KELLTON TECH

# OAuth2.0 Implementation

## USING SOFTWARE AG INTEGRATION SERVER V10.1

**AUTHOR**

Venkata Siva Subrahmanyam Chavali
Sr. Consultant, Kellton Tech Solutions

# Introduction:

OAuth (Open Authorization) is a widely used "Bearer token" based API security mechanism. Starting with webMethods version 9.8, Software AG supports OAuth2.0. This whitepaper serves as a guide to understand the framework. It also explains how to secure APIs built on a Software AG Integration Server, using the OAuth2.0 framework in the context of a Delegated Authorization, as well as a standard API key. The paper will provide the necessary configurations for both server side and client side.

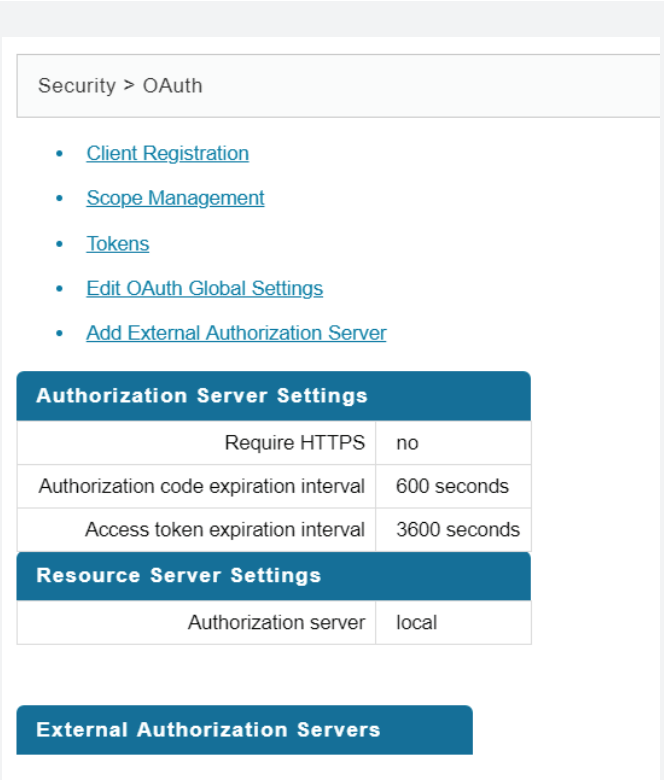



Figure 1: Authorization Server Settings

# Configuring OAuth2.0 on the webMethods Integration Server

## a. Configure Authorization Server Settings

Configure IS OAuth Authorization Server Settings as needed in your environment. As a starting point, I chose default settings. Navigate to IS Admin console ⟶ Security ⟶ OAuth. Uncheck the "Require HTTPS," but for projects, it is recommended to go with HTTPS.

## b. Client Registration

Register confidential client / public client as needed in your requirement.

- Go to IS Admin Console ⟶ Security ⟶ OAuth ⟶ Client Registration ⟶ Register Client, fill in the details, choose type as "Confidential"/ "Public," as appropriate for your use case and click on submit.

- **Redirect URI**

  URI is used by the authorization server to redirect the client after the client is authorized. It is like a service that generates token.

  This parameter is required if the client is registered with more than one redirect URI. The value for the redirect_uri must match one of the client's registered redirect URIs.

- Upon registering the client,the system generates a client_id and a client_secret. The client specific token expiration settings can be overridden at this stage.

- Return to Client Registration

**Client Configuration**

| | |
|---|---|
| ID | f53bd11391f74412b872d75cde1b3a41 |
| Secret | 5cf1265e0adb4977bbd129a82fd25795 |
| Name | DemoClient |
| Version | 1.0 |
| Type | Confidential ▼ |
| Description | OAuth Demo Purpose |
| Redirect URIs | http://localhost:5555/invoke/OAuthDemo.token/getTokenNeverExpire<br>Enter one URI per line |

**Token**

| | |
|---|---|
| Expiration Interval | ○ Use OAuth Global Setting < 3600 seconds ><br>◉ Never Expires<br>○ Expires in seconds |
| Refresh Count | ○ Unlimited<br>◉ Limit 0 |

Save

## c. Scope Definition

- The token-based authentication schemes provide an additional layer of security by giving us the ability to limit the client's scope to a particular predefined set of folders or services. To achieve that in wM IS, create a scope, add specific services and folders, and associate the client to that particular scope.

- To create a scope, Go to IS Admin Console → Security→ OAuth → Scope Management → Add Scope. In the below screenshot, I have added OAuthDemo resource to the scope, so that, the client can perform any CRUD operations on this particular resource.

Security > OAuth > Scope Management

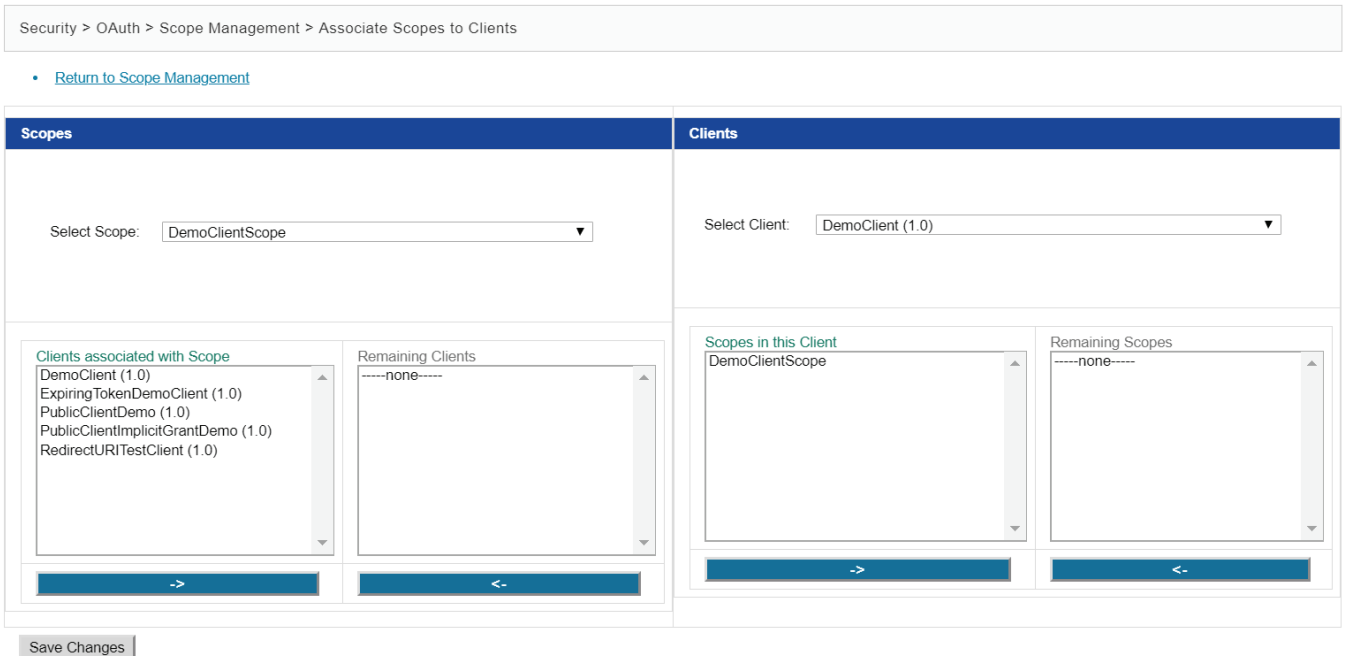
Saved scope DemoClientScope successfully.

- Return to OAuth
- Add Scope
- Associate Scopes to Clients

**Defined Scopes**

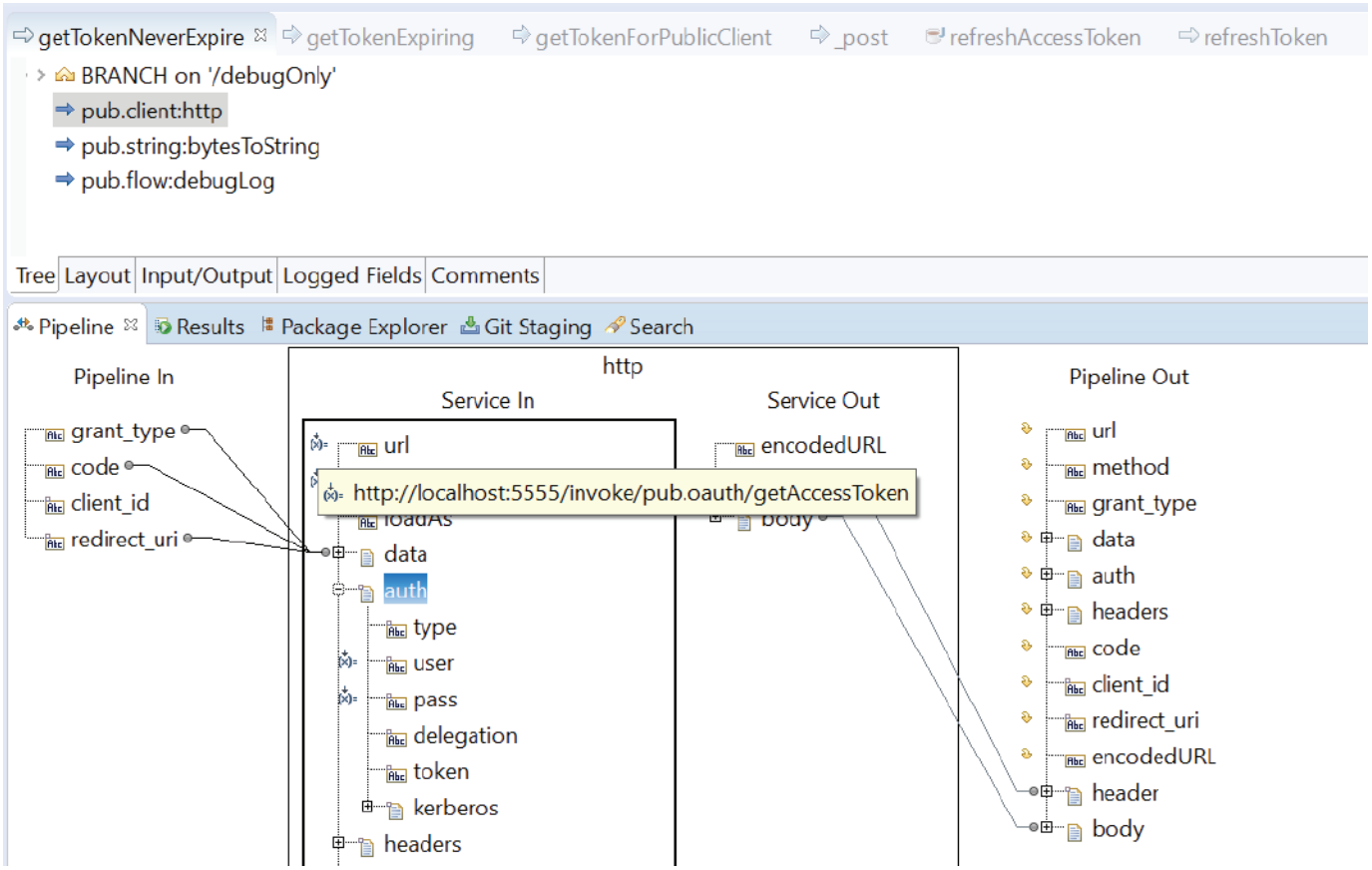
| Name | Folders and services | Delete |
|---|---|---|
| DemoClientScope | OAuthUtils.resource | ✕ |

- Associate the clients to the scope.




- **Redirect URI Sample Code**

With the above step, the OAuth configuration setup that is required for a confidential client is complete. Now, look at the below sample for the redirect URI.




- Here the pub.client:http service is used to route the client to pub.oauth:getAccessToken service over HTTP POST (the method must be post) upon the authorization approval process. Remember, the transport protocol must be https, if the OAuth global configurations are made as 'HTTPS required.'

• Client_id and client_secret must be passed as inputs for user ID and password in auth header and the data should include “grant_type,” “code,” and redirect_uri” as arguments.

## d. Client-side Activities

The client engaged in OAuth2.0 has to make multiple handshakes with the authorization server, before being able to access the resource/service on the resource server. This activity is based on the type of client and grant type chosen for communication.

Here I will use a browser to depict the client-side implementation.

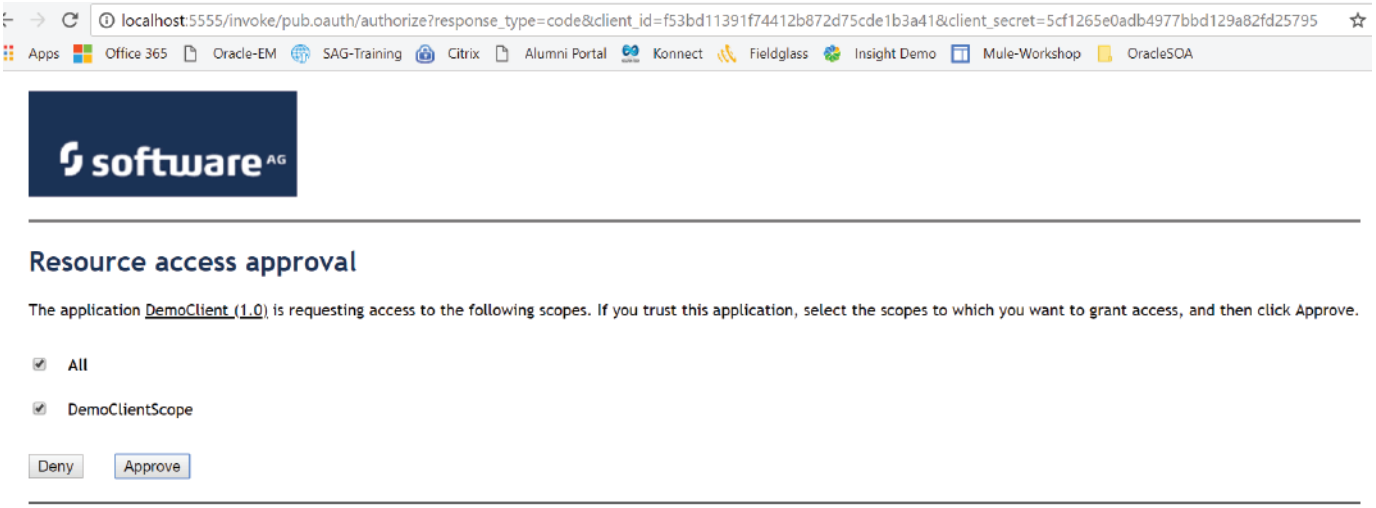
• Open browser (acting as OAuth client) and initiate OAuth Authorize request, as indicated below:

The client needs to invoke the pub.oauth:authorize built service, with the response_type, the client_id, and client_secret, which were generated at the time of the OAuth client registration on the server. The server administrator has to share these with the client application through a secure channel.

**Note**: Integration server supports two types of OAuth grant types:

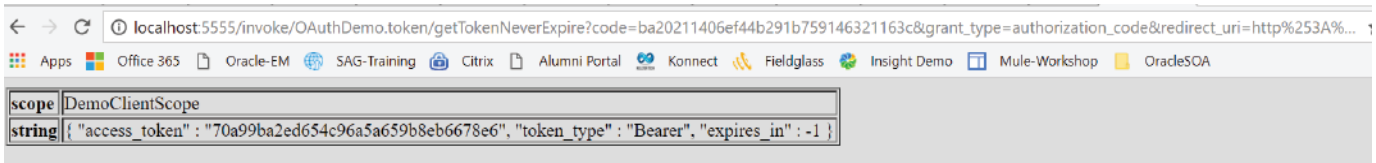
1. Authorization code: This code is used for confidential clients like web servers. The response_type for this case should be **“code.”**

2. Implicit Grant: It is used for public clients like web browsers or mobile applications; the response_type for this case should be **“token.”**



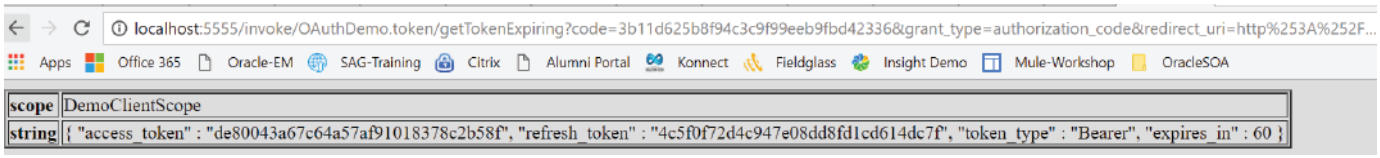

• Once the resource owner approves the request, the client will be issued an authorization code and the same will be automatically rerouted to the redirect URI mentioned at the time of client registration, without the knowledge of the client. The authorization code is available for one time use and the code is valid for 600 seconds as defined in the OAuth global settings. If the authorization code is used more than once or used after 600 seconds, then the authorization server will throw an error.

• The service located at the redirect URI, is supposed to invoke the built-in OAuth service pub.oauth.getAccessToken service (which actually generates the token) through HTTP post, while passing the client_id and client_secret as user and password in the pub.client.http service.




The above screen shot shows the access_token generated for the client. Because we chose the client token settings as “never expire,” it doesn’t have the refresh token entry.

- In case of an expiring access token, the generated token object would look like the below, which contains refresh_token as well.




- In case of an implicit grant for a public client, when the client invokes pub.oauth.authorize service, without explicit routing (as discussed in the above examples), the server implicitly routes the request to pub.oauth.getAccessToken service and gets the access token.


The client in this case, has to read the tokens through web technology like Java script, etc.

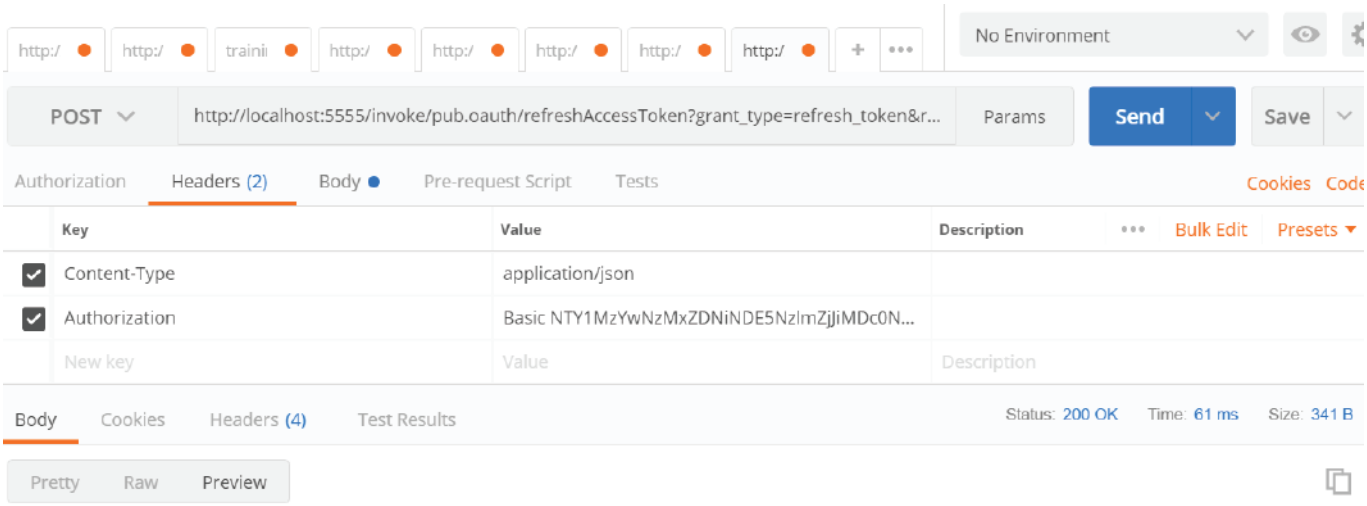
## e. Client Libraries

For any reference to server side or client side code, please refer to https://oauth.net/code/

## f. Refreshing Tokens

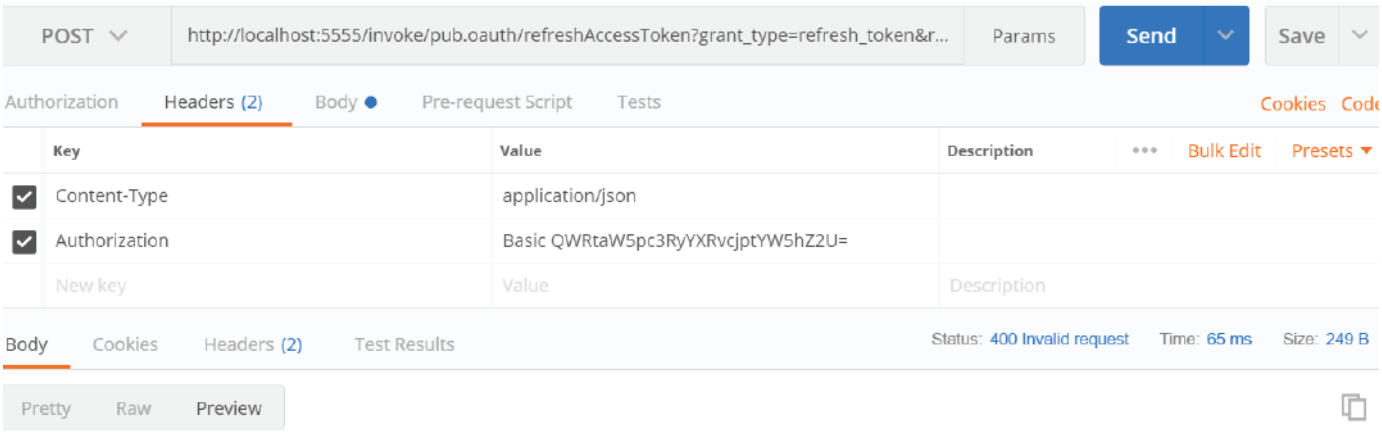
- For client-server communication, where more security is intended, it is recommended to go with short lived access tokens (the default is 1 hour). When the access token expires, the client has to make another handshake with the server, by sending a refresh token and generating a new access token. The client can renew the access tokens using the refresh tokens, as many times, as it is configured in the IS authorization server.

- To get a new access token using a refresh token, the client has to invoke pub.oauth.refreshAccessToken built-in service on the IS, through HTTP post (the method must be post). Postman is used as a client to depict the same.




{ "grant_type" : "refresh_token", "refresh_token" : "19063d7a1a46438b94311c08491773b4", "scope" : "DemoClientScope ", "access_token" : "bd0d16ae35bc414cab04361d4557e35c", "token_type" : "Bearer" }

Remember to pass the client_ID and client_secret as as the user ID and password for the service. If not, the following error would occur.

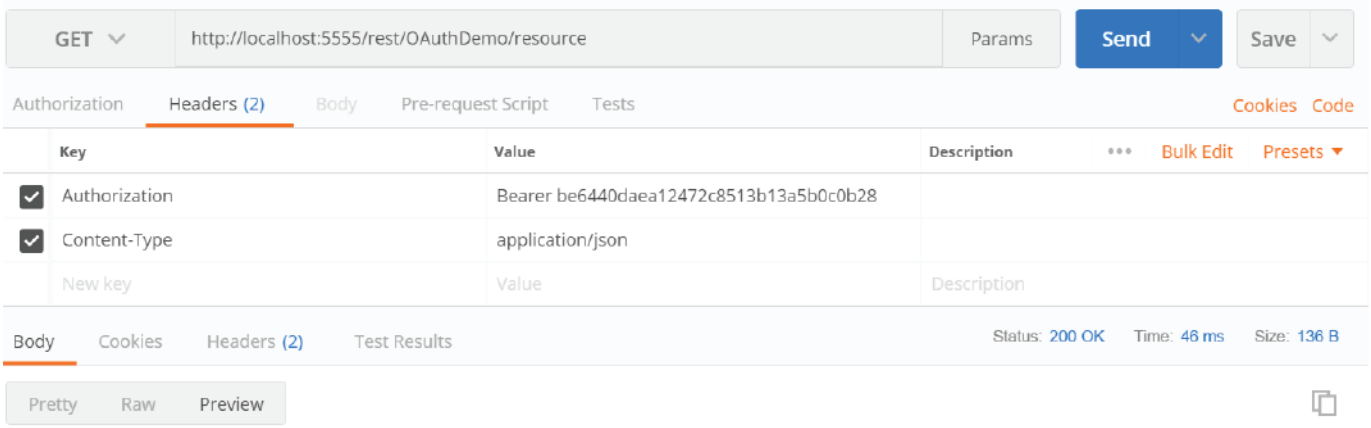



{ "error" : "invalid_grant", "error_description" : "[ISS.0010.8038] The refresh token supplied with the request was not issued to this client." }

## g. Accessing Resources using OAuth Token

Leave the permissions on the resource service as "Anonymous." Once we add a resource to the OAuth scope and associate it with the client name, the resource will become OAuth protected and won't be accessible by other consumers.

Any client can now pass the OAuth token as a Bearer token as indicated in the below screenshot and access the resource.




{"message":"get method was invoked successfully."}

# OAuth2.0 For Delegated Authorization

The best example of Delegated Authorization is "A person using a third-party photo printing mobile application to access a photo from his Facebook/Instagram account and print it, without having to share his/her Facebook / Instagram credentials with the third-party application." In this case, the server (referred to as an authorization server), would ask for the user's (referred as Resource owner's) consent (referred as Authorization) to let the third-party application access the photo (referred to as the resource). The user has to approve the request (delegating the access to the third-party application) while authenticating himself (by entering credentials) with the target server (Resource Server).

When using the webMethods integration server as an authorization server for confidential clients, the resource owner needs to be registered on the IS as a local user or as a user in the LDAP to which the IS connects for user management. The resource owner must have necessary access permissions, in order to be able to approve the request.

## OAuth2.0 as API Key

The server administrator can do one-time authorization on behalf of the resource owner/client and share the generated OAuth tokens as a standard API key. This helps in securing APIs in a simple client-server interaction use case.

The APIs hosted on the webMethods Integration Server can be secured by using the OAuth2.0 framework, when exposed to public clients such as web browsers/mobile applications or to confidential clients like applications hosted on web servers, etc.

## OAuth2.0 Built-In Services

The following services are available in the WmPublic package.

| S.No. | Service Name | Description |
|---|---|---|
| 1 | pub.oauth:authorize | Initiates an authorization request from the client application to the Integration server. |
| 2 | pub.oauth:getAccessToken | Requests an access token from the Integration server. The request includes the authorization code sent to the redirection URI by the authorization server.The authorization server validates the request and generates an access token and refresh token (optional). The tokens, along with the client identifier, expiration time, and scope are stored in the authorization server's cache. |
| 3 | pub.oauth:introspectToken | Checks to see whether an access token or a refresh token, generated by an Integration Server and used as an authorization server is active. |
| 4 | pub.oauth:refreshAccessToken | Sends a request to the Integration Server, used as an authorization server to refresh the access token. |
| 5 | pub.oauth:revokeToken | Revokes a token on an Integration Server, used as an authorization server. |

## 6. OAuth2.0 with External Authorization Servers

As an alternative to using an Integration Server, you can use a third-party server as the authorization server to authorize the clients using the OAuth tokens, generated and managed by the third-party providers.

Currently, the Integration Server can be used with an external authorization server that supports RFC 7662, OAuth 2.0 Token Introspection, including:

- Okta
- Ping Identity

## About the Author

Siva Subrahmanyam Chavali is currently working as Sr. webMethods Integration Consultant at Kellton Tech. With around 8 years of Onsite and Offshore IT experience in Enterprise Application Integration domain and webMethods Integration tool, he has strong skillset in developing, supporting, and administering Integrations in large scale ESB environment using Software AG product suite and Apple Computers' Proprietary Integration tools in UNIX / Linux / Windows environment.