OpenID Connect Dynamic Client Registration

OpenID Connect enables clients (relying party) to verify the identity and obtain profile information of the end user based on the authentication performed by an authorization server, in an interoperable and REST-like manner. Client applications must be registered before they can login end-users with OpenID Connect or receive OAuth 2.0 access tokens. The following section provides information on how an OpenID Connect Relying party can dynamically register with the end user's OpenID provider.

How it works

To register a new client at the authorization server:

1. Client discovers the client registration endpoint. For more information on how to do this, see OpenID Connect Discovery Support. The registration endpoint URL discoverable via WebFinger. For more information on discovering the client registration endpoint, see OpenID Connect Discovery.
2. Client sends an HTTP POST message to the client registration endpoint with and client metadata parameters that the client chooses to specify for itself during the registration.
3. The authorization server assigns a unique client identifier (client ID) and optionally a client secret.
4. The authorization server associates the metadata given in the request with the issued client ID.

Sample Requests

To register, the client sends an HTTP POST message to the client registration endpoint. A sample request and response is shown below.

<table>
<thead>
<tr>
<th>Send Registration Request</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sample Request</strong></td>
</tr>
<tr>
<td>POST /identity/connect/register</td>
</tr>
<tr>
<td>Content-Type: application/json</td>
</tr>
<tr>
<td>Accept: application/json</td>
</tr>
<tr>
<td>Host: server.example.com</td>
</tr>
<tr>
<td>Authorization: Basic YWRtaW46YWRtaW4=</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;redirect_uris&quot;: [&quot;server.example.com&quot;],</td>
</tr>
<tr>
<td>&quot;client_name&quot;: &quot;application_1&quot;,</td>
</tr>
<tr>
<td>&quot;ext_param_owner&quot;: &quot;application_owner&quot;,</td>
</tr>
<tr>
<td>&quot;grant_types&quot;: [&quot;password&quot;]</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
If the request is successful, you will see the following response:

```json
HTTP/1.1 201 Created
{
  "client_id": "s6BhdRkqt3",
  "client_secret": "ZJYCqe3GGRvdrudKyZS0XhGv_Z45DuKhCUk0gBR1vZk",
  "client_secret_expires_at": 1577858400,
  "redirect_uris": [
    "server.example.com"
  ],
  "client_name": "application_owner_application_1"
}
```

If there is an error, you will see the following response:

```json
HTTP/1.1 400 Bad Request
{
  "error": "invalid_redirect_uri",
  "error_description": "One or more redirect_uri values are invalid"
}
```

To register with multiple callbacks, use the following:

### Send Registration Request

#### Sample Request

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
<th>Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/identity/connect/register</td>
<td>Content-Type: application/json, Accept: application/json, Host: server.example.com, Authorization: Basic YWRtaW46YWRtaW4=</td>
</tr>
</tbody>
</table>

#### Request Body

```json
{
  "redirect_uris": [
    "regexp=(https://server.example.com/callback|https://testapp:8000/callback)"
  ],
  "client_name": "application_1",
  "ext_param_owner": "application_owner",
  "grant_types": ["password"]
}
```

To de-provision itself on the authorization server, the client sends a HTTP DELETE request to the client configuration endpoint.

### Client Delete Request
You need to specify the `userId` and `applicationName` parameters in the URL path variable as follows:

```
DELETE
/identity/register/TYxVD3VyYYkNtS3PYt0FI5l0rQYa?userId=admin&applicationName=admin_application_2
```

Following are the descriptions of the parameters you need to specify in the delete request:

- **userId**: The user name of the user who has logged in.
- **applicationName**: The name of the application that you want to delete.