Implementing OAuth on IBM WebSphere DataPower Appliances, Part 6: Grant type scenario for authorization codes

Paul Glezen (plezen@us.ibm.com)  
Consulting IT Specialist  
IBM

Shiu Fun Poon (shiufun@us.ibm.com)  
Security Architect  
IBM

Introduction

Authorization code is one of the grant types that support the traditional OAuth protocol (see Section 4.1 of the OAuth 2.0 Specification). In this tutorial, we will provide background information about the authorization code grant type and how to use WebSphere DataPower to provide this OAuth grant type support. This tutorial has been updated for firmware 6.0 features.

Overview of an authorization code grant type

As defined in the OAuth 2.0 Specification, an authorization grant type is optimized for confidential clients. A confidential client is a client that maintains the confidentiality of its credentials. With an authorization code grant type, the client must be able to interact with the resource owner's user agent (such as a browser or mobile application) and receive an incoming request with redirection. This grant type is appropriate for server side web applications that provide the required level of confidentiality. Figure 1 provides a high level overview of the authorization code grant type.
The explanation below is numbered according to the steps labeled in Figure 1.

1. Register an OAuth client application with the authorization server. This provides the OAuth Client application with a client ID and a client secret. Most likely, this step is performed by the application developer in conjunction with the API provider.
2. (Optional) The resource owner authenticates with the client application.
3. The resource owner triggers an operation requiring the OAuth client application to access the resource owner's resource that resides on the resource server.
4. The OAuth Client application redirects the resource owner to the authorization server.
5. The resource owner authenticates with the authorization server and grants permission for the OAuth client application to access the resource owner's resource.
6. The resource server sends a redirect to the resource owner's user agent. This redirect contains an authorization code for the OAuth client application.
7. The authorization code is sent to the OAuth client application with the redirect from Step 6.
8. The OAuth client application sends its credentials (from Step 1) and authorization code (from Step 7) to the authorization server.
9. After the authorization server verifies the credentials and authorization code, it issues an access token to the OAuth client application.
10. The OAuth client application uses the access token to access the resource that resides on the resource server.
11. The resource server, after validating the access token, returns the resource to the OAuth client application.
12. The OAuth client application undertakes the operation requested in Step 3.
Using the authorization code grant type

This grant type allows the resource owner to share resources with a third party client application without sharing his or her credentials. Moreover, the access permission (in the form of access token) is never shared with the resource owner. This provides additional security and assurance that access to resources is limited to the OAuth client.

The use case provided in the specification, (IETF-OAuth20) is a good example. It discusses how the resource owner wants to print pictures with a printing service (OAuth Client application). However, the pictures were located at a photo sharing site (resource server). For this scenario, the resource owner never shares the photo sharing site credentials (username, and password) with the printing service.

Where does WebSphere DataPower fit in?

In the scenario above, we position DataPower as the authorization server and as the enforcement point for the resource server.

When DataPower acts as an authorization server, it provides:

1. An authorization endpoint: This is used by the OAuth client application to obtain an authorization code from the authenticated resource owner.
2. The token endpoint: This is used by the OAuth client application to exchange an authorization code for an access_token.

When DataPower acts as an enforcement point for the resource server, it verifies the access token. DataPower only allows the request to the backend resource server if the access token is valid.

There are three distinct operations:

1. Register the OAuth client application with DataPower (Step 1 in Figure 1):
   • Create an "OAuth Client Profile".
   • Define an AAA policy to implement the authorization server:
     • Authenticate the Resource Owner (Step 5 in Figure 1).
     • For this tutorial, we will extract the resource owner's identity using "HTTP Authentication header" (Identity extraction, IE), and verify the identity using "Use DataPower AAA Info File" in the Authentication (AU) step.
   • Define an AAA policy for the resource server enforcement point.
2. Configure DataPower to provide an authorization server (Steps 5 to 9 in Figure 1).
   • Create a Web Token Service using the authorization and token AAA created in Step 1.
3. Configure DataPower to provide an enforcement point for the resource server (Steps 10 to 11 in Figure 1).
   • Create a Multi-Protocol Gateway to use the enforcement endpoint AAA policy from Step 1.
Preparing for DataPower configuration

In order to concentrate on OAuth configuration, some preconfigured DataPower objects are provided from the Downloads section of this tutorial.

1. Create a new DataPower application domain for this tutorial and switch to it.
2. Set the default log level of the new domain to information.
3. This tutorial uses the same set of shared secret and SSL keys used by the other tutorials in this series. Download OAuthArticleKeys.zip and unzip it if you haven’t already. Upload sharedSecretKey.txt, sslserver-privkey.pem, and sslserver-sscert.pem into the cert: directory of your new application domain.
4. This tutorial uses a simple XML Firewall for the backend. This is packaged in an archive named AccountLoopback.zip. Download and import this file if you haven’t already. If this XML Firewall is already running in another application domain from a previous exercise, then skip this step to avoid a conflict on TCP port 5001.
5. Download and import Part6BeginState.zip. It contains:
   • An SSL Proxy Profile, sslserver, that references key files uploaded in Step 3.
   • A shared secret key object, crypto-key, that references the shared secret key file uploaded in Step 3.
6. Verify the import:
   a. The SSL Proxy Profile sslserver state is up.
   b. The Crypto Shared Secret Key crypto-key is up.
   c. The XML Firewall loopback state is up.

You can test the loopback process by performing the following cURL GET requests. It returns a sample REST document if properly configured. You may wish to review the XML Firewall and the policy rules it contains.

```bash
curl http://<appliance-host>:5001/getCustomerInfo
returns { "id":0, "name":"", "access_token":"Unknown" }
curl http://<appliance-host>:5001/getAccountInfo
returns { "name": "myAccount", "balance":1.00 }
```

Register the OAuth client application

An OAuth client application is represented on DataPower by an OAuth Client Profile object. The registration of a client is the inclusion of its OAuth client profile within an OAuth Client Group object associated with the authorization or enforcement service. An OAuth client group is associated to a service through a reference within an AAA policy. Figure 2 illustrates this relationship.

**Figure 2. OAuth client groups, profiles, and AAA policies**

The following steps create an OAuth client group:

1. In the Control Panel, type OAuth and click OAuth Client Profile.
2. Click Add to register a new OAuth client.
3. Enter the following information in the **Main** tab shown in Figure 3:
   - **Name**: myregistered_oauthclient
   - **Generate Client Secret**: Remove the check from the checkbox.
   - **Client Secret**: passw0rd
   - **Scope**: Specify the scopes this client will support as a regular expression: ^/getAccountInfo|/getCustomerInfo
   - **Shared Secret**: Select the imported shared secret object, **crypto-key**. It is used to protect the token.
   - **Redirect URI**: List the valid redirect URIs that the OAuth client supports. For this tutorial, we will use a regular expression to match all the HTTPS URLs: ^https://\S+. Click **Add** to add the entry to the list.

**Figure 3. New OAuth client profile (Main tab)**

4. Create an "OAuth Client Group" by selecting **OAuth Client Group** from the Control Panel, as shown in Figure 4.
Implementing OAuth on IBM WebSphere DataPower Appliances,
Part 6: Grant type scenario for authorization codes

Create AAA for OAuth authorization and token endpoint

In this step, we create an AAA policy that accepts identity credentials from either basic authentication HTTP headers or via an OAuth authorization code token. This policy will later be attached to a Web Token Service that accepts requests, validates basic authentication headers, and produces the authentication code. The authorization code will then be exchanged for an access token. While you could implement this with a Multi-protocol Gateway (MPGW) service, a web token service provides automatic policy creation as discussed in Part 2 of this tutorial series.

1. From the Control Panel, in the Search Box, type AAA and click AAA Policy.
2. Click Add to add a new AAA policy for the authorization server.
3. On the Identity Extraction tab, enter AAA-AZ for the Name field.
4. Check HTTP Authentication header for the resource owner authentication method.
5. Check OAuth. This reveals the Registered OAuth clients dropdown field. Choose OAuth-Code-Client from the dropdown.
6. Select the Authentication tab.
7. Select Use DataPower AAA Info File for the method field, and select store:///AAAInfo.xml as the AAA Info File URL. This defines how to authenticate the resource owner. There are multiple users defined in store:///AAAInfo.xml. For this tutorial, we will use the pre-established identity of "fred" with a password of "smith".
8. Select the Resource extraction tab.
9. Check Processing Metadata and select oauth-scope-metadata to pick up the requested scope from the OAuth client application.
10. Click Apply to save the AAA policy for the authorization server.

Create an AAA policy for the token enforcement point

In this step, we create the AAA policy that will validate incoming OAuth access tokens for the enforcement point. The access token will have been produced by a Web Token service using the

---

5. Enter OAuth-Code-Client for the Name field.
6. Select myregistered_oauthclient in the Client drop down list. Click Add to add the selection to the Client list.
7. Click Apply.
previous AAA policy (AAA-AZ). After creating this AAA policy, we'll attach it to a MPGW service to act as the enforcement point for the resource server.

1. Click **Add** on the AAA policy page to create a new AAA policy.
2. On the **Identity extraction** tab:
   - **Name**: AAA-EP-RS.
   - **Check OAuth**.
   - From the **Registered OAuth clients** dropdown list, choose **OAuth-Code-Client**.
3. On the **Authentication** tab, the request will have been verified during **Identity extraction**. So choose **Pass Identity Token to the Authorization Step**.
4. On the **Resource extraction** tab, select **URL sent by client** and **Processing metadata** with **oauth-scope-metadata**.
5. Click **Apply**.

Starting with firmware 6.0.0.x, DataPower will automatically validate the scope against the oauth-scope-metadata with these settings. However in 5.0.0.x, this must be done with a custom stylesheet in the "Credential mapping" stage.

**Register DataPower to act as an authorization server**

In this step, we create the Web Token service that uses the AAA policy (AAA-AZ) to authenticate the resource owner using basic authentication headers. It validates the OAuth client request and produces an authorization code. For a token request, it produces an access token. The access token will later be sent to a MPGW authorizing the request to the resource server.

1. Create a Web Token Service by using the wizard. Type **Web Token Service** at the search box under the Control Panel and select **New Web Token Service**.
2. For **Web Token Service Name**, enter **oauth-wts** and click **Next**.
3. For the **Source Addresses** section, enter Port **5060**, and the SSL proxy, **sslserver**. Click the **Add** button to add the entry (Figure 5). Click **Next**.

**Figure 5. Define the port and SSL proxy profile for the authorization server**

1. For the **Source Addresses** section, enter Port **5060**, and the SSL proxy, **sslserver**. Click the **Add** button to add the entry (Figure 5). Click **Next**.

4. For the **AAA Information** panel, select the **AAA-AZ** policy that you defined for the authorization server and click **Next**.
5. Review the configuration and click the **Commit** button shown in Figure 6. Select **Done** in the next screen.
Figure 6. Commit the changes

![Create a Web Token Service](image)

**Configure DataPower as enforcement point for the resource server**

In this step, we create the MPGW that validates the access token produced by the web token service to authorize requests for resources on the resource server.

1. In the **Control Panel**, click the **Multi-Protocol Gateway** icon and click **Add** to create a new Multi-Protocol Gateway.
2. Enter the following:
   a. **Multi-Protocol Gateway Name**: resource-enforcement
   b. **Default Backend URL**: `http://127.0.0.1:5001` (the URL of the Account loopback resource server)
   c. **Request Type**: Non-XML
   d. **Response Type**: Pass through
3. Next to the **Front Side Protocol** field, click "+" and select **HTTPS Front Side Handler**. Complete these fields:
   a. **Name**: `https5061`
   b. **Port**: `5061`
   c. Check the **GET method** checkbox.
   d. For **SSL Proxy**, select `sslservr`.
   e. Click **Apply**. This closes the window and brings you back to the MPGW window.
   f. Click the **Add** button to add the front side handler to the list on the MPGW.
4. Next to the **Multi-Protocol Gateway Policy** field, click "+" to create a new policy. This starts a new panel. The completed panel is shown in Figure 7.
5. Enter a Policy Name, `mpgw`, and click **Add Policy**.
6. Click **New Rule** to create the first of two rules. This rule addresses the favicon.ico request from browsers. For **Rule Direction**, select **Client > Server**.
7. Double-click the **Match** action created initially. In the **Match Action** panel, select the `favicon` match rule and click **Done**.
8. Drag a **Results** action after the **Match** action. Double-click the action and change the **Method** field to **GET**.
9. Click the **Advanced** tab of the **Results** action and change the **Output type** to **binary**. Click **Done**.
10. Click **New Rule** to create the second of two rules. This rule verifies the access token for the resource access. For **Rule Direction**, select **Client > Server**.

11. Double-click the lone **Match** action. In the **Match Action** panel, select the **matchAll** match rule and click **Done**.

12. Drag an **Advanced** action onto the rule after the Match action. Double-click it, select **Convert Query Params to XML**, click **Next**, and then **Done**. This action converts the query parameters in the URL to an XML format for input to the subsequent AAA policy.

13. Drag the **AAA** action onto the rule after the **Convert** action. Double-click it and select the **AAA-EP-RS** policy from the dropdown. Click **Done**.

14. Drag the **Results** action onto the rule after the **AAA** action. Double-click it and change the **Method** field to **GET**.

**Figure 7. Create a Multi-Protocol Gateway Policy**

![Configure Multi-Protocol Gateway Style Policy](image)

15. Click **Apply Policy** and close its window.

16. Click **Apply** on the Configure Multi-Protocol Gateway window.

17. Save the configuration.

This completes the OAuth configuration.

**Testing the configuration**

Testing the authorization code grant is more involved than **Part 4**, resource owner password credential scenarios, and **Part 5**, client credential scenarios. In this part, we have to simulate three distinct parties: (1) the client, (2) the resource owner, and (3) the token/resource server (combined as one for this exercise). This tutorial provides three options for testing the authorization code grant scenario.

1. **cURL**: This command line tool is easy to install and is in wide-spread use. However, for this exercise, it will require the most copy-and-paste activity.
2. **Node.js**: A Node.js server is provided to act as the client. The Node.js program runs from the command line and prints the information it is sending and receiving. It annotates its output with the step numbers in Figure 1.

3. **PHP**: A PHP client is provided to act as the client. All interactions (including those between client and resource server) happen through the browser.

The cURL tool is the easiest to install, but it requires the most manual effort to complete the test. This extra manual effort does provide additional insight into the details of the exchanges. Node.js and PHP require you to install their respective runtimes. The Node.js option is closest to what one encounters in practice.

**Testing with cURL Client**

This section demonstrates testing of our configuration using the cURL tool. This process simply shows the results of each service invocation. You need to copy the authorization code tokens and access tokens between steps. The DataPower IP address will be referenced as "[DP_IP]" in the samples below. The imaginary client IP is "1.2.3.4".

1. Initiate an authorization request from the resource owner. This corresponds to Step 5 of Figure 1, which would normally be the second part of a redirect. You simulate Step 4 by providing the credentials with the `--user` parameter. The `-o test.html` sends the output to `test.html` rather than directly to the screen. Since you have no actual client to redirect to, just specify `https://1.2.3.4` for the IP address.

   ```
curl -k "https://[DP_IP]:5060/authen?response_type=code&client_id=myregistered_oauthclient&redirect_uri=https://1.2.3.4&scope=/getAccountInfo%20/getCustomerInfo&state=MyTest" --user fred:smith -v -o test.html
   
   curl -k "https://[DP_IP]:5060/authen?response_type=code&client_id=myregistered_oauthclient&redirect_uri=https://1.2.3.4&scope=/getAccountInfo%20/getCustomerInfo&state=MyTest" --user fred:smith -v -o test.html
   ```

2. Use a browser to open the output file test.html. Choose **Allow access** and **Submit** as shown in Figure 8. Wait for the browser to timeout since "1.2.3.4" is not a valid IP address.

**Figure 8. Authorization challenge**

![Authorization challenge](image)

If your browser does not trust your DataPower certificate, you have to add it as an exception. You should receive an error from the browser, since the 302 HTTP redirect from the DataPower authorization endpoint cannot reach 1.2.3.4. However, you can copy the code from the browser header, as shown in Figure 9. Make sure to start the copy after "code=" and stop before the next parameter that starts with an ampersand (&).
3. The next step is to convert the authorization code to an access token. You can use the basic authentication header (--user parameter) to pass in the client_id and client_secret parameters. The authorization code must be URL-encoded when it is sent back to the token endpoint on the DataPower appliance. Since it was already URL-encoded within the redirect, it can be copied from the redirect URL and pasted in place of %CODE_FROM_BROWSER% as shown below.

```
curl -k https://[DP_IP]:5060/token -d "grant_type=authorization_code&code=%CODE_FROM_BROWSER%&redirect_uri=https://1.2.3.4" --user myregistered_oauthclient:passw0rd
```

From the above command, you should receive an access token that can be used for accessing the resource.

```
{"token_type":"bearer","access_token":"AAEYbX+y......","expires_in":3600,"scope":"/getAccountInfo /getCustomerInfo"}
```

4. You can use the access token to retrieve the resource. Invoke a GET request through the MPGW enforcement point, which will respond with the requested resource. But, the access token in the previous JSON response is not URL-encoded. It must be URL-encoded before including it in the request. An online utility for URL-encoding is listed in the Resources section.

```
curl -k https://%DP_ID%:5061/getAccountInfo -H "Authorization:Bearer %URL-ENCODED_ACCESS_TOKEN%"
```

Testing with the Node.js client

Download the Node.js client from its OAuth Clients hosting site. Follow the instructions for unzipping the archive and starting the client on the command line. The client command line output will display a "Client App Homepage" for Part 6 (this tutorial). While following the steps below, it is recommended to have the command line visible as you interact with the browser so you can see what the client is doing behind the scenes.

1. Point your browser to the client app homepage URL for Part 6 as listed in the Node.js output. You will have to agree to accept the self-signed SSL certificate before the page will display.
2. Click the Begin Step 3 button.
3. Provide `fred/smith` for the username/password at the browser password challenge.
4. After allowing access, click **Submit**.

After the client receives the authorization code, it responds to the browser with a simple text message. The rest of the interaction between the client (Node.js) and DataPower is displayed on the command line.

**Testing with PHP client**

A PHP OAuth client and its corresponding HTML page (code.php and code.htm) are provided in the [Part6Supplements.zip](#) archive with this tutorial. The PHP client simulates an OAuth client that supports the authorization code grant type. The PHP client should be running on WebServer with support for curl, PHP, and SSL. The TCP ports in your configuration may differ from the screenshots in this section.

1. Access the code.htm on the web server, `http://<yourwebserver>/code.htm` (see Figure 10):
   - **Authorization Endpoint**: Authorization endpoint hosted by the Web Token Service (WTS) from above: `[DP_IP]:5060/authen`.
   - **client_id**: The sample client ID used for this exercise.
   - **redirect_uri**: The URI of the PHP client to which the browser will be redirected after a successful grant by the resource owner.
   - **scope**: The scopes granted to the client by the resource owner.

   **Figure 10. Specify the authorization endpoint and other information**

2. Click **Do It**. The authorization server, which is a WTS listening at 5060 (not 8080 as shown in Figure 10), will prompt for the resource owner’s authentication.
3. Use `fred/smith` as the resource owner username/password as shown in Figure 11. Click **OK**.
4. If the authentication in Step 2 is successful, the resource owner is prompted for permission to share his resource for the cURL case, as shown in Figure 8. Click **Allow access** and then **Submit**.

The response from the authorization server is redirected back to the PHP client. The response from that redirect is shown in Figure 12. The authorization code that was received is displayed after "code". The redirect_uri and endpoint are displayed as well as the client secret and the client ID. This is for the OAuth client to request an access token based on the authorization code it received earlier.

**Figure 12. Requesting for an access_token with the code**

![Authorization Code Grant Example Client](image)

5. The **client_secret** field in Figure 12 is used to communicate the client secret to the test client. Of course, a real client would not need to be told its password. This is just to make the test client more flexible for test cases. Click **Do It** to submit the request to the client to access the resources on the resource server.

The OAuth client requests an access token from the authorization server. Figure 13 shows the OAuth client revealing the access token back to your browser (the resource owner). This is for educational purposes only. Normally, the OAuth client would instead issue a resource request...
directly to the resource server with the access token (as the Node.js client does). Notice how the resource is enforced by the appliance based on the setting in the AAA (“Resource Extraction” or “Resource Mapping”).

**Figure 13. Receive an access token**

![Authorization Code Grant Example Client](image)

6. Change the resource request to `/getCustomerInfo` as highlighted in Figure 14.
7. Click **Do It**.

**Figure 14. Access resource /getCustomerInfo**

![Authorization Code Grant Example Client](image)

The response from `/getCustomerInfo` shows a subset of the information in the access token (Figure 15).

**Figure 15. Resource owner information in the access token**

![Authorization Code Grant Example Client](image)
This response was returned to the OAuth client. The OAuth client is providing it to your browser for illustrative purposes only. The exchanges shown in Figures 14 and 15 would normally occur exclusively between the OAuth client and the resource server as demonstrated with the Node.js sample.

Conclusion

In this tutorial, we covered the motivations for using an authorization code grant type. This grant type is appropriate for server-side web applications, which maintain the confidentiality of client credentials. This grant type also provides additional security benefits, including the ability to authenticate the OAuth client and pass the access token to the OAuth client without exposing it to the resource owner.

Acknowledgments

The authors would like to thank John Rasmussen and Peter Argue for their review of this tutorial.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code sample</td>
<td>OAuthArticleKeys.zip</td>
<td>3KB</td>
</tr>
<tr>
<td>Code sample</td>
<td>Part6BeginState.zip</td>
<td>516KB</td>
</tr>
<tr>
<td>Code sample</td>
<td>AccountLoopback.zip</td>
<td>499KB</td>
</tr>
<tr>
<td>Code sample</td>
<td>Part6Supplements.zip</td>
<td>4KB</td>
</tr>
</tbody>
</table>
Resources

- IBM WebSphere DataPower SOA Appliance Handbook
- IETF Web Authorization Protocol (oauth)
- IETF Bearer Token
- string-functions.com: URL-encode
- Node.js OAuth Article clients
- WebSphere DataPower Knowledge Center
- WebSphere DataPower library
- developerWorks WebSphere zone
- WebSphere DataPower discussion forum
About the authors

Paul Glezen

Paul Glezen is an IT Specialist with IBM Software Services for WebSphere (ISSW). He started application server consulting in 1998 with the IBM CORBA-based C++/Java product known as Component Broker. His current product focuses are DataPower and WebSphere Application Server development and security configurations.

Shiu Fun Poon

Shiu Fun Poon is the Security Architect in the WebSphere DataPower Appliance development team. She has worked in many different areas and features of DataPower appliances. She currently leads DataPower's support for OAuth 2.0 functionality. Her past DataPower projects included WS-Security Policy, various WS-Security features, appliance RAS features, 7198/7199 appliance release, and third party integration with the appliance.

© Copyright IBM Corporation 2012, 2015
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)