Implementing OAuth on IBM WebSphere DataPower Appliances, Part 10: Customization scenario for additional extension points

Shiu Fun Poon (shiufun@us.ibm.com)
Senior Technical Staff Member
IBM

06 May 2015

The final part of this multi-part series describes how you can use extension points for fine-grained control of WebSphere DataPower's native OAuth support.

View more content in this series

Introduction

In firmware version 6.0, WebSphere® DataPower provides extension points to customize the standard handling of OAuth protocol phases. In this tutorial, we will cover how to extend or customize the operations. You can enable this feature by selecting a stylesheet value for the Additional OAuth Process field as shown in Figure 1. This field appears on the Advanced tab of the OAuth client profile object.

Figure 1. OAuth extension point field

For DataPower to act as an authorization server, it has to provide the following functions:

- Issue and verify an authorization code.
- Issue and verify an access token.
- Issue and verify a refresh token (new to firmware 6.0.0).
- Support revocation by either the OAuth client or by the resource owner.
- Support token introspection.

By populating the Additional OAuth Process field, you can provide a custom stylesheet that supports the operations that you wish to override. If the operation is not specified in the stylesheet, DataPower will use its standard behavior.
The supported operations are:

- **authorization_request**: This is invoked after an authorization code is generated successfully (as in an authorization code grant type). It allows additional information to be returned to the OAuth client along with the authorization code.
- **access_request**: This is invoked after an access token is generated successfully. It allows additional customized name-value pairs to be returned to the OAuth client as part of the JSON object containing the access token.
- **resource_request**: This is invoked after an access token has been verified successfully. It allows additional information to be sent to the backend resource server for further processing. This is mostly likely done with additional HTTP header values. For example, the resource owner identity could be sent to the backend.
- **authorization_form** (6.0.0 firmware): This is invoked after the authorization form is returned to the appliance for OAuth grant types of "Authorization Code" or "Implicit." This allows a customer to handle the condition in which the resource owner grants access to a subset of the requested scopes from the OAuth client.
- **preapproved_check** (6.0.0 firmware): This provides a way to by-pass the authorization permission request with the resource owner. It is useful with the "Authorization Code" and "Implicit Grant" types when you have a framework that allows the resource owner to pre-authorize or pre-deny request for a given OAuth client under certain conditions defined by you.
- **validate_request** (6.0.0 firmware): This is invoked after an access token is verified successfully. The user can optionally add attributes or perform additional handling. This is similar to the access_request extension point mentioned earlier. Its use is restricted to the DataPower validate grant type discussed below in the validate_request section.
- **miscinfo_request** (7.0.0 firmware): This allows additional miscellaneous information to be added as part of the either authorization code, access_token (also as refresh_token). The maximum characters that can be added is limited to 512 characters.

The following two operations will only be used when "Caching" is set to "Custom" in "OAuth Client Profile". This indicates that you want to provide a custom process to handle revocation.

- **revoke_request** (6.0.0 firmware): This allows a customer to provide a custom plugin when a revocation request is received by the appliance.
- **check_revocation_request** (6.0.0 firmware): This is called during a token verification to determine whether it has been revoked. It is the counterpart of the previous operation, revoke_request.

**authorization_request operation**

In the authorization code grant type scenario, a temporary authorization code is provided to the OAuth client to be exchanged for the access token.

The authorization code is returned to the OAuth client using the following HTTP 302 redirect:

```
HTTP/1.1 302 Processed
Location: https://<client-hostname-redirect-url>?code=........&state=........
```
The **authorization_request** extension point allows you to append additional information to the authorization code grant response.

**Input:**

```
<operation>authorization_form</operation>
<oauth-id>........</oauth-id>
<OauthSupportedClient>........</OAuthSupportedClient>
```

**Expected output:**

```
<result>
  <name1>text</name1>
  ...
  <name-9>text</name-9>
</result>
```

With the above output, the 302 redirect will be augmented as follows:

```plaintext
HTTP/1.1 302 Processed
Location: https://<client-hostname-redirect-url>?code=........&state=........&name1=text&name-9=text
```

**access_request operation**

For all the supported grant types, at the end of a successful OAuth handshake, an access token will be returned in a JSON object or as part of the redirect URI:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token": "........",
  "token_type": "bearer",
  "expires_in": 3600,
  "refresh_token": "........"
}
```

The **access_request** extension point allows you to add additional information to an access token response, or handle additional processing. In the **stylesheet sample** provided with this tutorial, it adds additional information to the access_token payload response.

**Input:**

```
<operation>access_request</operation>

<result>
  <access_token>........</access_token>
  <expires_in type='json:number'>........</expires_in>
  <scope>........</scope>
</result>
```
Expected output:

```xml
<result>
  <custom1>text</custom1>
  ...
  <custom9 type='json:number'>88</custom9>
</result>
```

With the above output, the access token JSON object will be augmented to the following:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"........",
  "token_type":"bearer",
  "expires_in":3600,
  "refresh_token":"........",
  "custom1":"text",
  ...
  "custom9":88
}
```

**resource_request operation**

After an access_token is verified successfully by the appliance, but before the request is sent to the backend application server, this operation provides an opportunity to handle the payload before the request is sent. This operation does not expect an output, and any output returned will be ignored by DataPower. You can use this to set additional HTTP headers for the backend application server.

**Input:**

```xml
<input>
  <operation>resource_request</operation>
  ....
  <container>
    same input for AAA – PostProcess Custom stylesheet ...
  </container>
</input>
```

**Expected output:** N/A

**authorization_form operation**

For authorization code and implicit grant types, the resource owner will be presented with an authorization form in which the resource owner can grant access to the requested resources by the OAuth client. By default, DataPower allows the resource owner to grant only a subset of the requested scope by the client.

The **authorization_form** extension point allows you to fine-tune this choice when the authorization form is returned to the appliance.
Input:

```xml
<input>
  <operation>authorization_form</operation>
  <oauth-id type="authorization_request">
    <response_type>...</response_type>
    ........
    <authorization-request>
      <args src="body">
        ........
      </args>
    </authorization-request>
  </oauth-id>
  <OAuthSupportedClient>.....</OAuthSupportedClient>
</input>
```

Expected output:

```xml
<result>
  <scope>....supported scopes separated by space....</scope>
</result>
```

The scopes specified in the expected output will be used as the supported scopes for the OAuth handshakes.

**preapproved_check operation**

For authorization code and implicit grant types, an authorization form will be presented to the resource owner for approving the client's request. DataPower supports pre-approving or pre-denying such requests. By marking whether a request has been pre-approved or pre-denied, DataPower will skip the authorization form presentation to the resource owner and proceed to the next step in the OAuth process.

The **preapproved_check** extension point allows you to short-circuit the authorization grant process so that some requests can be approved or denied. These cases will not be presented with an authorization form when the client is trying to access the resource.

Input:

```xml
<input>
  <operation>preapproved_check</operation>
  <container>
    .... same input for AAA - PostProcess Custom stylesheet ...
  </container>
</input>
```

Expected output:

```xml
<result><approved>yes|no|unknown</approved></result>
```

The choices have the following meanings:

- **yes**: The request has been pre-approved, the authorization form will not be presented.
- **no**: The request has been pre-denied, the authorization form will not be presented.
• **unknown**: The authorization form will be presented.

### validate_request operation

As of the 6.0.0 firmware release, DataPower provides a way for OAuth clients to verify an access token in their possession. A new grant type, `urn:ibm:datapower:validate`, was added to support this validation request. See the sample curl command below for details on how to invoke DataPower with this custom grant type.

The **validate_request** extension point allows you to add additional information or perform additional processing to the response after an access token has been verified successfully.

For the following request:

```bash
curl -k -v https://<dp>:<port>/<uri>?client_id=<client_id>&grant_type=urn%3Aibm%3Adatapower%3Avalidate&access_token=......
```

This is the output example:

```json
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store, no-cache
Pragma: no-cache

{
	"valid":true,
	"token_type":"bearer",
	"client_id":"<client_id>",
	"not_after":"178947675",
	"not_after_text":"2012-09-02T03:41:15Z",
	"not_before":178944075,
	"not_before_text":"2012-09-02T02:41:15Z",
	"resource_owner":"spoon",
	"scope":"/getAccountInfo"
}
```

**Input:**

```xml
<result>
	<extra1>customerid</extra1>
	...
	<extra8 type='json:number'>8888</extra8>
</result>
```

With the above output, the response will be augmented to the following:
miscinfo_request operation

As of the 7.0.0 firmware release, DataPower provides a way to add additional information as part of the crypto protected token. This allows you to add up to 512 characters of data. For example, if there is a session that is persistent in a database somewhere, you can use this field to store the index of the entry to the database.

- **@type**: This indicates when the OAuth protocol of this operation is called.
  - **dp-state**: This is part of the information for the authorization/consent form when it is presented. If the SSO session is tracked in a database for the resource owner, you can add the SSO session ID to the miscellaneous information.
  - **az-code**: This is part of the information for the authorization code
  - **access-token**: This is part of the information for access_token, which also impacts refresh_token.

If the miscellaneous information is added during dp-state, DataPower will automatically carry the value forward to az-code and access-token. However, you can override that information during @type=az-code, or @type=access-token if needed.

If the miscellaneous information is added during az-code, DataPower will automatically carry the value forward to access_token and refresh_token, unless it is overridden later during @type=access-token.

Input:

```xml
<operation @type='dp-state|az-code|access_token'>
    validate_request
</operation>
<container>
    .. same input for AAA - PostProcess Custom stylesheet ......
</container>
</input>
```

Expected output:
To use the customized revocation support on the persisting revocation information, you can upgrade to the following fixpacks of the DataPower firmware: 6.0.0.15, 6.0.1.11, 7.0.0.8, 7.1.0.5. The revocation is broken down into three distinct operations. In this section, we will cover how to persist the revocation information in a persistent storage off DataPower, so you can provide revocation support in a high availability (HA) environment. Given any file access, network access is one of the most time consuming tasks for an application. DataPower will only trigger the following operation when it is absolutely necessary.

In the majority of cases, the revocation of the access is only a small subset of all the access that was granted by different resource owners to the many applications. Storing all the authorization tokens from OAuth for this purpose does not make sense. By providing the option to track the revocation, DataPower provides some flexibility on handling the revocation (instead of a "one size fits all" solution).

- Handling a revocation request (revoke_request)
- Determining whether a client or the token has been revoked (check_revocation_request)
- Recording the revocation during a normal OAuth protocol handling (access_request)

In order to provide the best performance and allow HA flexibility in a cluster environment, DataPower will utilize its runtime cache first before it will use any of the above operations for the revocation check, as shown in Figure 2.

**Figure 2. Revocation check**
revoke_request operation

This handles the recording of the revocation. There are two types of revocation: the client/application against the tokens it received, and the resource owner against a client/application. This operation is triggered after DataPower has confirmed the token is valid and has not expired yet. Because of this design, the expensive and time consuming file-based and network-based operation is only incurred after the token is deemed worthwhile for this overhead. You can take advantage of this behavior to make your implementation more robust. For example, for the resource owner revocation, consider storing the timestamp when the resource owner has revoked the permission. When checking a token for revocation (in operation check_revocation_request), use this information to reject any tokens issued before the above timestamp. This allows the application with the new permission to access the resources.

Input for the client revoking its token:

```
<operation>
  <operation>revoke_request</operation>
  <!-optional access_token and refresh_token, since client can revoke access_token only, refresh_token only, or both  <access_token count="1" src="body">...</access_token>  <refresh_token count="1">....</refresh_token>
  <client_id src="basic-auth" count="1">client</client_id>
  <oauth-id type="client_revoke_request">
    <access_token count="1" src="body">...</access_token>
    <refresh_token count="1">....</refresh_token>
    <grant_type count="1">urn:ibm:datapower:client:revoke</grant_type>
    <client_id src="basic-auth" count="1">client</client_id>
    <client_secret src="basic-auth" sanitize="true" count="1">..</client_secret>
    <original-url count="1" type="request">https://dp:7777/revoke</original-url>
  </oauth-id>
  <OAuthSupportedClient>....</OAuthSupportedClient>
</input>
```

For the resource owner revoking a client's permission:

```
<operation>
  <operation>revoke_request</operation>
  <resource_owner>tonyf</resource_owner>
  <client_id count="1" src="body">client</client_id>
  <entry type="oauth">
    <oauth-id type="owner_revoke_request">
      <grant_type count="1">urn:ibm:datapower:owner:revoke</grant_type>
      <client_id count="1" src="body">client</client_id>
      <original-url count="1" type="request">https://dp:7777/revoke</original-url>
    </oauth-id>
    <OAuthSupportedClient><client-id>..</client-id>......</OAuthSupportedClient>
    <oauth-verified state="ok">
      <result>
        <grant_type count="1">urn:ibm:datapower:owner:revoke</grant_type>
        <client_id count="1" src="body">client</client_id>
      </result>
      <oauth-verified>
        <entry>
          <input>
        </entry>
      </oauth-verified>
    </oauth-verified>
  </entry>
</input>
```

Expected output:

```
<result><status>success|failure</status></result>
```
check_revocation_request operation

During the OAuth or resource access processing, DataPower will need to verify whether the token (or access) has been revoked. This operation provides the needed information for DataPower to proceed. This operation is only called when DataPower has verified the token is valid, and it has not been revoked due to reuse within the DataPower runtime cache.

Input:

```xml
<input>
  <operation token-type="access_token|refresh_token|az-code|dp-state">check_revocation_request</operation>
  <token>....</token>
  <verified-token>
    <client_id>...</client_id>
    <not_after>...</not_after>
    <not_after_text>...</not_after_text>
    <not_before_text>2014-08-08T08:08:08Z</not_before_text>
    <resource_owner>xx</resource_owner>
    <scope>xx</scope>
    <miscinfo>..</miscinfo>
  </verified-token>
  <oauth-id type="...">.....</oauth-id>
  <OAuthSupportedClient>...</OAuthSupportedClient>
</input>
```

Output if the token is revoked:

Having a node `<revoked/>` indicates that the token is revoked.

```xml
<result><revoked/></result>
```

If the token is not revoked, there is no need to return any nodeset.

access_request operation

This operation is optional. However, we highly recommend that you use it. This operation occurs when an access token is generated for the verified operation. There are two cases that you should consider to save the information in external storage: when `access_token` is created as part of the authorization code grant type, or for a `refresh_token` grant type. You may want to keep a copy of the authorization code or `refresh_token` in the same revocation store for the other tokens.

Conclusion

The last part of this series, Part 10, described the operations that you can override to modify the standard DataPower OAuth support. You can use the `stylesheet` that is provided with this tutorial to use the extension points, either individually or combined in different combinations, to meet your business requirements.

Acknowledgments

The author would like to thank Paul Glezen for reviewing this article.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code sample</td>
<td>code_sample.zip</td>
<td>7KB</td>
</tr>
</tbody>
</table>
Resources

- OAuth 2.0 IETF specification
- Regular expression
- WebSphere DataPower Knowledge Center
- WebSphere DataPower library
- developerWorks WebSphere zone
- WebSphere DataPower discussion forum
About the author

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 2015
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)