuth Access Token** filters have been modified. See *Redirect resource owner to authorization server* on page 95 and *Get OAuth client access token* on page 94.
- Changes to the monitoring settings in all OAuth server filters.
- The section on OAuth message attributes has been moved to an appendix. See *OAuth 2.0 message attributes* on page 113.
- The default user names and passwords have been removed from all documentation for security reasons. You can obtain the default user names and passwords from your Oracle Account Manager.

OAuth and OpenID Connect concepts 1

OAuth 2.0 is a *delegation protocol* that is useful for conveying *authorization decisions* across a network of web-enabled applications and APIs. OAuth 2.0 is not an authentication protocol, however, OpenID Connect can be used along with OAuth to create an authentication and identity protocol on top of this delegation and authorization protocol.

OAuth 2.0

OAuth 2.0 is specified in the OAuth 2.0 Authorization Framework. OAuth can be used to provide:

- Delegated access
- Reduction of password sharing between users and third-parties
- Revocation of access

For example, when users share their credentials with a third-party application, the only way to revoke access from that application is for the user to change their password. However, this means that access from all other applications is also revoked. With OAuth, users can revoke access from specific applications without breaking other applications that should be allowed to continue to act on their behalf.

OAuth achieves this by introducing an authorization layer and separating the role of the client from that of the resource owner. OAuth defines four primary roles:

- Resource owner (RO): The entity that is in control of the data exposed by an API (for example, an end user).
- Client: The mobile application, web site, and so on, that wants to access data on behalf of the resource owner.
- Authorization server (AS): The Security Token Service (STS) or OAuth server that issues tokens.
- Resource server (RS): The service that exposes the data (for example, an API).

The client requests access to resources controlled by the resource owner and hosted by the resource server, and is issued a different set of credentials than those of the resource owner. Instead of using the resource owner's credentials to access protected resources, the client obtains an access token - a string denoting a specific scope, lifetime, and so on. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.

OAuth defines two kinds of tokens:

- Access tokens: These tokens are presented by a client to the resource server (for example, an API), to get access to a protected resource.

- Refresh tokens: These are used by the client to get a new access token from the authorization server (for example, when the access token expires).

OAuth tokens can include a scope. Scopes are like permissions or rights that a resource owner delegates to a client, so that they can perform certain actions on their behalf. A client can request specific rights, but a user might only grant a subset, or might grant others that were not requested. The OAuth specification does not define specific scopes, meaning that you can use any string to represent an OAuth scope.

OAuth defines several different flows or message exchange patterns. The most commonly used is the authorization code (web server) flow. For more details on this flow and the other flows that API Gateway supports, see *API Gateway OAuth 2.0 authentication flows* on page 17.

OpenID Connect 1.0

OpenID Connect is specified in the OpenID Connect 1.0 specification. OpenID Connect builds on the OAuth protocol and defines an interoperable way to use OAuth 2.0 to perform user authentication. OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It enables clients to verify the identity of the user based on the authentication performed by an authorization server, as well as to obtain basic profile information about the user.

OpenID Connect defines the following roles:

- Relying party (RP): An OAuth client that supports OpenID Connect. The mobile application, web site, and so on, that wants to access data on behalf of the resource owner.
- OpenID provider (OP): An OAuth authorization server that is capable of authenticating the user and providing claims to a relying party about the authentication event and the user.

OpenID Connect defines a new kind of token, the ID token. The OpenID Connect ID token is a signed JSON Web Token (JWT) that is given to the client application alongside the regular OAuth access token. The ID token contains a set of *claims* about the authentication session, including an identifier for the user (`sub`), an identifier for issuer of the token (`iss`), and the identifier of the client for which this token was created (`aud`). Since the format of the ID token is known by the client, it can parse the content of the token directly.

In addition to the claims in the ID token, OpenID Connect defines a standard protected resource (the UserInfo endpoint) that contains claims about the current user. OpenID Connect defines a set of standardized OAuth scopes that map to claims (`profile`, `email`, `phone`, and `address`). If the end user authorizes the client to access these scopes, the OP releases the associated data (claims) to the client when the client calls the UserInfo endpoint. OpenID Connect also defines a special `openid` scope that switches the OAuth server into OpenID Connect mode.

Introduction to API Gateway OAuth 2.0 server 2

OAuth 2.0 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 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.

API Gateway OAuth concepts

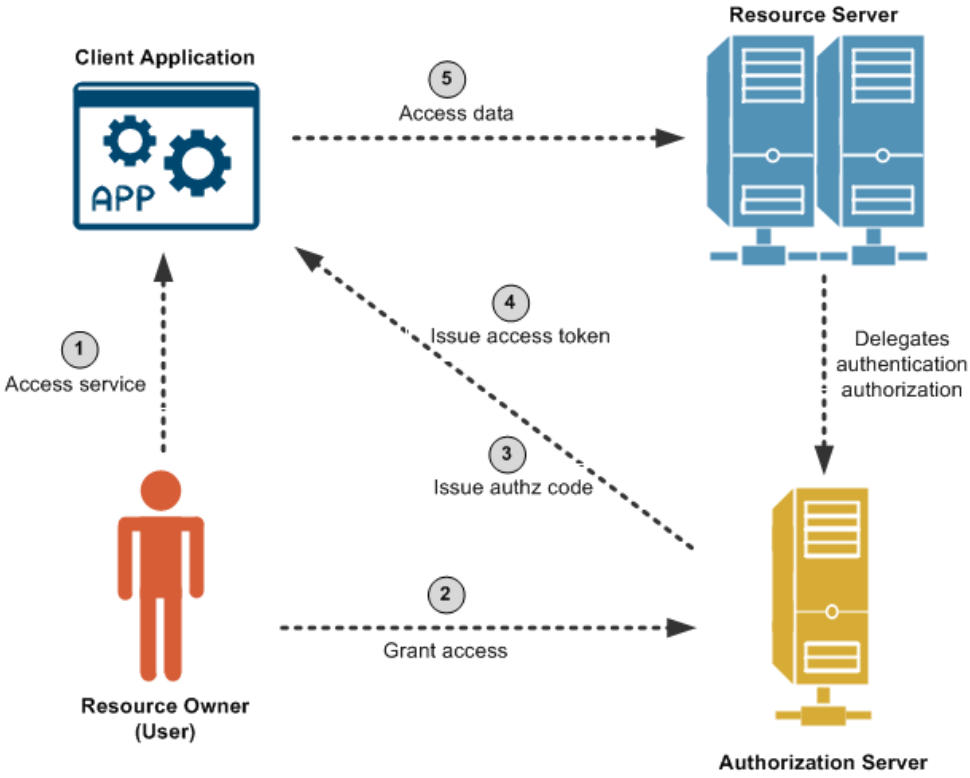
The API Gateway uses the following definitions of basic OAuth 2.0 terms:

- 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 server 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 preregisters with Flickr and obtains a client ID and 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 application redirects you to the Flickr OAuth authorization server. You must already have a Flickr account.

2. You log in to your Flickr account, and the Flickr authorization server asks you "Do you want to allow the Canon printing application to access your photos?" You click **Yes** to authorize.
3. When successful, the printing application receives an authorization code at the callback URL that was preregistered out of band.

   Note You have not shared your Flickr user name and password with the printing application. At this point, you as resource owner are no longer involved in the process.

4. The client application gets the authorization code, and must exchange this short-lived code for an access token. The client application 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 application sends the access token to the API (resource server), and receives the photos as requested.

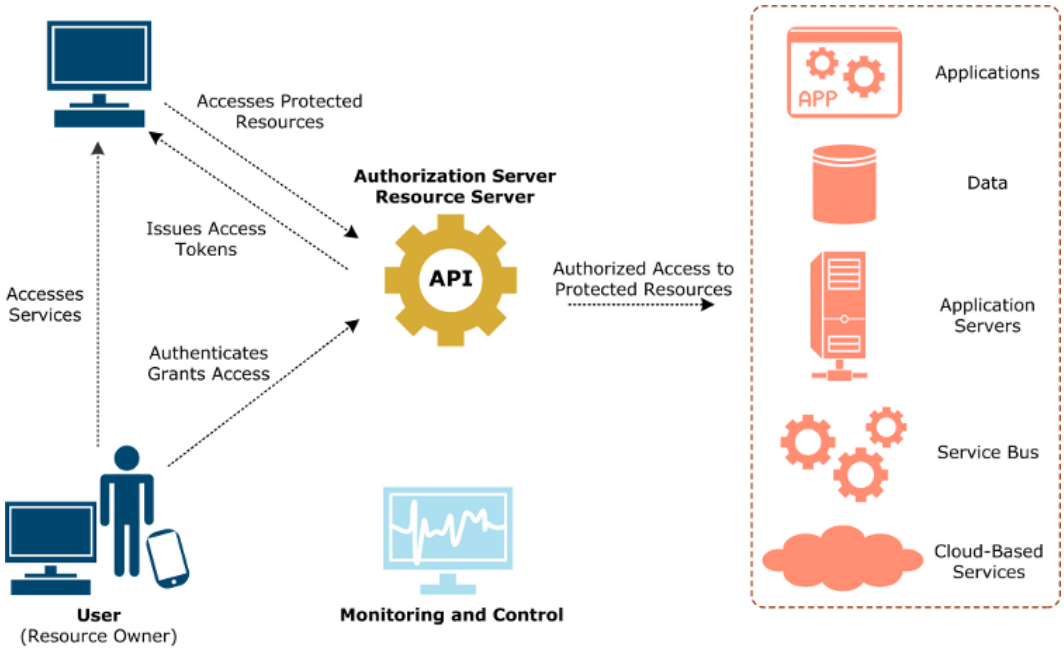
API Gateway OAuth server features

API Gateway provides 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 authentication flows:
  - Authorization code grant (web server)
  - Implicit grant (user agent)
  - Resource owner password credentials
  - Client credentials grant
  - JWT
  - Refresh token
  - Revoke token
  - Token information service
  - SAML assertion

  For more information on the supported flows, see *API Gateway OAuth 2.0 authentication flows on page 17*.
- 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:



Further information

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

- *API Gateway OAuth 2.0 authentication flows* on page 17
- *Set up API Gateway as an OAuth 2.0 server* on page 43
- *API Gateway as an OAuth 2.0 authorization server* on page 48
- *API Gateway as an OAuth 2.0 resource server* on page 75

For more details on the OAuth 2.0 specification, go to:

OAuth 2.0 Authorization Framework

API Gateway OAuth 2.0 authentication flows 3

API Gateway can use the OAuth 2.0 protocol for authentication and authorization. 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 section describes each of the supported OAuth 2.0 flows in detail, and shows how to run sample scripts demonstrating the flows.

The API Gateway supports the following authentication flows:

- 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.
- 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.
- Resource owner password credentials – This user name and password authentication flow can be used when the client application already has the resource owner's credentials.
- Client credentials grant – This user name and 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.
- 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.
- 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.
- 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.
- Token information service – The OAuth token information service responds to requests for information on a specified OAuth 2.0 access token.
- SAML assertion – The OAuth 2.0 **Access Token using SAML Assertion** filter enables an OAuth client to request an access token using a SAML assertion. This flow 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.

Run the sample scripts

API Gateway includes sample Jython scripts for all supported OAuth flows in the following directory of your API Gateway installation:

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

To run a sample script:

1. Open a UNIX shell or DOS command prompt in the following directory:

   ```
   INSTALL_DIR/samples/scripts
   ```

2. Use the `run.sh` or `run.bat` utility to execute the appropriate script.

The following examples show how to run the `implicit_grant.py` sample script:

Windows

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

UNIX/Linux

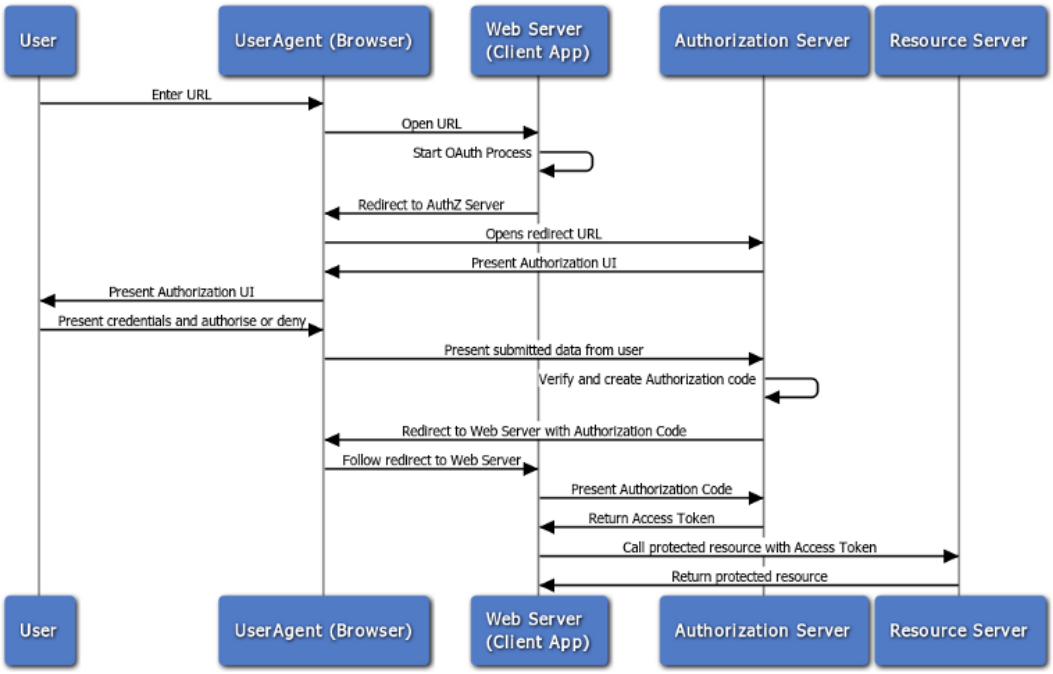
```
sh run.sh oauth/implicit_grant.py
```

Authorization code grant (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 the user's data.



Obtain an access token

This section details the steps for obtaining an access token.

Web server redirects user to authorization endpoint

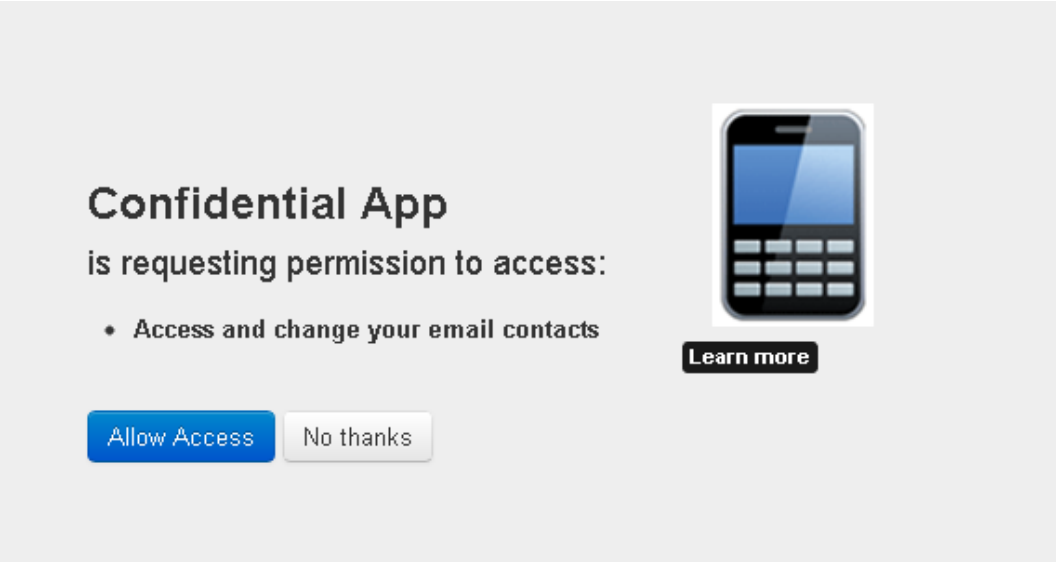
Redirect the user to the authorization endpoint with the following parameters:

Parameter	Description
`response_type`	Required. Must be set to `code`.
`client_id`	Required. The client ID generated when the application was registered in the Client Application Registry.
`redirect_uri`	Optional. The location where the authorization code will be sent. This value must match one of the values provided in the 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 window.



Web server receives callback with authorization code

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
```

Web server exchanges authorization code for access token

After the web server receives the authorization code, it can 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`.

Parameter	Description
code	Required. The authorization code received in the redirect above.
redirect_uri	Required. The redirect URL registered for the application during application registration.
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

Note 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%3A8090%2Fauth%2Fredirect.html
 &grant_type=authorization_code
 &format=query
```

Web server receives access token

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 can be used to obtain a new access token.
expires	The remaining lifetime on the access token.

Parameter	Description
type	Indicates the type of token returned. 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",
}
```

Web server uses access token to access protected resources

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 the INSTALL_DIR/samples/scripts directory.
2. 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
```

3. Copy the URL into a browser, and perform the following steps as prompted:
   - Provide login credentials to the authorization server. The default Client Application Registry user name is `regadmin`.
   - When prompted, grant access to the client application to access the protected resource.

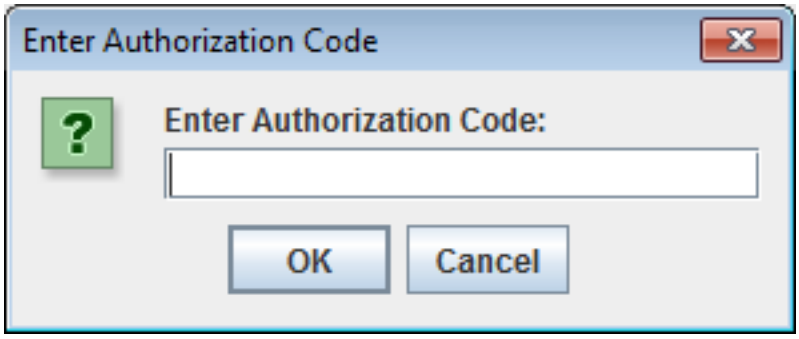
   After the resource owner has authorized and approved access to the application, the authorization server redirects a fragment containing the authorization code to the redirection URI. For example:

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

   In this example, the authorization code is:

   ```
   AaI5Or3RYB2uOgiyqVsLs1ATIY0ll0
   ```

4. Enter this value into the **Enter Authorization Code** dialog.



The script exchanges the authorization code for an access token, and then accesses 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:

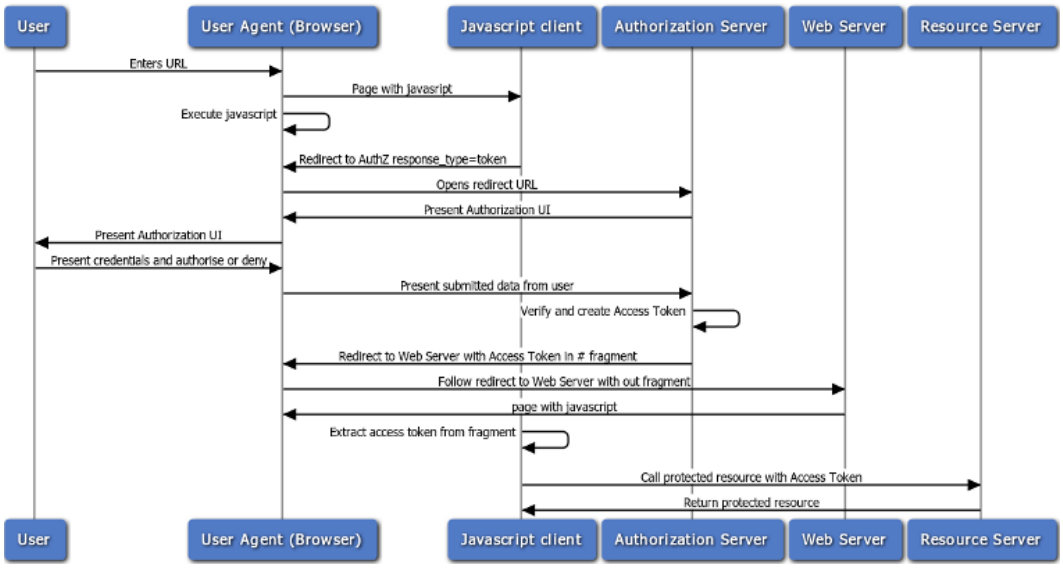
- *Get access token using authorization code* on page 53
- *Consume authorization requests* on page 64

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.



Obtain an access token

This section details the steps for obtaining an access token.

Web server redirects user to authorization endpoint

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

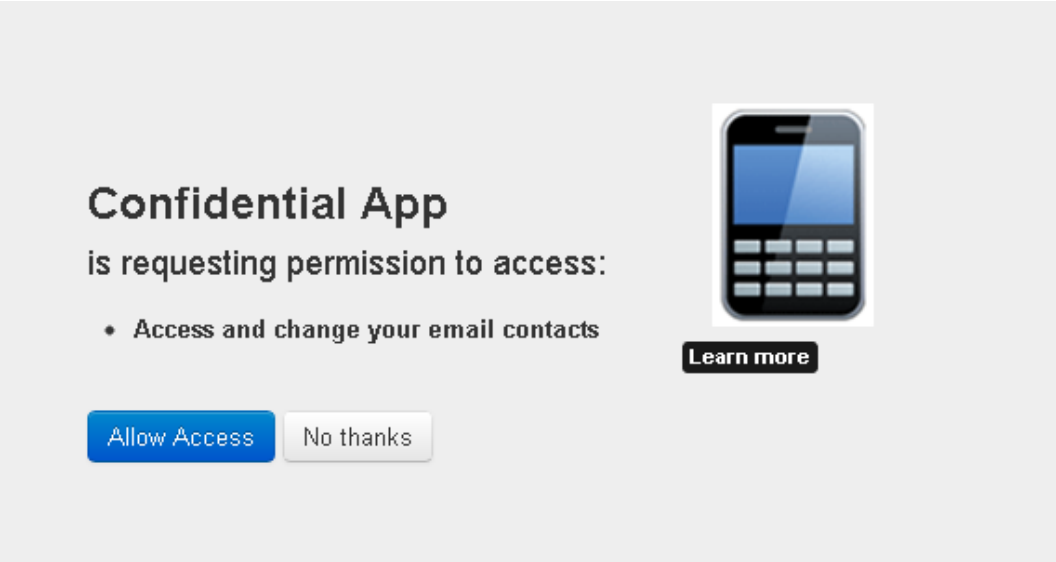
Parameter	Description
`response_type`	Required. Must be set to `token`.
`client_id`	Required. The client ID generated when the application was registered in the Client Application Registry.
`redirect_uri`	Optional. The location where the access token will be sent. This value must match one of the values provided in the 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 window.



Web server receives callback with access token

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.

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. This field always has a value of `Bearer`.

Parameter	Description
`state`	Optional. If the client application sent a value for state in the original authorization request, the state parameter is populated with this value.

Web server uses access token to access protected resources

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 the `INSTALL_DIR/samples/scripts` directory.
2. 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
   ```

   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

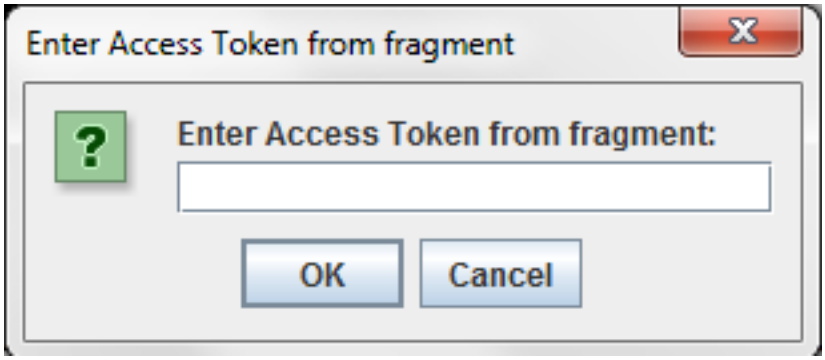
example:

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

In this example, the access token is:

```
4owzGyokzLLQB5FH4tOMk7Eqf1wqYfENEDXZ1mGvN7u7a2Xexy2OU9
```

3. Enter this value into the **Enter Access Token from fragment** dialog.



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

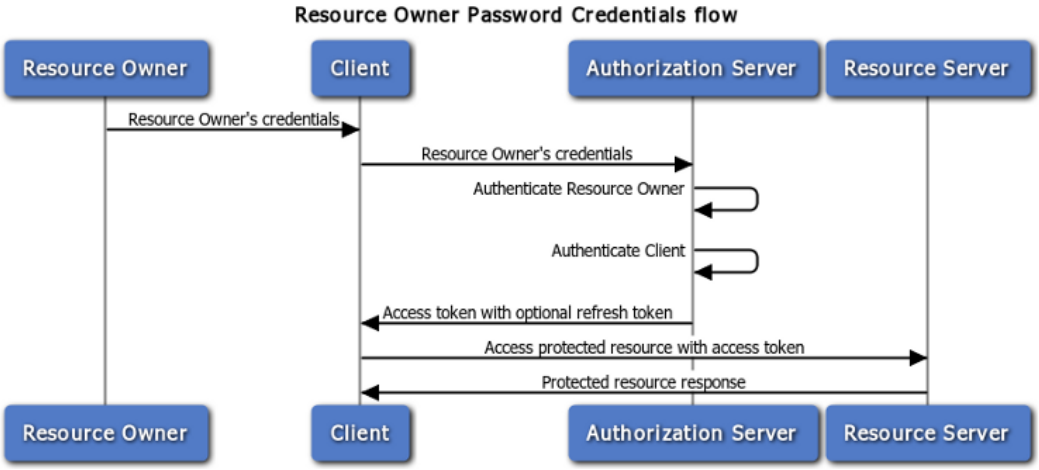
For details on the API Gateway filter that supports this flow, see *Consume authorization requests* on page 64.

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 (user name 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.



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.

Parameter	Description
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 czZCaGRSa3F0MzpnWDFmQmF0M2JW
 grant_type=password&username=johndoe&password=A3ddj3w
```

Handle the response

The API Gateway validates the resource owner's credentials and authenticates the client against the 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:

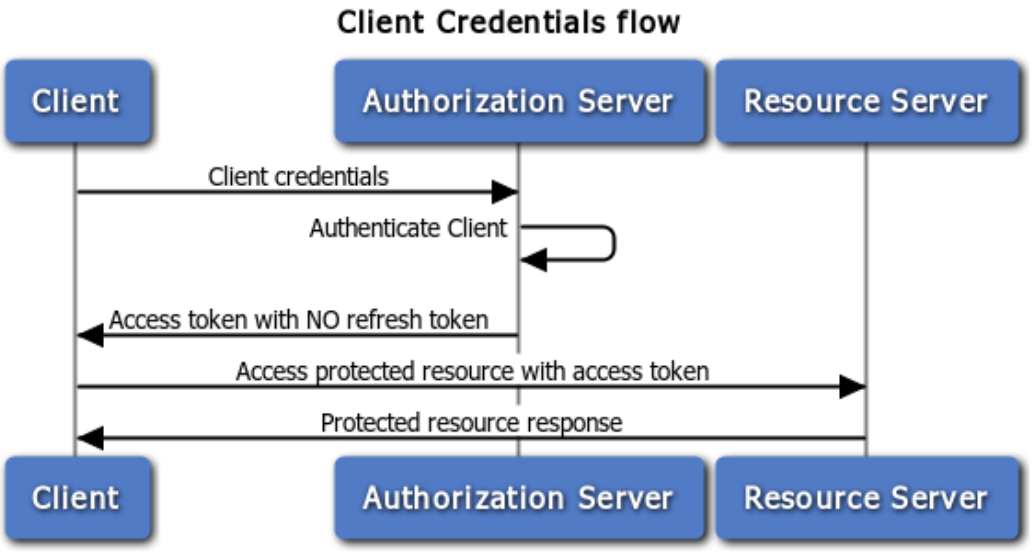
```
Sending up access token request using grant_type set to password
Response from access token request:200
Parsing the json response
**********ACCESS TOKEN RESPONSE**********
Access token received from authorization server lrGHhFhFwSmycXStIza1jjvXlSaac9
 JNIgviF7oPiV8OnxlSIsrxVA
Access token type received 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* on page 70.

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 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 script outputs the following:

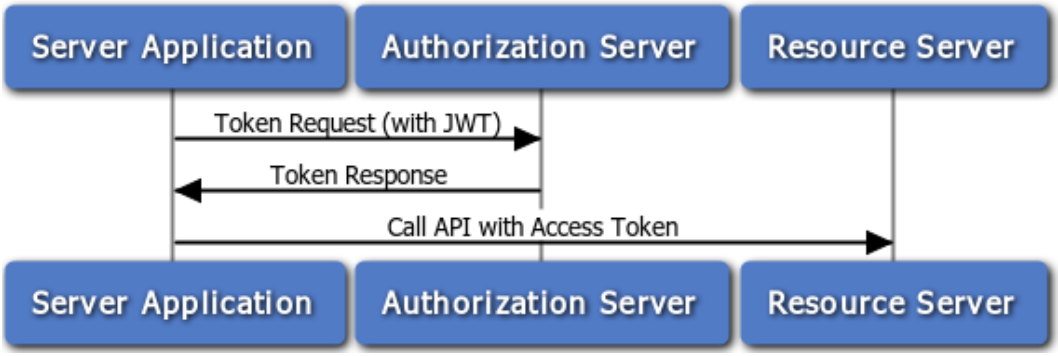
```
Sending up access token request using grant_type set to client_credentials
Response from access token request:200
Parsing the json response
**********ACCESS TOKEN RESPONSE**********
Access token received from authorization server OjtVvNusLg2ujy3a6IXHhavqdE
 PtK7qSmIj9fLl8qywPyX8bKEsjqF
Access token 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* on page 55.

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 Client Application Registry. The API Gateway uses this certificate to verify the signature of the JWT claim.

For more details on the OAuth 2.0 JWT flow, go to:
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, 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-token

   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.eyJpc3MiOiAiU2FtcGxlQ29uZmlkZW50aWFsQXBwIiwgImF1ZCI6ICJ
odHRw

Oi8vYXBpc2VydmVyL2FwaS9vYXV0aC90b2tlbiIsICJleHAiOiAiMTM0MTM1NDYwNSIsICJpYXQi
OiAi
 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.eyJpc3MiOiAiU2FtcGxlQ29uZmlkZW50aWFsQXBwIiwgImF1ZCI6ICJ
odHR

wOi8vYXBpc2VydmVyL2FwaS9vYXV0aC90b2tlbiIsICJleHAiOiAiMTM0MTM1NDYwNSIsICJpYXQ
iOiA

iMTM0MTM1NDMwNSJ9.ilWR8O8OlbQtT5zBaGIQjveOZFIWGTkdVC6LofJ8dN0akvvD0m7IvUZtPp
4dx3
 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`.
`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=eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiAiU2FtcGxlQ29uZmlkZW50a
 WFsQXBwIiwgImF1ZCI6ICJodHRwOi8vYXBpc2VydmVyL2FwaS9vYXV0aC90b2tlbiIsI
 CJleHAiOiAiMTM0MTM1NDYwNSIsICJpYXQiOiAiMTM0MTM1NDMwNSJ9.ilWR8O8OlbQt
 T5zBaGIQjveOZFIWGTkdVC6LofJ8dN0akvvD0m7IvUZtPp4dx3KdEDj4YcsyCEAPhfop
 UlZO3LE-iNPlbxB5dsmizbFIc2oGZr7Zo4IlDf92OJHq9DGqwQosJ-s9GcIRQk-IUPF
 4lVy1Q7PidPWKR9ohm3c2gt8
```

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 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:

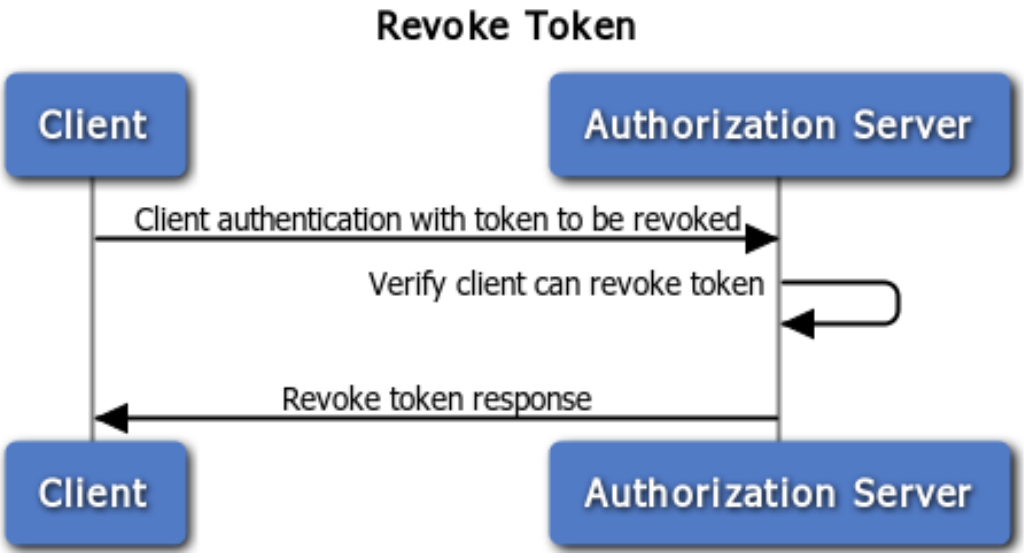
```
> run oauth/jwt.py
```

Further information

For details on the API Gateway filter that supports this flow, see *Get access token using JWT* on page 58.

Revoke token

In some cases a user might 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://HOST:8089/api/oauth/revoke`

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

Parameter	Description
`token`	Required. A token to be revoked (for example, `4eclEUX1N6oVIOoZBbaDTI977SV3T9KqJ3ayOvs4gqhGA4`).
`token_type_hint`	Optional. A hint specifying the token type. For example, `access_token` or `refresh_token`.

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
&token_type_hint=refresh_token
```

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
```

Paste the value associated with the token in the dialog:

Select the type of token hint in the next dialog:

If you select `none`, no `token_type_hint` parameter is specified in the POST request. If you select `access_token`, the POST request contains `token_type_hint=access_token`. If you select `refresh_token`, the POST request contains `token_type_hint=refresh_token`.

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 the token was revoked successfully or if an invalid token was submitted.
- HTTP 401 if client authentication failed.
- HTTP 403 if the client is not authorized to revoke the token.

The following is an example response:

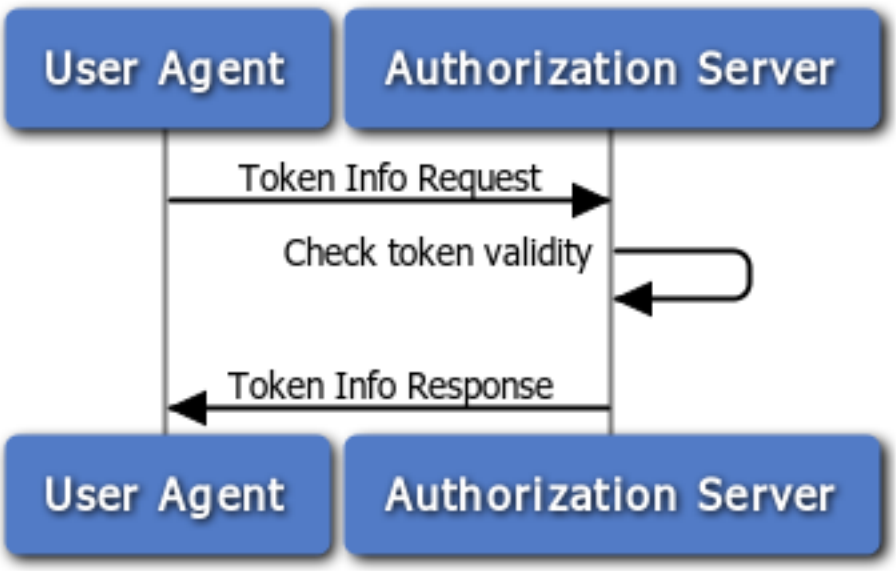
```
Token to be revoked:3eXnUZzkODNGb9D94Qk5XhiV4W4gu9muZ56VAYoZiot4WNhIZ72D3
Revoking token..............
Response from revoke token request is:200
Successfully revoked token
```

Further information

For details on the API Gateway filter that supports this flow, see *Revoke token* on page 73.

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://HOST:8089/api/oauth/tokeninfo`

Getting information about a token from the authorization server only requires a `GET` request to the token info 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=4eclEUX1
N6oVIOoZBbaDTI977SV3T9KqJ3ayOvs4gqhGA4
```

Run the sample client

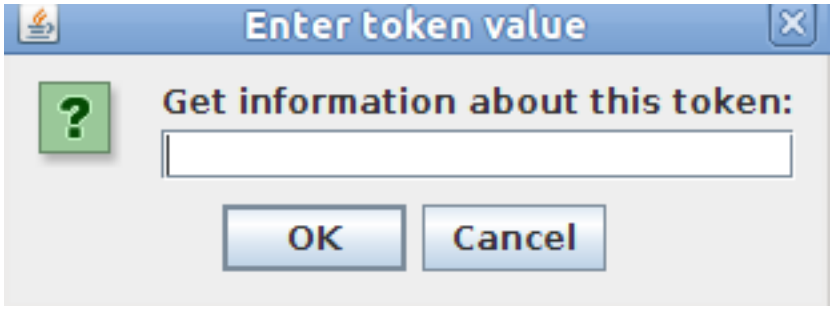
The following Jython sample client creates a token revoke request to the authorization server:

`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 information 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 message attributes* on page 113.

Further information

For details on the API Gateway filter that supports this flow, see *Get access token information* on page 52.

Set up API Gateway as an OAuth 2.0 server 4

This section describes how to set up API Gateway as an OAuth authorization server and OAuth resource server. It describes the following:

- How to enable the OAuth 2.0 endpoints used to manage client applications – See *Enable OAuth management* on page 43.
- How to import the preregistered sample client applications provided with API Gateway – See *Import sample client applications* on page 45.
- How to migrate existing OAuth 2.0 client applications – See *Migrate existing client applications* on page 46.

Enable OAuth management

The OAuth service is not available in the basic installation. You must deploy it manually. A convenience script is provided in `$VDISTDIR/samples/scripts/oauth` for deploying the OAuth 2.0 services listener, supporting policies, and sample applications. `VDISTDIR` is the directory in which API Gateway is installed.

To deploy the OAuth service, change directory to `$VDISTDIR/samples/scripts`, and run the `deployOAuthConfig.py` script as follows:

UNIX/Linux

```
./run.sh oauth/deployOAuthConfig.py
```

Windows

```
run.bat oauth\deployOAuthConfig.py
```

This deploys the OAuth server components on port 8089 and the client demo on port 8088.

The parameters for the script are as follows:

Option	Description
-h, --help	Display help for the script.
-u USERNAME, -- username=USERNAME	The administrator user name to use to connect to the topology. This is the administrator user name you entered during API Gateway installation.

Option	Description
-p PASSWORD, --password=PASSWORD	The password for the administrator user to connect to the topology. This is the administrator password you entered during API Gateway installation.
--port=PORT	The port the Client Application Registry is listening on. The default is `8089`.
--admin=ADMIN	The administrator user name for the Client Application Registry. The default is `regadmin`.
--adminpw=ADMINPW	The administrator password for the Client Application Registry.
--type=TYPE	The deployment type. The options are: • authzserver • clientdemo • all The default is `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://HOST:8089/api/oauth/authorize
Token Endpoint (REST API)	https://HOST:8089/api/oauth/token
Token Info Endpoint (REST API)	https://HOST:8089/api/oauth/tokeninfo
Revoke Endpoint (REST API)	https://HOST:8089/api/oauth/revoke
Client Application Registry (HTML Interface)	https://HOST:8089
Client Application Registry (REST API)	https://HOST:8089/api/kps/ClientApplicationRegistry

In this table, HOST refers to the machine on which API Gateway is installed.

Note To enable these endpoints, you must first enable the OAuth listener port in the API Gateway. For more details, see *Enable OAuth endpoints* on page 45.

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 sample client applications

The API Gateway ships with a number of preregistered sample client applications. For example, the default sample client applications include the following:

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

Note The sample client applications are for demonstration purposes only and should be removed before moving the authorization server into production.

If you have used the `deployOAuthConfig.py` script to deploy the OAuth service, these samples are already imported. For more information on using the `deployOAuthConfig.py` script, see *Enable OAuth management* on page 43.

If you did not use the `deployOAuthConfig.py` script, the samples are not imported, and you must import them manually into the Client Application Registry. Perform the following steps:

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

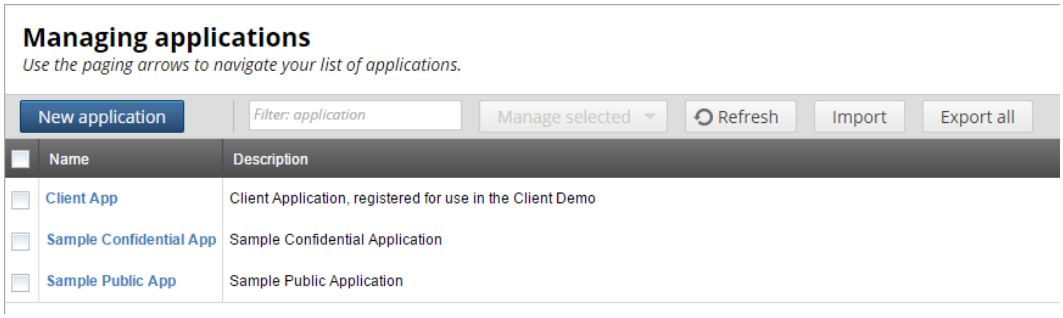
   ```
   https://localhost:8089
   ```

2. Enter the Client Application Registry user name and password. The default user name is `regadmin`.
3. Click the **Import** button at the top right of the window.
4. Select the following sample file in the dialog:

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

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

5. You can also enter a **Decryption Secret** in the dialog. However, the `sampleapps.dat` file is in plain text format, and does not require a password.
6. Click **OK** to import the sample applications. The following figure shows these applications imported into the Client Application Registry:



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
```

Edit this script to configure your user credentials and file location.

Migrate existing client applications

If you are migrating from API Gateway version 11.1.2.0.x, 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 *Import sample client applications* on page 45. This script has a `--password` parameter that you can use to encrypt the exported data for transport.

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

1. After installing API Gateway 11.1.2.4.0, 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 Gateway11.1.2.0.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 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` into a running API Gateway 11.1.2.4.0 using the Client Application Registry web interface. For more details, see *Import sample client applications* on page 45.

   Note When importing encrypted data, you must enter the password in the **Decryption Secret** field.

API Gateway as an OAuth 2.0 authorization server 5

This section describes how to configure API Gateway as an OAuth authorization server. It describes the following:

- How to manage OAuth access tokens and authorization codes – See *Manage access tokens and authorization codes* on page 48.

Related topics

- *OAuth 2.0 authorization server filters* on page 51
- *OAuth 2.0 message attributes* on page 113

Manage access tokens and authorization codes

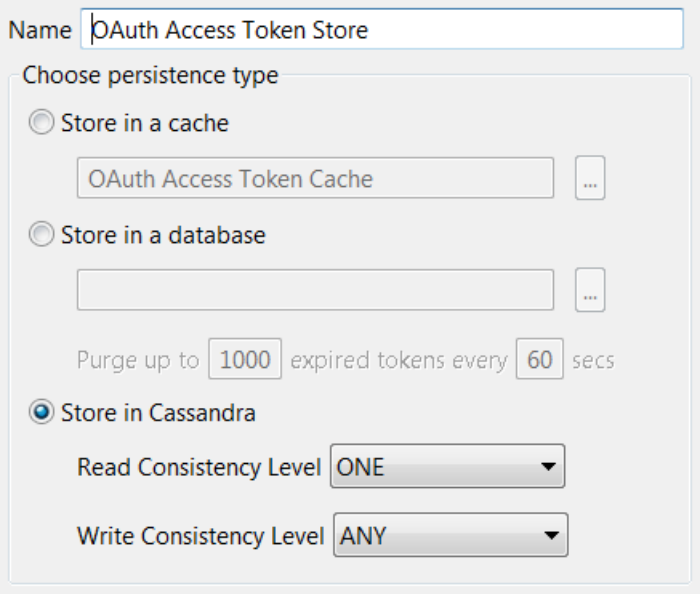
API Gateway can store generated authorization codes and access tokens in its caches, in an embedded database, or in a relational database. The authorization server issues tokens to clients on behalf of a resource owner. These tokens are used when authenticating subsequent API calls to the resource server.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

You can configure authorization code and access token stores under the **Libraries > OAuth2 Stores** node in the Policy Studio tree. 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 figure shows these options in Policy Studio:



The **Purge expired tokens every** setting enables you to configure a time interval in seconds after which a background process polls the database looking for expired access or refresh tokens or authorization codes and purges them.

Store in a cache

To store access tokens or authorization codes 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 the dialog, 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`).

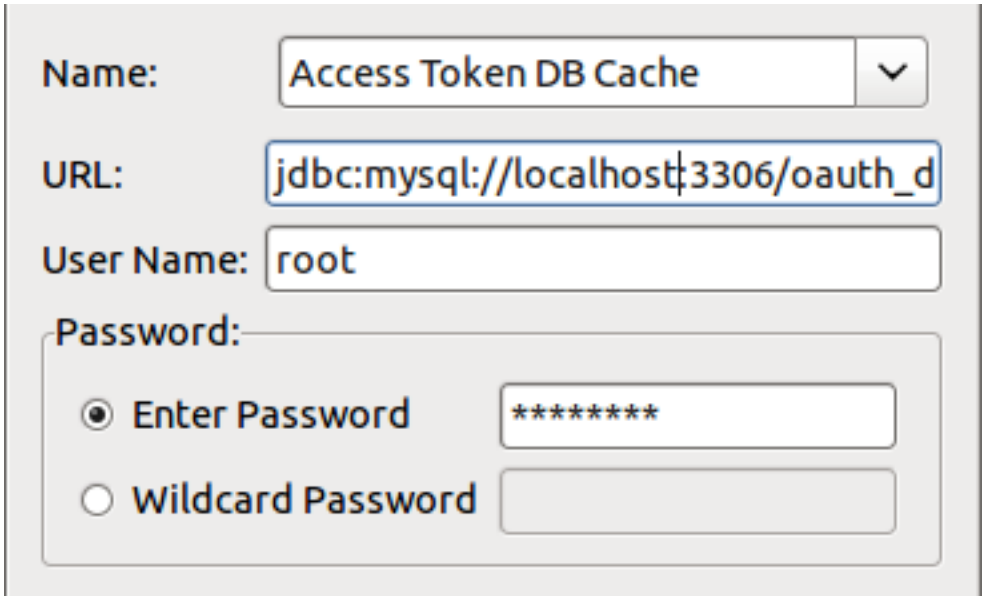
For more details on API Gateway caches, see the *API Gateway Policy Developer Guide*.

Store in a relational database

To store access tokens or authorization codes 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_TYPE\oauth-server.sql` where `DBMS_TYPE` is the database management system being used. Schema are provided for Microsoft SQL Server, MySQL, Oracle, and IBM DB2.
2. Right-click **Access Token Stores** in the Policy Studio tree, and select **Add Access Token Store**.
3. In the dialog, 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 Policy Developer Guide*.



Store in Cassandra

To store access tokens or authorization codes in Apache 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 a dialog. 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 on Apache Cassandra, see the *API Gateway Key Property Store User Guide*.

OAuth 2.0 authorization server filters 6

This section describes the filters you can use when API Gateway is acting as an OAuth authorization server. These include:

Filter	Related flows
Consume authorization requests on page 64	• *Authorization code grant (or web server) flow* on page 18 • *Implicit grant (or user agent) flow* on page 24
Get access token using authorization code on page 53	*Authorization code grant (or web server) flow* on page 18
Get access token using resource owner credentials on page 70	*Resource owner password credentials flow* on page 29
Get access token using client credentials on page 55	*Client credentials grant flow* on page 31
Get access token using JWT on page 58	*JWT flow* on page 33
Get access token using SAML assertion on page 61	SAML assertion
Get access token information on page 52	*Token information service* on page 39
Revoke token on page 73	*Revoke token* on page 37
Refresh access token on page 68	Refresh token

Related topics

- *OAuth 2.0 message attributes* on page 113

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* on page 17.

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 *Manage access tokens and authorization codes* on page 48.

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 Policy Developer Guide*.

Monitoring settings

The real-time monitoring options enable you to view service usage in API Gateway Manager. For more information on real-time monitoring, see the *API Gateway Administrator Guide*.

Enable monitoring

Select this option to enable real-time monitoring. If this is enabled you can view service usage in the web-based API Gateway Manager tool.

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 user name 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* on page 17.

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 *Manage access tokens and authorization codes* on page 48.

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 *Manage access tokens and authorization codes* on page 48.

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`).

Monitoring settings

The real-time monitoring options enable you to view service usage in API Gateway Manager. For more information on real-time monitoring, see the *API Gateway Administrator Guide*.

Enable monitoring

Select this option to enable real-time monitoring. If this is enabled you can view service usage in the web-based API Gateway Manager tool.

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 user name 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.

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* on page 17.

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 *Manage access tokens and authorization codes* on page 48.

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 a set of strings on the message whiteboard.

Monitoring settings

The real-time monitoring options enable you to view service usage in API Gateway Manager. For more information on real-time monitoring, see the *API Gateway Administrator Guide*.

Enable monitoring

Select this option to enable real-time monitoring. If this is enabled you can view service usage in the web-based API Gateway Manager tool.

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 user name 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.

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* on page 17.

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 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 *Manage access tokens and authorization codes* on page 48.

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 a set of strings on the message whiteboard.

Monitoring settings

The real-time monitoring options enable you to view service usage in API Gateway Manager. For more information on real-time monitoring, see the *API Gateway Administrator Guide*.

Enable monitoring

Select this option to enable real-time monitoring. If this is enabled you can view service usage in the web-based API Gateway Manager tool.

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 user name 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.

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* on page 17.

For more information on SAML, see the IETF draft document:
SAML 2.0 Profile for OAuth 2.0

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.

To guarantee the integrity of an XML signature in a message, the **Access Token using SAML Assertion** filter should use the **XML Signature Verification** filter. For more information, see the **XML Signature Verification** filter in the *API Gateway Policy Developer Guide* . When API Gateway receives the assertion, it converts the assertion into a W3C DOM document and stores this value in a

message attribute named `oauth.saml.doc`. This message attribute is used by the **XML Signature Verification** filter. A sample SAML bearer policy flow is available after you have completed setting up OAuth (see *Set up API Gateway as an OAuth 2.0 server* on page 43).

Access token settings

Configure the following fields on 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 *Manage access tokens and authorization codes* on page 48.

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 a set of strings on the message whiteboard.

Monitoring settings

The real-time monitoring options enable you to view service usage in API Gateway Manager. For more information on real-time monitoring, see the *API Gateway Administrator Guide*.

Enable monitoring

Select this option to enable real-time monitoring. If this is enabled you can view service usage in the web-based API Gateway Manager tool.

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 user name 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.

Consume authorization requests

Overview

The OAuth 2.0 **Authorization Code Flow** filter is used to consume OAuth authorization requests. This filter supports the OAuth 2.0 authorization code (web server) 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 flow is also known as the *three-legged OAuth* flow.

The OAuth 2.0 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.

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 Code Flow** 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* on page 17.

Validation settings

The settings on the **Validation/Templates** tab enable you to specify login and authorization forms to authenticate the resource owner.

Configure the following fields:

Login Form:

Enter the full path to the HTML form that the resource owner can use to log in. Defaults to the value `${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 or deny) client application access to the resources. Defaults to the value `${environment.VDISTDIR}/samples/oauth/templates/requestAccess.html`.

Selector:

Enter a selector for the message attribute that contains the `authentication.subject.id` of the current user if they have already been authenticated. Defaults to the `${authentication.subject.id}` message attribute. For more details on selectors, see the *API Gateway Policy Developer Guide* .

Note Previous versions of API Gateway enabled you to call a policy to authorize the resource owner, and store the subject in a message attribute. This field is used to provide backwards compatibility with configurations using that option. If an authenticated user is not found in the message, the filter automatically uses the internal flow and returns the specified login form.

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 authorization code (for example, in the default **Authz Code Store**). To add an authorization code 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 *Manage access tokens and authorization codes* on page 48.

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` (10 minutes).

Additional parameters to store for this Authorization Code:

To store additional meta data 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 on selectors, see the *API Gateway Policy Developer Guide* .

Note If you entered parameters for the authorization code and parameters for the access token, the data will be merged. For example, if you set `Name:John` and `Department:Engineering` as additional parameters for the authorization code, and set `Department:HR` as an additional parameter for the access token, 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 *Manage access tokens and authorization codes* on page 48.

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 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 a set of strings on the message whiteboard.

Advanced settings

The settings on the **Advanced** tab include monitoring settings and cookie settings.

The real-time monitoring options enable you to view service usage in API Gateway Manager. For more information on real-time monitoring, see the *API Gateway Administrator Guide*.

Enable monitoring

Select this option to enable real-time monitoring. If this is enabled you can view service usage in the web-based API Gateway Manager tool.

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 user name 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.

The traffic monitoring options enable you to view message traffic in API Gateway Manager. For more information on traffic monitoring, see the *API Gateway Administrator Guide*.

Record Outbound Transactions

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

Resource Owner Cookie

Enter the name of the resource owner's cookie in the **Cookie Name** field.

Authorization Session Cookie

Enter the following details for the session cookie:

- Cookie Name – Name of the session cookie
- Domain – Domain value for the `Set-Cookie` header
- Path – Path value for the `Set-Cookie` header
- Expires in – The length of time until the cookie expires
- Secure – Select the check box to add a `secure` flag to the `Set-Cookie` header
- HttpOnly – Select the check box to add a HttpOnly flag to the `Set-Cookie` header

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* on page 17.

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 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 *Manage access tokens and authorization codes* on page 48.

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 real-time monitoring options enable you to view service usage in API Gateway Manager. For more information on real-time monitoring, see the *API Gateway Administrator Guide*.

Enable monitoring

Select this option to enable real-time monitoring. If this is enabled you can view service usage in the web-based API Gateway Manager tool.

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 user name 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.

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* on page 17.

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 Policy Developer 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 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 *Manage access tokens and authorization codes* on page 48.

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`).

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 a set of strings on the message whiteboard.

Monitoring settings

The real-time monitoring options enable you to view service usage in API Gateway Manager. For more information on real-time monitoring, see the *API Gateway Administrator Guide*.

Enable monitoring

Select this option to enable real-time monitoring. If this is enabled you can view service usage in the web-based API Gateway Manager tool.

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 user name 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.

The traffic monitoring options enable you to view message traffic in API Gateway Manager. For more information on traffic monitoring, see the *API Gateway Administrator Guide*.

Record Outbound Transactions

Select whether to record outbound message traffic. You can use this setting to override the **Record Outbound Transactions** setting in **Server Settings > Monitoring > Traffic Monitor**. 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* on page 17.

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, 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 *Manage access tokens and authorization codes* on page 48.

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 real-time monitoring options enable you to view service usage in API Gateway Manager. For more information on real-time monitoring, see the *API Gateway Administrator Guide*.

Enable monitoring

Select this option to enable real-time monitoring. If this is enabled you can view service usage in the web-based API Gateway Manager tool.

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 user name 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.

API Gateway as an OAuth 2.0 resource server 7

This section describes how to configure API Gateway as an OAuth resource server. It describes the following:

- How to use the Client Application Registry web interface to register and manage client applications – See *Register and manage OAuth client applications* on page 75.
- How to manage the OAuth scopes that a client application can access – See *Manage OAuth scopes* on page 76.

Related topics

- *OAuth 2.0 resource server filters* on page 80
- *OAuth 2.0 message attributes* on page 113

Register and manage OAuth client applications

Client applications that send OAuth requests to the API Gateway's authorization server must be registered with the authorization server. This section describes how to register and manage client applications (and the scopes they can access) using the Client Application Registry web-based interface.

Note You must perform the steps described in *Set up API Gateway as an OAuth 2.0 server* on page 43 (for example, enable the OAuth endpoints and import or migrate client applications) before you can manage client applications using the Client Application Registry.

Manage client applications

API Gateway provides the Client Application Registry web-based interface for managing client applications. API Gateway also provides the Client Application Registry REST API to enable you to manage client applications on the command line.

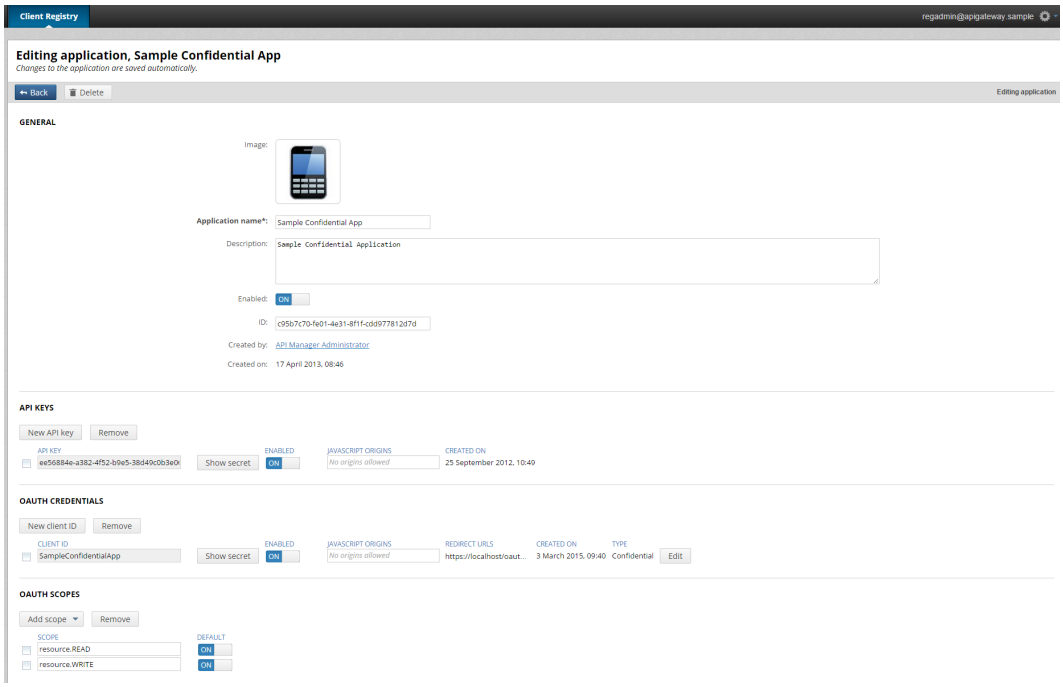
You can access the Client Application Registry web interface at the following URL:

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

Log in using the Client Application Registry user name and password. The default user name is `regadmin`.

To register a new client application, click the **New application** button.

To edit an existing client application, click the application name in the list of applications. You can add API keys, OAuth credentials, and OAuth scopes for the application. For more information on OAuth scopes, see *Manage OAuth scopes* on page 76.

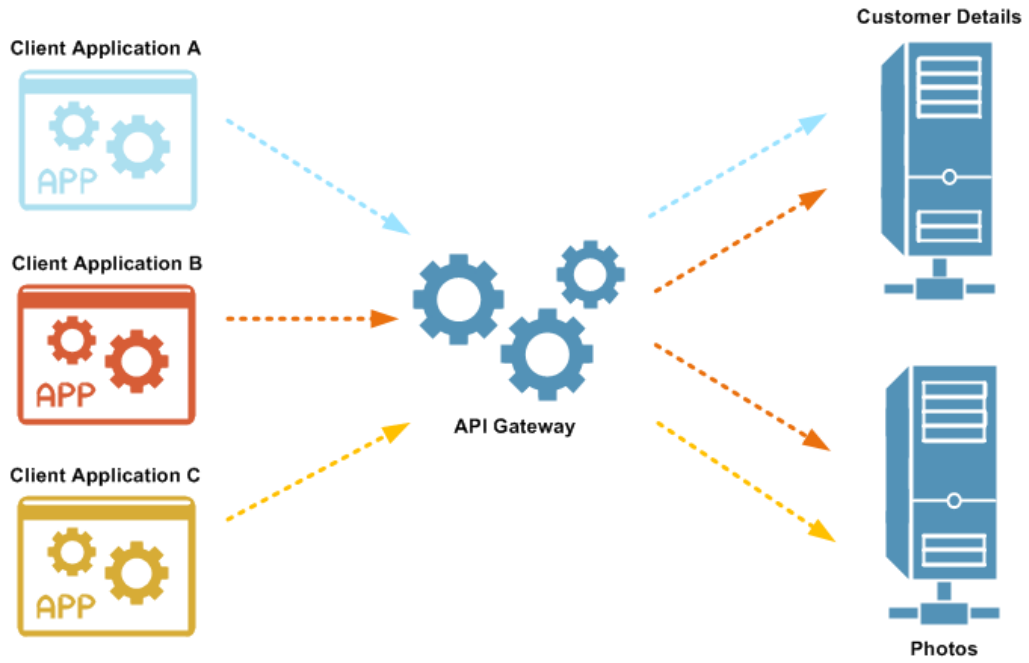


Manage OAuth scopes

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:

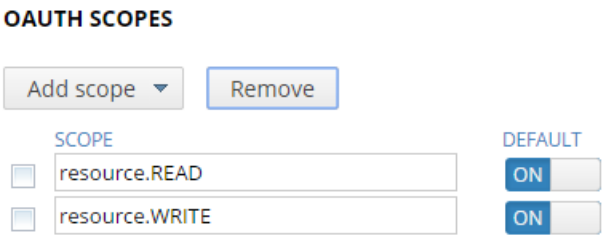
- Client application A can access the `customer_details` scope.
- Client application B can access the `customer_details` and `photos` scopes.
- Client application C can access the `photos` scope only.



You can configure the scopes that a client application can access in the Client Application Registry web interface. When editing the client application, select the **Authentication** tab. In the OAUTH SCOPES section you can specify scopes as free-form text or select a scope from a list of known configured scopes. You can also select a scope as a default scope for the client application. Default scopes are used when an authorization or token request does not contain scopes. The full list of scopes (default and non-default) represents the list of scopes that can be included in an authorization or token request.

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.

The following figure shows the default scopes for a client application:



Note 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 the **Validate Access Token** filter in the server configuration. For more details on this filter, see *Validate access token* on page 80.

Note 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 configured 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 Policy Developer Guide*.

Client Application Registry

By default, OAuth client application data is stored in a Key Property Store (KPS) backed by an Apache Cassandra database. For more details on KPS and Apache Cassandra, see the *API Gateway Key Property Store User Guide*.

Relational database-backed Client Application Registry

The Client Application Registry KPS can also be backed by a relational database such as Oracle, MySQL, IBM DB2, or Microsoft SQL Server. For more information on KPS and database storage, see the *API Gateway Key Property Store User Guide*.

OAuth relational database schemas

You can view the OAuth relational database schemas by using SQL commands. For example, to view a table in a MySQL database, use the following command:

```
show columns from TABLE_NAME;
```

The OAuth table names are:

- oauth_access_token
- oauth_refresh_token
- oauth_authz_code

Data security

If you have set an encryption passphrase for API Gateway, the OAuth secret and API secret are encrypted in the Client Application Registry.

If you change the encryption passphrase at any point, you must re-encrypt the data in the Client Application Registry or you will not be able to connect to the Client Application Registry web-based interface.

To re-encrypt the data, use the `kpsadmin` tool, and select the option to `Re-encrypt All`. This re-encrypts all data in all tables in a collection. You are prompted for the old passphrase (needed to decrypt the data). The data is then re-encrypted with the current API Gateway passphrase. Repeat this process for each KPS collection.

For more information on the `kpsadmin` tool, see the *API Gateway Key Property Store User Guide*.

OAuth 2.0 resource server filters 8

This section describes the filters you can use when API Gateway is acting as an OAuth resource server. These include:

Filter	Description
Validate access token on page 80	Validate an access token and allow access to a protected resource.

Related topics

- *OAuth 2.0 message attributes* on page 113

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* on page 17.

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 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 *Manage access tokens and authorization codes* on page 48.

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.

API Gateway as an OAuth 2.0 client 9

This section describes how to configure API Gateway as an OAuth client. It describes the following:

- OAuth client features of API Gateway and a sample OAuth client workflow – See *Introduction to API Gateway OAuth client* on page 83.
- How to set up API Gateway as an OAuth client – See *Set up API Gateway OAuth client* on page 86.
- How to configure OAuth client application credentials in API Gateway – See *Configure OAuth client application credentials* on page 86.

Related topics

- *OAuth 2.0 client filters* on page 93
- *OAuth 2.0 message attributes* on page 113

Introduction to API Gateway OAuth client

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.

API Gateway can act as the client application in an OAuth 2.0 scenario, and as such 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. This provides the following benefits:

- The OAuth client burden is moved to API Gateway
- The resource owner's credentials are never shared with the client application
- The access token is never shared with the resource owner's user agent

Note This document assumes that you are familiar with both the terms and concepts described in the OAuth 2.0 Authorization Framework and the OAuth server features of API Gateway (for more information, see *Introduction to API Gateway OAuth 2.0 server* on page 13).

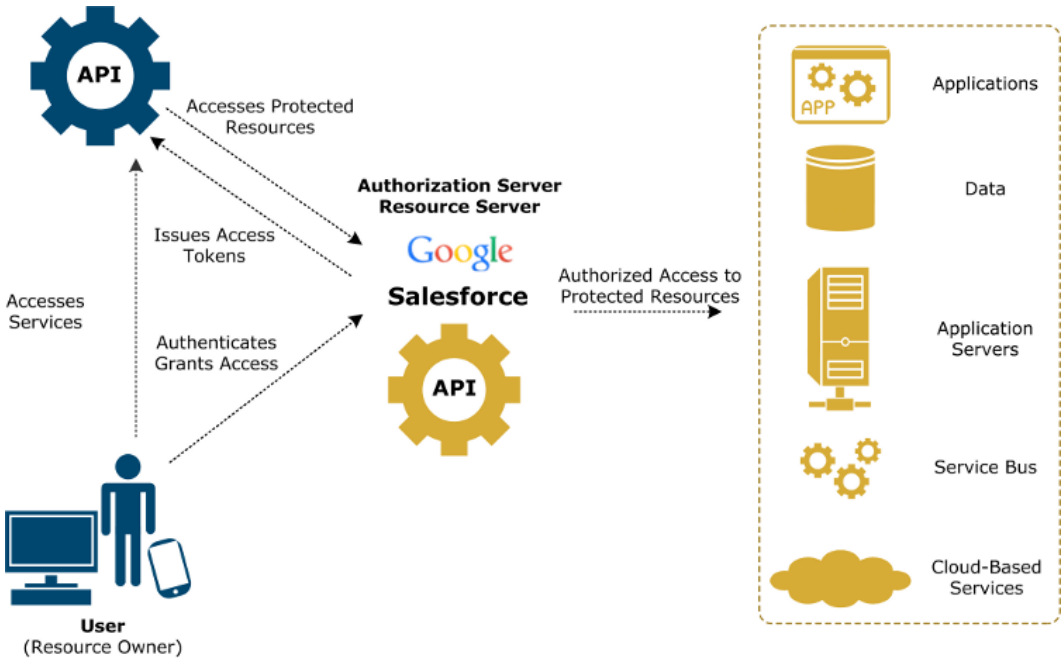
API Gateway OAuth client features

API Gateway provides the following features to support OAuth 2.0 client functionality:

- Provider profiles for defining OAuth service providers and the applications registered with them.
- A set of preconfigured sample provider profiles for use with Oracle, Google, and Salesforce OAuth services.
- Storage of received tokens.
- Support for the following OAuth flows:
  - Authorization code grant
  - Resource owner password credentials
  - Client credentials grant
  - JWT
  - SAML assertion

    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 it is not applicable for API Gateway acting as an OAuth client.

The following diagram shows the role of API Gateway as an OAuth 2.0 client application accessing OAuth services provided by Oracle API Gateway, Google, and Salesforce:

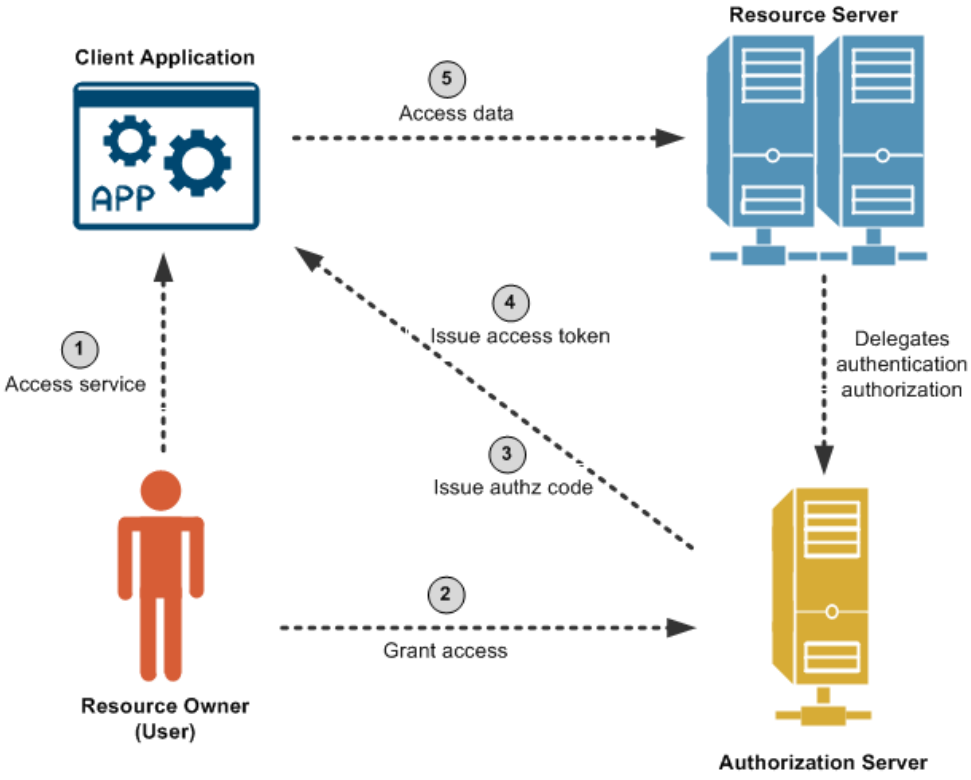


For more information on how to integrate your application with Google or Salesforce APIs using API Gateway and OAuth 2.0, see the *API Gateway Google OAuth 2.0 Integration Guide* and the *API Gateway Salesforce OAuth 2.0 Integration Guide*.

OAuth 2.0 example client workflow

This example is similar to the *OAuth server example workflow* on page 14, but in this context API Gateway acts as a client, and the service provider is Google.

Assume that you, as a resource owner, are using a service that wants to access your Google calendar (a protected resource). The service is defined on API Gateway (API Gateway is an OAuth client). You do not want to reveal your Google credentials to API Gateway. This problem can be solved using the example OAuth 2.0 web server flow shown in the following diagram:



Out of band, API Gateway preregisters with Google and obtains a client ID and secret. API Gateway also registers a callback URL to receive the authorization code from Google when you, as resource owner, authorize access to your Google calendar. The 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 OAuth provider profile using Policy Studio (for more information, see *Configure OAuth client application credentials* on page 86). 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 an HTTP listener on API Gateway, with a filter for receiving the authorization code.

The steps in the diagram are described as follows:

1. Using a browser or mobile phone, you access a service defined on API Gateway, which needs to access your Google calendar on your behalf. The client application initiates the authorization

flow by redirecting your browser to the authorization endpoint defined in the Google OAuth provider profile.

2. After following the redirect you log in to your Google account and authorize the application for the requested scope.

   Note You have not shared your Google user name and password with the API Gateway application. At this point, you, as the resource owner, are no longer involved in the process.

3. The authorization server then redirects your browser to the callback URL on API Gateway, along with an authorization code.
4. The API Gateway client application gets the authorization code, and must exchange this short-lived code for an access token. The client application 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.
5. The client application sends the access token to the API (resource server), and receives the calendar information as requested.

Set up API Gateway OAuth client

API Gateway includes a number of sample client applications. You can deploy the sample applications using the `deployOAuthConfig.py` script or you can import them manually.

- For more information on using the `deployOAuthConfig.py` script, see *Enable OAuth management* on page 43.
- For more information on importing them manually, see *Import sample client applications* on page 45.

Configure OAuth client application credentials

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 the following 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
- API Gateway applications can be registered in the Client Application Registry (port 8089).

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 as an OAuth 2.0 server* on page 43. The API Gateway provider uses the existing OAuth server samples for authorization and token endpoints (for example, `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 access the preconfigured OAuth providers and add client application credentials under the **External Connections** node in the Policy Studio tree. You can also add new OAuth providers. See *Add OAuth provider* on page 91 for more information.

Add application credentials

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 *API Gateway OAuth 2.0 authentication flows* on page 17.

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 *Create a callback URL listener* on page 91).

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 *Manage OAuth scopes* on page 76.

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, 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}`).

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

The API Gateway uses a SAML template to generate the SAML assertion. The template file is stored under the `Resources/Stylesheets` directory in Policy Studio when the `deployOAuthConfig` script is executed (see *Enable OAuth management* on page 43). Alternatively, you can find this file in `$VDISTDIR/samples/oauth/templates/samltemplate.xml` and import it via Policy Studio. At runtime the values in the template are substituted with values configured for the OAuth client SAML application.

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.

Note 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	Default 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 or extended 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 document) to identify the resource owner for whom a token is being requested. Service defined claims must also be added here. Unrecognized claims should be ignored by service providers.

Claim	Default value
`sub`	The subject ID of the resource owner. This identifies the resource owner for whom the request is being made. The property **prn** can also be used here for some providers (for example, SalesForce), but the use of this property has been superseded by **sub** in the IETF specification.
`scope`	A space delimited list of scopes within the claim, defining the required permissions of the request.

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 Policy Developer Guide*.

Add OAuth 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 *Manage client access tokens* on page 91.

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.

Create 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 an HTTP listener. The policy for this path needs only to add a **Redirect resource owner to Authz Server** filter (see *Redirect resource owner to authorization server* on page 95). The filter must be configured with a reference to the relevant provider profile for this callback URL.

Manage client access tokens

You can configure client access token stores under the **Libraries > OAuth2 Stores** node in the Policy Studio tree view. API Gateway can store client access tokens in its cache, in an embedded database, or in a relational database. For more information on the persistent storage options, see *Manage access tokens and authorization codes* on page 48.

To store client access tokens in a relational database, use the SQL script in `$VDISTIR\system\conf\sql\DBMS_TYPE\oauth-client.sql` to create the supporting schema. The OAuth client table names are:

- oauth_client_access_token
- oauth_client_token_props
- oauth_client_refresh_token

OAuth client access tokens are purged on expiry. After a successful token request the OAuth client access token (and potentially a refresh token) are stored in persistent storage. OAuth authorization servers usually return an expiry with the access token, otherwise the default expiry of 3600 seconds is used. An expiry is not usually returned with a refresh token, but a default expiry of 30 days is used. You can alter these expiry settings in the **Client Token Cleanup Settings** section of the client access token store.

Note For client access token stores backed by a database, you can configure the **Purge expired tokens** field to perform a purge (for example, run a query to remove tokens) at a specified interval.

OAuth 2.0 client filters 10

This section describes the filters you can use when API Gateway is acting as an OAuth client. These include:

Filter	Description
Delete an OAuth client access token on page 93	Delete a client access token.
Get OAuth client access token on page 94	Request a client access token.
Redirect resource owner to authorization server on page 95	Redirect a resource owner to an authorization server to grant or deny access to a resource.
Refresh an OAuth client access token on page 96	Refresh a client access token.
Retrieve OAuth client access token from token storage on page 97	Retrieve a previously stored client access token.
Save an OAuth client access token on page 98	Save a client access token.

Related topics

- *OAuth 2.0 message attributes* on page 113

Delete an OAuth client access token

Overview

You can use the **Delete an OAuth Client Token** filter to explicitly delete an OAuth client access token, or refresh token, or both.

This filter requires the message attribute `oauth.client.application` to determine if the application has an access token, or refresh token, or both. The filter returns false if the required message attribute is not set or if there is a problem removing an access or refresh token from token storage.

General settings

Configure the following general setting for the **Delete an OAuth Client Token** filter:

Name:
Enter a suitable name for this filter.

Get OAuth client access token

Overview

You can use the **Get OAuth Access Token** filter to request a token. This filter attempts to get the access token from persistent storage, and if a token is not available it sends an outbound token request.

General settings

Configure the following general settings for the **Get OAuth Access Token** filter:

Name:
Enter a suitable name for this filter.

Choose OAuth Token Key:
Enter the message attribute to be used as the key to look up the token. Defaults to `${authentication.subject.id}`.

If no key is specified in this field, the access token is not saved. This is to accommodate anonymous use of OAuth whereby the user starting the process does not need to authenticate with API Gateway first.

Optionally select a client credential profile:
Select this option and click the browse button to select an OAuth 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 Policy Developer 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 Policy Developer Guide*.

Redirect resource owner to authorization server

Overview

The purpose of the **Redirect resource owner to Authz Server** filter is to redirect the resource owner's user agent to the OAuth authorization server. This filter can only be used in the authorization code flow.

General settings

Configure the following general settings for the **Redirect resource owner to Authz Server** filter:

Name:
Enter a suitable name for this filter.

Choose OAuth Token Key:
Enter the message attribute to be used as the key to look up the token. The token key must be set to the authentication value you require for the OAuth token. In the context of an OpenID Connect flow the OAuth token key can be a cookie value. Defaults to `${authentication.subject.id}`.

If no key is specified, this field is ignored. This is to accommodate anonymous use of OAuth whereby the user starting the process does not need to authenticate with API Gateway first.

Override scopes setup for authz request:
This field allows you to override the scopes assigned to a client profile. This could potentially be

used in the OpenID Connect flow whereby you could specify scopes for the OpenID Connect flow, or if you already have a session you would not need to specify OpenID Connect scopes. (For an example, see the OAuth client demo.)

Optionally select a client credential profile:
Select this option and click the browse button to select an OAuth 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.

Configure OAuth State Map:
This allows you to configure extra parameters in the OAuth state map (for example, to implement cross-site request forgery (CSRF) protection). By default the state map contains `client_id` and `oauth.token.id`. Any extra parameters added here are added to the state map. The message attribute `oauth.state.map` will be available at the callback endpoint when an authorization code is exchanged for a token.

Select one of the following options:

- **Add extra state properties**:

  Click **Add** to store additional state properties, and enter the **Name** and **Value** in the dialog.
- **Retrieve properties from selector**:

  Enter the attribute that stores the properties. Defaults to `oauth.state.map`.

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. This filter looks up the token and checks for a refresh token. If it finds a refresh token, the filter sends an outbound refresh token request to the OAuth authorization server to obtain a new access token (and possibly a new refresh 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.

Choose OAuth Token Key:
Enter the message attribute to be used as the key to lookup the token. Defaults to `${authentication.subject.id}`.

Optionally select a client credential profile:
Select this option and click the browse button to select an OAuth 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 Policy Developer 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 Policy Developer Guide*.

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, or the embedded Apache Cassandra database. (For more details on client access token stores, see *Manage client access tokens* on page 91.)

A configured token store is associated with an OAuth provider (see *Add OAuth provider* on page 91) and is shared by all client applications registered with that provider.

These stored tokens can be retrieved by this filter by specifying the OAuth 2.0 provider profile (the client application registered with a provider) and the token key (for example, the authentication subject id of the current user). Stored tokens are indexed by the client ID of the the client application and the token key. 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 OAuth Token Key:
Enter the message attribute to be used as the key to lookup the token. Defaults to `${authentication.subject.id}`.

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

Save an OAuth client access token

Overview

You can use the **Save an OAuth Client Token** filter to save a token with a different token key. This filter can fail if:

- The token has no token key
- The token has no application
- The token key is the same key that was already stored for the application
- There is a problem saving the token to persistent storage

General settings

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

Name:
Enter a suitable name for this filter.

Choose OAuth Token Key:
Enter the message attribute to be used as the token key. Defaults to `${authentication.subject.id}`.

API Gateway and OpenID Connect 11

This section describes how to configure API Gateway as an OpenID Connect identity provider (IdP) and as an OpenID Connect relying party (RP). It describes the following:

- OpenID Connect features of API Gateway – See *Introduction to API Gateway OpenID Connect* on page 100.
- Sample OpenID Connect workflow – See *OpenID Connect flow* on page 104.
- How to set up API Gateway as an OpenID Connect server – See *Build an OpenID Connect IdP server* on page 105.
- How to set up API Gateway as an OpenID Connect client – See *Build an OpenID Connect client* on page 105.
- How to use the client demo for an OpenID Connect flow – See *Use the API Gateway OAuth client demo* on page 106.

Related topics

- *OpenID Connect filters* on page 110
- *OAuth 2.0 message attributes* on page 113

Introduction to API Gateway OpenID Connect

The OpenID Connect 1.0 (OID) protocol is a simple identity layer on top of the OAuth 2.0 protocol. OAuth 2.0 provides an access authorization delegation protocol, and Open ID Connect leverages OAuth features to allow authorized access to user authentication session APIs in an interoperable manner. OpenID Connect uses a REST interface and simple JSON assertions called JSON Web Tokens (JWTs) to provide identifying *claims* about users. The protocol is designed to be API-friendly and adaptable to multiple formats such as web and mobile, and it has built-in provisions for robust signing and encryption.

In its simplest form an OpenID Connect deployment allows applications (such as browsers, mobiles, and desktop clients), to request and receive information about the identities of a user. This allows the user to authenticate to a third-party application that acts as a relying party (RP) using an identity established with the Open ID Connect identity provider (IdP).

This section describes the concepts behind OpenID Connect and demonstrates how to use the API Gateway as an OpenID Connect identity provider and as a relying party. The following sections use the client demo that ships with API Gateway to illustrate OpenID Connect concepts (see *Use the API Gateway OAuth client demo* on page 106).

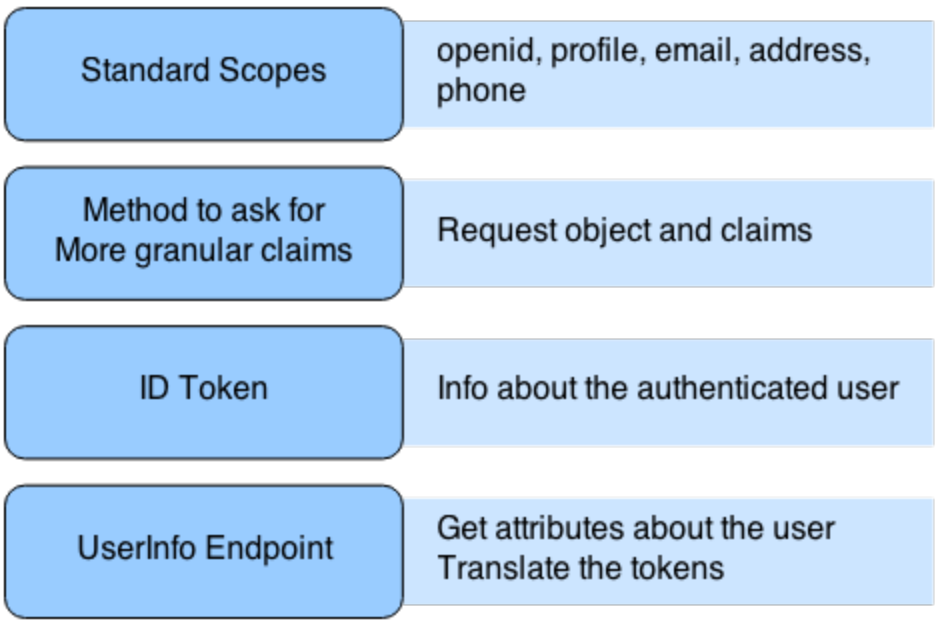
OpenID Connect concepts

OpenID Connect is specified in the OpenID Connect 1.0 specification. It defines the following concepts:

- Claim: A piece of information about an authenticated user (for example, email or phone number).
- Relying party (RP): OAuth 2.0 client application requiring end-user authentication and claims from an OpenID Connect identity provider. API Gateway can act as a relying party consuming services from a third party such as Google.
- Identity provider (IdP): OAuth 2.0 authorization server that is capable of authenticating the end-user and providing claims to a relying party about the end user. API Gateway can act as an identity provider for relying parties.
- ID token: JSON Web Token (JWT) that contains claims about the authenticated user.
- UserInfo endpoint: Protected resource that, when presented with an access token by the client, returns authorized information about the end user.

Relationship to OAuth 2.0

To support maximum interoperability the OID specification defines standard scopes, defined request objects and corresponding claims, the ID token format, and a UserInfo endpoint. These features represent the primary additions to the OAuth 2.0 standard and should be available across all IdP implementations of OpenID Connect.



Standard scopes

The OpenID Connect specification defines a set of predefined scopes for use in the OpenID Connect authorization flow.

- `openid` – REQUIRED. Informs the authorization server that the client is making an OpenID Connect request. This is the only scope directly supported in the OpenID Connect filters. If the `openid` scope is not present in an authorization request the OpenID Connect **Create ID Token** filter passes, leaving the message unmodified. When it is present, and the request is valid, the ID token associated with the authentication session is returned. If the `response_type` includes `token`, the ID token is returned in the authorization response along with the access token. If the `response_type` includes `code`, the ID token is returned as part of the token endpoint response.

The following scopes are not directly catered for in the OpenID Connect filters, but are made available on the message so that policy developers can correctly respond to them in policies. They differ to OAuth 2.0 scopes in that they do not need to be expressly associated with a resource or API, and are automatically considered valid scopes for client applications. However, these scopes can be considered regular OAuth 2.0 scopes when accessing the UserInfo endpoint, and if they are present in the access token, policy developers can construct an appropriate claims response.

- `profile` – OPTIONAL. This scope requests that the issued access token grants access to the end user's default profile claims (excluding the `address` and `email` claims) at the UserInfo endpoint. The profile claims can include: `name`, `family_name`, `given_name`, `middle_name`, `nickname`, `preferred_username`, `profile`, `picture`, `website`, `gender`, `birthdate`, `zoneinfo`, `locale`, and `updated_at`.
- `email` – OPTIONAL. This scope requests that the issued access token grants access to the `email` and `email_verified` claims at the UserInfo endpoint.

- `address` – OPTIONAL. This scope requests that the issued access token grants access to the `address` claim at the UserInfo endpoint.
- `phone` – OPTIONAL. This scope requests that the issued access token grants access to the `phone_number` and `phone_number_verified` claims at the UserInfo endpoint.

Request object and claims

The OpenID Connect request object is an optional part of the specification and is not supported in the current version of API Gateway. The request object is used to provide OpenID Connect request parameters that might differ from the default ones. Request objects might be supported in future versions of API Gateway.

ID token

The ID token is a JWT-based security token that contains claims about the authentication of an end user by an authorization server. When using the authorization code flow the ID token is returned as a property of the access token. For the implicit and some hybrid flows the ID token is returned in response to the authorization request. As an IdP API Gateway will produce an ID token using the **Create ID Token** filter, either in the authorization endpoint policy or the token endpoint policy. This token will be signed for verification by the client and can include user defined claims. This provides flexibility in creating claims from any user store. Acting as an RP, a client can verify a received ID token with the **Verify ID Token** filter. This should be done in the callback policy as API Gateway does not support implicit flows as a client. After being verified, the ID token can be used to look up or create a user record and create an authenticated session for the user.

UserInfo endpoint

The UserInfo endpoint is defined as an OAuth 2.0 protected resource that returns extended claims about the authenticated end user. To access this resource an API Gatewayacting as an RP must use the access token received in the OpenID Connect authentication process. A successful request will return a JSON object with the claims for the user. As an IdP the implementation of the UserInfo endpoint should be similar to any OAuth protected resource with a minimum scope requirement of `openid`. The implementation of this endpoint is deliberately left open for policy developers to integrate their own authentication stores.

Prerequisites

To use the OpenID Connect features of API Gateway, the following are required:

- A knowledge of the OAuth 2.0 specification
- A knowledge of API Gateway and Policy Studio
- A local installation of API Gateway and Policy Studio
- A deployed client demo (see *Use the API Gateway OAuth client demo* on page 106)

OpenID Connect flow

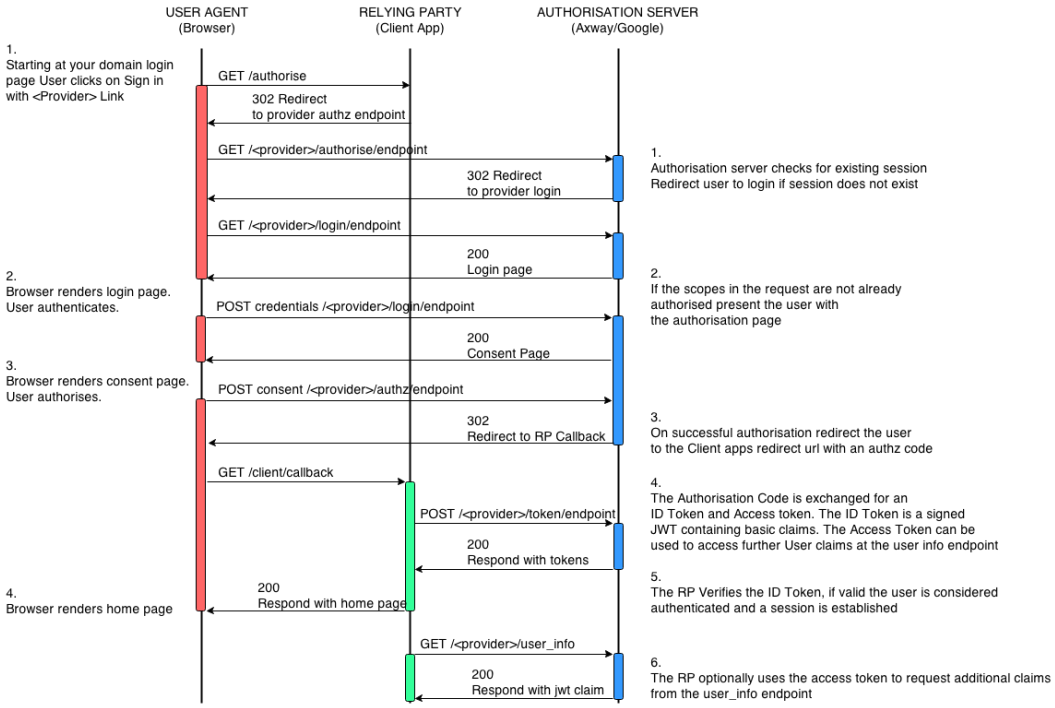
The OpenID Connect process follows the OAuth 2.0 three-legged authorization code flow, but with the additional concepts of an ID token and a UserInfo endpoint.

The authorization code flow consists of the following steps:

- Relying party prepares an authentication request containing the desired request parameters.
- Relying party sends the request to the authorization server by redirecting the end user via their user agent (browser).
- Authorization server authenticates the end user.
- Authorization server obtains end user consent and authorization.
- Authorization server sends the end user back to the relying party with an authorization code.
- Relying party requests a response using the authorization code at the token endpoint.
- Client receives a response that contains an ID token and access token in the response body.
- Client validates the ID token and retrieves the end user's subject identifier.

On successful completion of these steps the user can be considered authenticated in the relying party's application. At this point the client application can make a request for further claims and, if required, create a local record of this user for future use, or retrieve a previously created user record.

The following diagram illustrates these steps.



Build an OpenID Connect IdP server

To build an IdP server on top of an existing OAuth deployment, follow these steps:

1. Add a **Create ID Token** filter to the token endpoint policy after the **Access Token using Authorization Code** filter. If API Gateway will also support implicit or hybrid grant types then add a **Create ID Token** filter to the authorization endpoint policy after the **Authorization Code Flow** filter.
2. Create a UserInfo endpoint. This is similar to any OAuth protected resource using a **Validate Access Token** filter with a minimum scope requirement of `openid`. After a successful validation the UserInfo policy must create a JSON object response representing claims about the user associated with the access token. The user can be identified by the **Validate Access Token** filter with the `authentication.subject.id` message property.

   The following is a non-normative example of the JSON response:

```
{
 "kind": "APIGatewayOpenIdConnect",
 "gender": "femail",
 "sub": "sampleuser",
 "name": "Sample User",
 "given_name": "Sample",
 "family_name": "${User}",
 "picture": "https://URL.TO.IMAGE/",
 "email": "sampleuser@gatweway",
 "email_verified": "true",
 "locale": "en"
}
```

Build an OpenID Connect client

Building a simple OpenID Connect client involves adding only one additional step to the OAuth authorization code flow for an OAuth client (see *OAuth 2.0 example client workflow* on page 85):

1. Verify the ID token in the callback policy of the client application. To verify the ID token, use the **Verify ID Token** filter (see *Verify an OpenID Connect ID token* on page 111) after the **Get OAuth Access Token** filter (see *Get OAuth client access token* on page 94). If the **Verify ID Token** filter relies on a JSON Web Key (JWK) set for verification, you must download (and optionally cache) the IdP key set. The client demo includes an example of using the Google JWK set (see *Use the API Gateway OAuth client demo* on page 106).

If the ID token is successfully verified, the filter passes and the following message properties are created:

- claim (`openid.idtoken.claim`): The claim is a JSON object representing basic claims about the user.
- subject (`openid.idtoken.sub`): The subject is the IdP's unique identifier for the user.

The policy designer can decide how best to use these properties. For example, the subject could be used to look up a local user store to see if there is an existing relationship with the user. If there is no existing user, the claim could be used to generate a new user record for future use. At this point, the user can be considered authenticated and a new session can be created. If an access token was returned it can be used to retrieve additional claims from the IdP's UserInfo endpoint.

Note When API Gateway is acting as a client application, the implicit grant type is not supported.

Use the API Gateway OAuth client demo

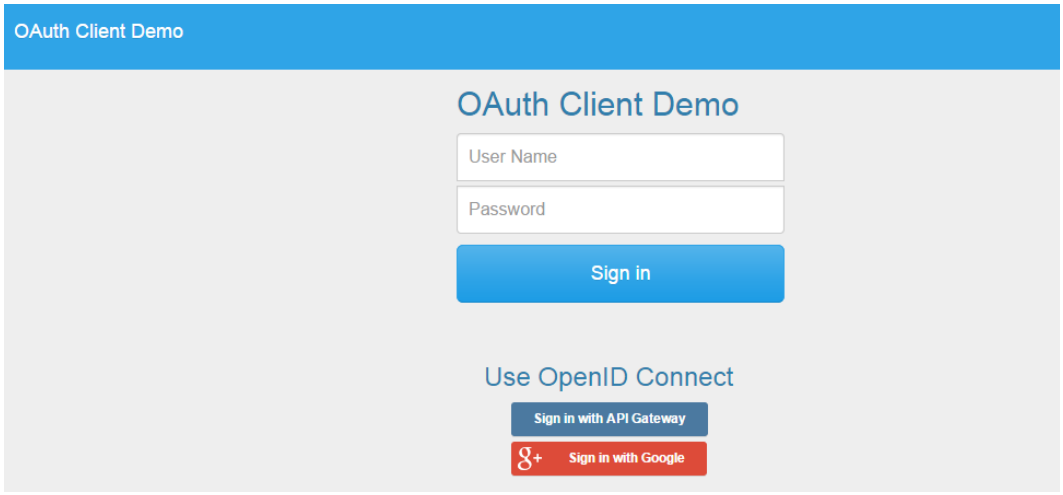
API Gateway ships with a client demo that shows a typical use case for OAuth 2.0 and OpenID Connect.

There are a number of actors involved in this demo:

- API Gateway acts as both a OpenID Connect relying party (RP) and an OpenID Connect identity provider (IdP). It also acts as a basic OAuth client application, an OAuth authorization server (AS) and an OAuth resource server (RS).
- Google is configured as an OAuth AS and RS and as an OpenID Connect IdP
- Salesforce is configured as an OAuth AS and RS but not as an OpenID Connect IdP.

To complete the configuration for Google and Salesforce you must register a client application with each provider, and update the relevant provider profiles with the received client ID and secrets. For more information, see *Add application credentials* on page 87.

When you connect to the client demo (for example, at `https://localhost:8088`), a login page is displayed:



This page presents three login options:

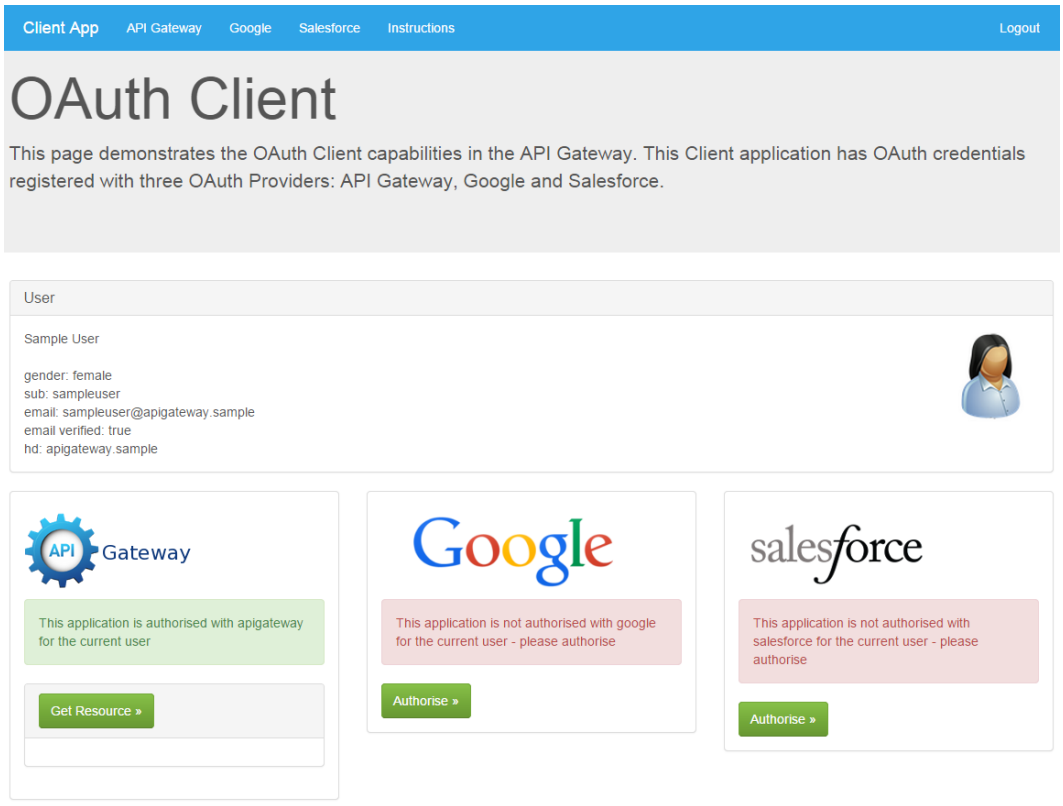
1. Enter a user name and password and click **Sign In**. When you sign in with this option, API Gateway acts as a direct authentication server. This is regular form-based authentication backed by the relevant filter. The user credentials are checked against the local user store and a session is created for a valid user. After being authenticated the user can instantiate an OAuth 2.0

authorization flow to get an access token for one of the configured OAuth authentication servers.

2. **Sign in with API Gateway**. When you sign in with this option API Gateway acts as both the RP and the corresponding IdP. The IdP role is conceptually a separate API Gateway instance but for the purposes of the demo a single instance fulfills both roles. This option causes the API Gateway as RP to issue an authorization redirect through the user agent to the IdP. The request includes scopes for an ID token and a regular access token. After being authenticated the access token can be used to access a protected resource as well as the UserInfo endpoint.
3. **Sign in with Google**. When you sign in with this option the API Gateway acts as the RP with Google acting as the IdP. You must update the configuration in Policy Studio to add a valid set of Google OAuth credentials (see *Add application credentials* on page 87). This option redirects the user to Google for authentication and authorization. The authorization request asks for OpenID and access to the users calendar.

   Note The credentials you create with Google must have access to the Calendar and Google+ APIs.

After successful authentication, you are presented with the following page:



When using one of OpenID Connect options the user information presented on this page is acquired by accessing the UserInfo endpoint of the relevant IdP. The **Get Resource** button uses the received access token to access a protected resource (for example, the Google resource accessed is the user's Google calendar).

Deploy the client demo

API Gateway ships with a preconfigured client demo that demonstrates the use of API Gateway and Google as Open ID providers, and API Gateway as a client. You must deploy the demo manually, as described in *Enable OAuth management* on page 43.

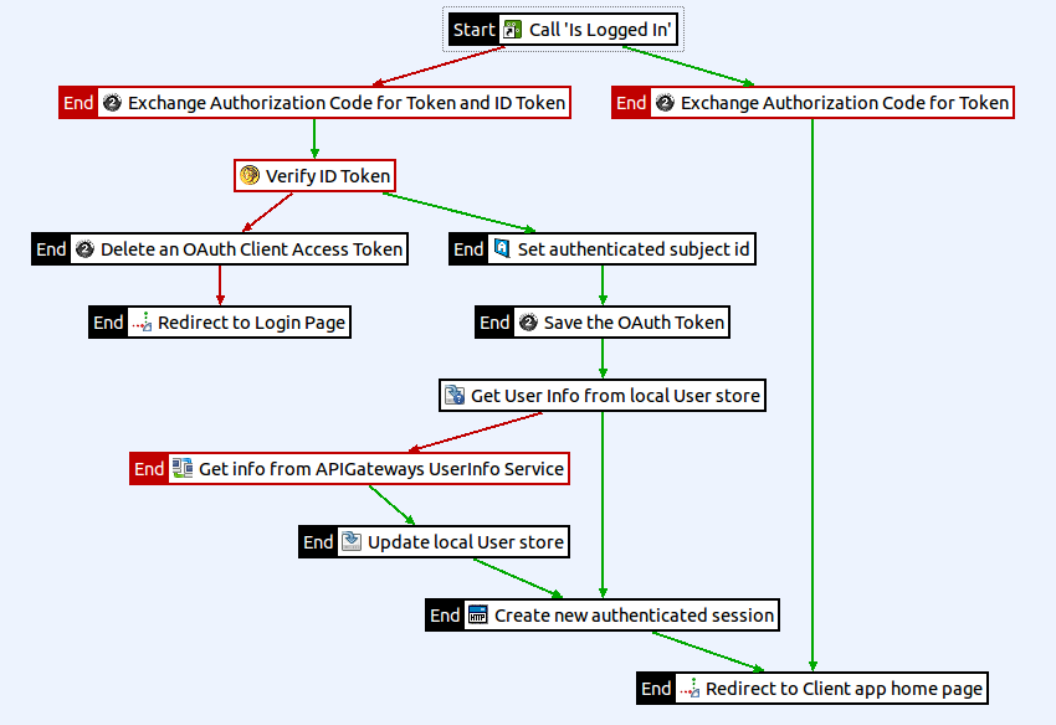
If you have already deployed OAuth using the `deployOAuthConfig.py` script with the default settings, the demo is already available. If you have deployed only the OAuth server configuration using the script, run the script again with the `--type=clientdemo` option to deploy the client demo.

Client policies

The majority of the work in this demo is carried out in the client policies. To support the different methods of authentication (form-based and OpenID Connect), the demo is configured to issue an anonymous session to start the process. This anonymous session is replaced with an updated user session after the user has been identified with either a successful form login or an ID token.

Callback sample

In the demo configuration the callback policy first checks if the current session is for an authenticated user. If it is an anonymous session the policy exchanges the code for an access token and attempts to verify an ID token if one is received. If the ID token is valid it sets the `authentication.subject.id` to the `sub` identifier and saves the token. Using the `sub` the policy then checks the local user store for additional user information (in this case the local user store is a cache set up to simulate an actual user store). If an entry cannot be found for the user, a request is made to the provider's UserInfo endpoint using the access token. A successful request updates the local user store with the returned user data. Finally, a new authenticated session is created for the user and they are returned to the client application home page where they are now signed in (in a real world example a user might be presented with a form to alter or embellish the retrieved data before it is persisted).


Start
Call 'Is Logged In'
End
Exchange Authorization Code for Token and ID Token
End
Exchange Authorization Code for Token
Verify ID Token
End
Delete an OAuth Client Access Token
End
Set authenticated subject id
End
Redirect to Login Page
End
Save the OAuth Token
Get User Info from local User store
End
Get info from APIGateways UserInfo Service
End
Update local User store
End
Create new authenticated session
End
Redirect to Client app home page

OpenID Connect filters 12

This section describes the filters you can use for OpenID Connect flows. These include:

Filter	Description
Create an OpenID Connect ID token on page 110	Create an ID token.
Verify an OpenID Connect ID token on page 111	Verify an ID token.

Related topics

- *OAuth 2.0 message attributes* on page 113

Create an OpenID Connect ID token

Overview

You can use the **Create ID Token** filter to create an OpenID Connect ID token when API Gateway is acting as an OpenID Connect server (also known as OpenID provider or OP). It is the responsibility of the OAuth authorization server to generate an ID token. The ID token is a security token that contains claims about the authentication of an end user by an authorization server when using a client. The ID token is represented as a JSON Web Token (JWT).

To generate an ID token the following claims are required:

- `iat`
- `iss`
- `exp`
- `sub`
- `aud`

For more information on the required claims within an ID token, see the OpenID Connect specification.This filter enables you to specify the subject (`sub`), the issuer (`iss`), and the expiration time (`exp`) of the ID token. Other claims (for example, `iat`, `exp`, and `aud`) are handled internally. The JWT expiry in seconds is appended on to the current time to give the ID token an expiry.

This filter also enables you to specify how to sign the ID token, and to add additional claims to the ID token.

General settings

Configure the following general settings for the **Create ID Token** filter:

Name:
Enter a suitable name for this filter.

Subject (sub):
Subject identifier. A locally unique and never reassigned identifier within the issuer for the end-user, which is intended to be consumed by the client. The default value is `${authentication.subject.id}`.

Issuer (iss):
Issuer identifier for the issuer of the response. The default value is `${http.request.url}`.

Expiration time in secs (exp):
Enter the expiry time for the ID token, in seconds. The default value is 60 seconds.

Apply a signature algorithm:
Select one of the following options:

- **Sign ID token with private key**:
  Select this option and click **Signing Key** to select a private key certificate that has been registered with the OpenID provider, and use it to sign the ID token.
- **Sign using client secret**:
  Select this option to sign the ID token using a client secret issued by the OpenID provider.

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

Verify an OpenID Connect ID token

Overview

You can use the **Verify ID Token** filter to verify an OpenID Connect ID token using a JSON Web Key (JWK) set, certificate, or client secret. The ID token is a security token that contains claims about the authentication of an end user by an authorization server when using a client. The ID token is represented as a JSON Web Token (JWT).

This filter requires the message attributes `openid.idtoken` and `oauth.client.application` to be on the message whiteboard. The filter parses the ID token into a JWT header and claim and validates the signature using either a JWK set, a certificate, or a client secret. On success the filter returns true and sets the ID token (`openid.idtoken`) `claim` and `sub` values on the message whiteboard.

For more information on OpenID Connect, see the OpenID Connect specification.

General settings

Configure the following general settings for the **Verify ID Token** filter:

Name:
Enter a suitable name for this filter.

Clock Skew (seconds):
Enter a number of seconds to allow for clock skew. This allows for clock skew when verifying the token's issued at (`iat`), expiration time (`exp`), and not before values. The default value is 60 seconds.

Issuer (iss):
Issuer identifier for the issuer of the response. The default value is `${oauth.client.application.getTokenURL().toString()}`.

Verify ID Token:
Select one of the following options:

- **With JSON Web Key Set**:
  Select this option to verify the ID token using a JWK set.
- **With Certificate**:
  Select this option and click **Signing Key** to select a private key certificate, and use it to verify the ID token.
- **With Client Secret**:
  Select this option to verify the ID token using a client secret.

Appendix A: OAuth 2.0 message attributes

This section describes the message attributes that are available in the API Gateway OAuth server and OAuth client filters.

OAuth 2.0 server message attributes

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 Policy Developer 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:

```
${oauth.client.details.getClientType()}
${oauth.client.details.getApplication()}
${oauth.client.details.getBase64EncodedCert()}
${oauth.client.details.getX509Cert()}
${oauth.client.details.getName()}
${oauth.client.details.getDescription()}
${oauth.client.details.getLogo()}
${oauth.client.details.getApplicationID()}
${oauth.client.details.getContactPhone()}
${oauth.client.details.getContactEmail()}
${oauth.client.details.getClientID()}
${oauth.client.details.getClientSecret()}
${oauth.client.details.getRedirectURLs()}
${oauth.client.details.getScopes()}
${oauth.client.details.getDefaultScopes()}
${oauth.client.details.isEnabled()}
```

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

```
confidential
com.vordel.common.apiserver.model.Application@126c334d
-----BEGIN CERTIFICATE-----MIICwzCCAasCBgE6HBsdpzANBg ......END CERTIFICATE
CN=Change this for production
```

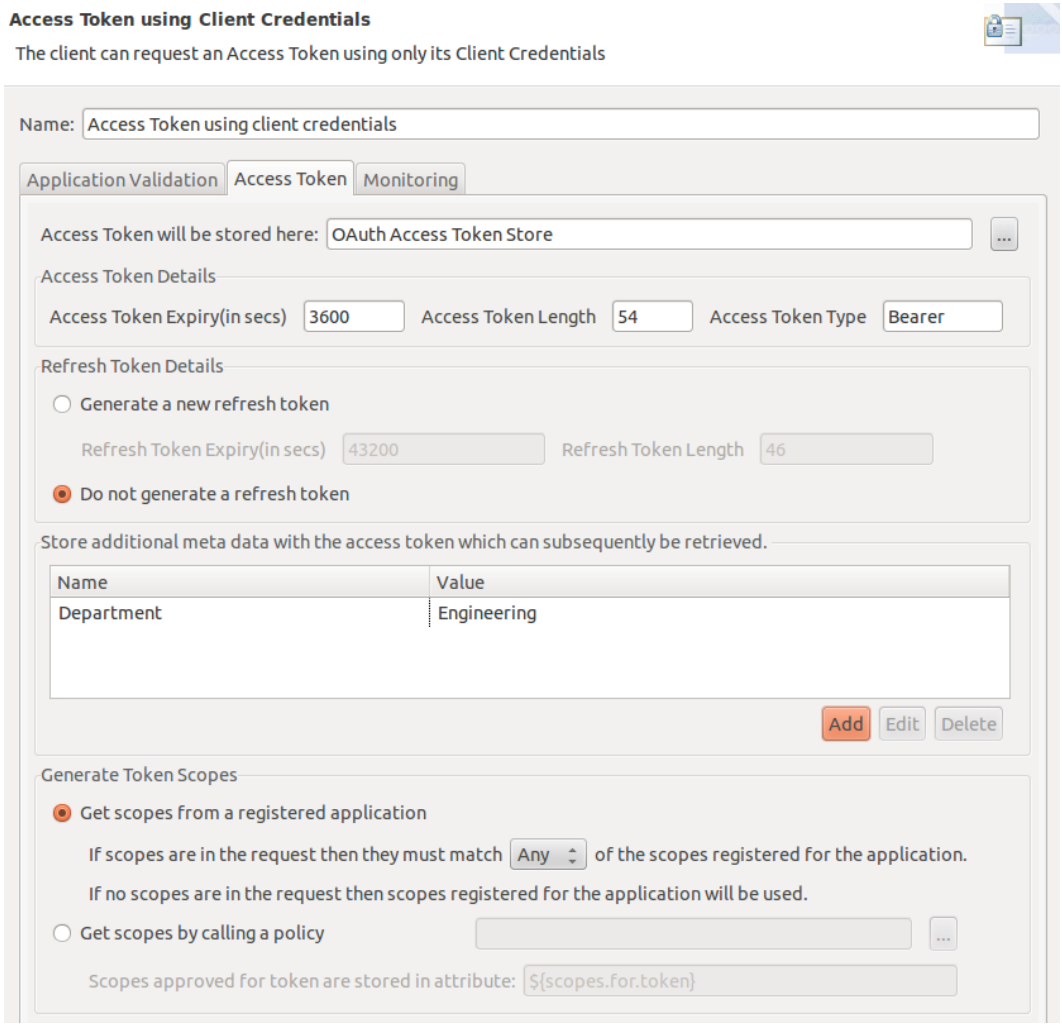
```
Demo App
Demo App Desc
https://localhost:80/images/logo.png
dce2efc8-e9d4-4976-8a0f-3d2a2ec3a26d
000-111-222-333
temp@axway.com
d0e8952f-cefe-18e1-b2bf-8accdc456933
796501dd-7df5-4a94-a111-146c7bbab22a
[https://localhost:8088/redirect]
[com.vordel.common.apiserver.discovery.model.OAuthAppScope@6a25ce50]
[com.vordel.common.apiserver.discovery.model.OAuthAppScope@6a25ce50,
  com.vordel.common.apiserver.discovery.model.OAuthAppScope@580c1ca1]
true
```

Example of querying a message attribute

If you add additional access token parameters to the OAuth **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=xxxxxxxx}",
  "Access Token Type:" :"Bearer"
```

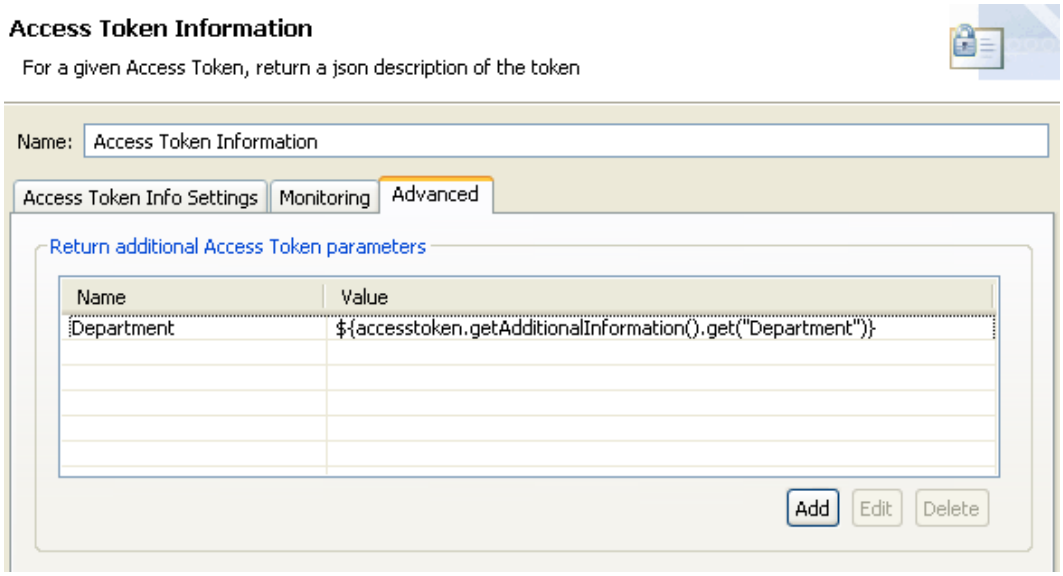
You also have the added flexibility to add extra name/value pair settings to access tokens upon generation. The OAuth 2.0 access token generation filters provide an option to store additional parameters for an access token. For example, if you add the name/value pair `Department/Engineering` to the **Client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



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:



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 Policy Developer 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 attribute. 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"
```

OAuth SAML bearer attributes

The message attribute `oauth.saml.doc` is set on the message whiteboard by the **Access Token using SAML Assertion** filter. This is a W3C DOM document view of the SAML bearer assertion that API Gateway receives from an OAuth client application. The document in this form can be verified by other filters, but in an OAuth context the **XML Signature Verification** is typically used. For more information, see the **XML Signature Verification** filter in the *API Gateway Policy Developer Guide*.

OAuth 2.0 client message attributes

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`

For more details on selectors, see the *API Gateway Policy Developer 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.getAccessToken()}
${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
 SIDnxbYabJwRZpKexUx6R3dTEwKOj0afQo7sr2DrDYuJaVCAb9xvPBk
 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
```