Header-Based JWT Claim Mapping

Use cases:

To pass extra parameters like channel\_id from backend during user authentication process and add it as claims to JWT. To cater to this specific requirement, we have developed a custom SPI for Keycloak, that maps parameters received as HTTP headers into JSON Web Token (JWT) claims. This data will be sent to protocol/openid/tokenendpoint .

Steps involved:

Build a Keycloak provider package (.jar) using the keycloak extension interface. (Source code available in Azure repo). Copy the Jar file to the provider folder of keycloak and restart the application.

* **Client scope configuration**

1. Navigate to client scopes >Create client scope
2. Provide appropriate name (**eg. custom headers)**
3. choose openID protocol
4. Set Include in token scope if you wish the name of client scope to be present in JWT.
5. Navigate to Mappers > Add mapper >select “configure a new mapper” > select final **Custom custom Header to Claim Mapper**
6. Provide name and header name(channel\_id). This is the parameter that will be passed to the token end-point as header.
7. Select “add to access token” and any other relevant options

* **Client Configuration**
  + Navigate to clients >select client > Client scopes > Add client scopes > select the client scope created in previous step
  + Set assigned type to optional

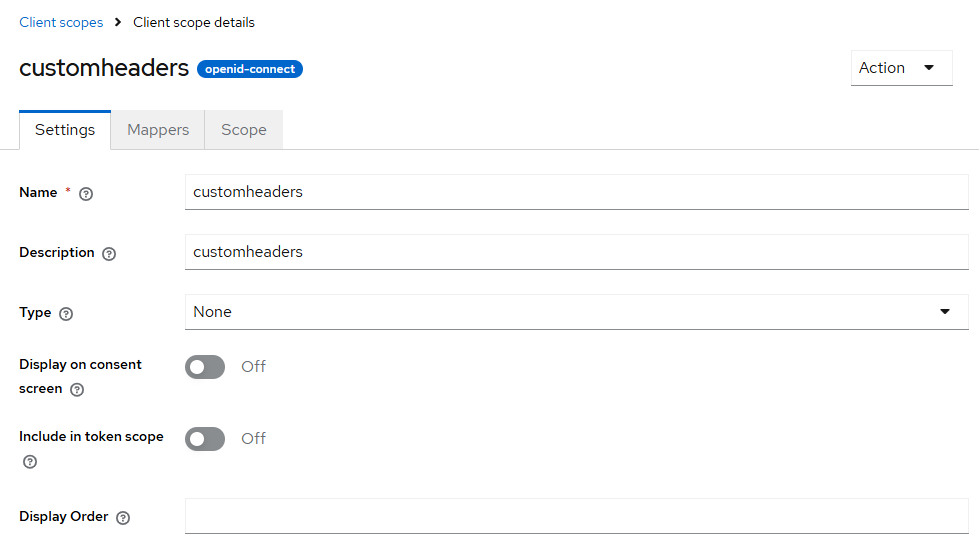


Fig. 1 : creation of client scope

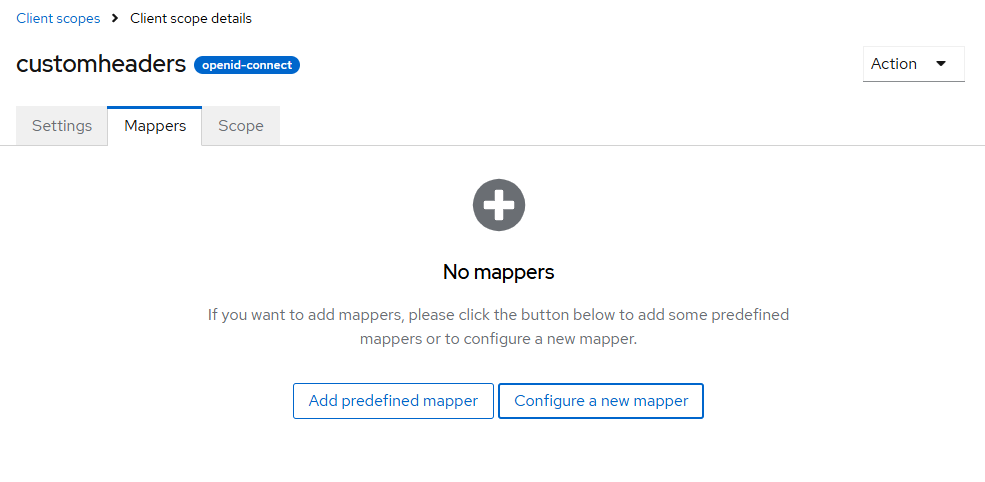


Fig. 2: Select configure a new mapper

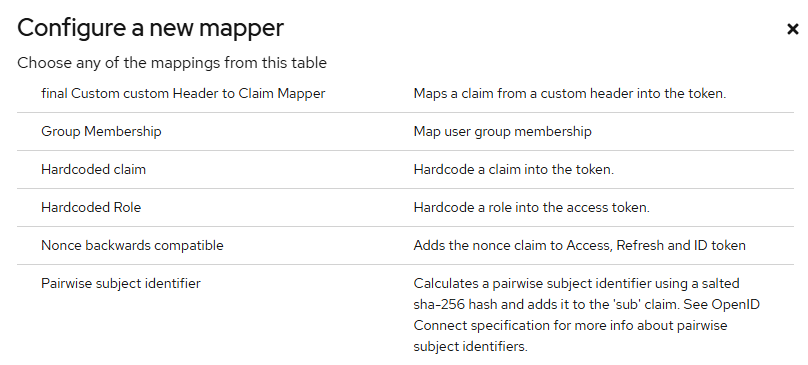


Fig 3. Selecting custom SPI

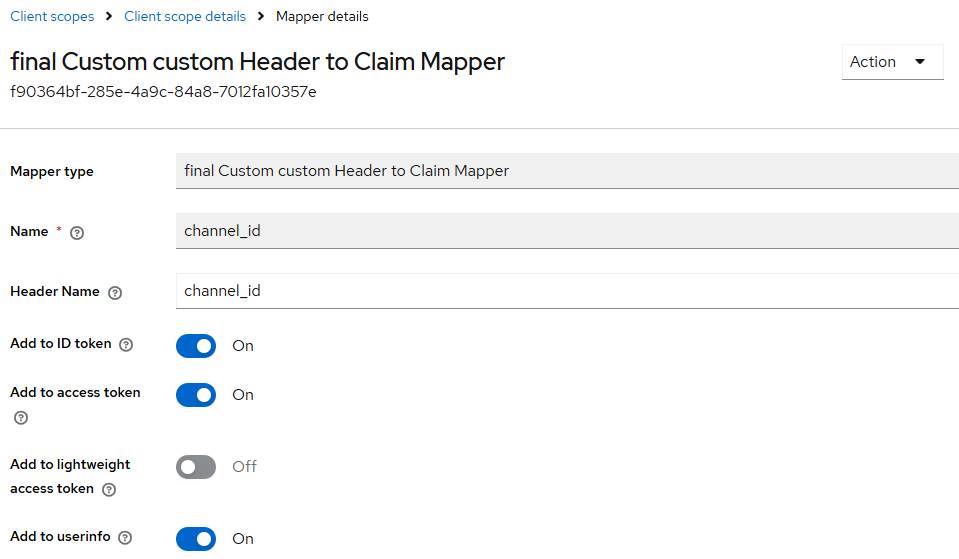


Fig. 4 : Setting parameter names

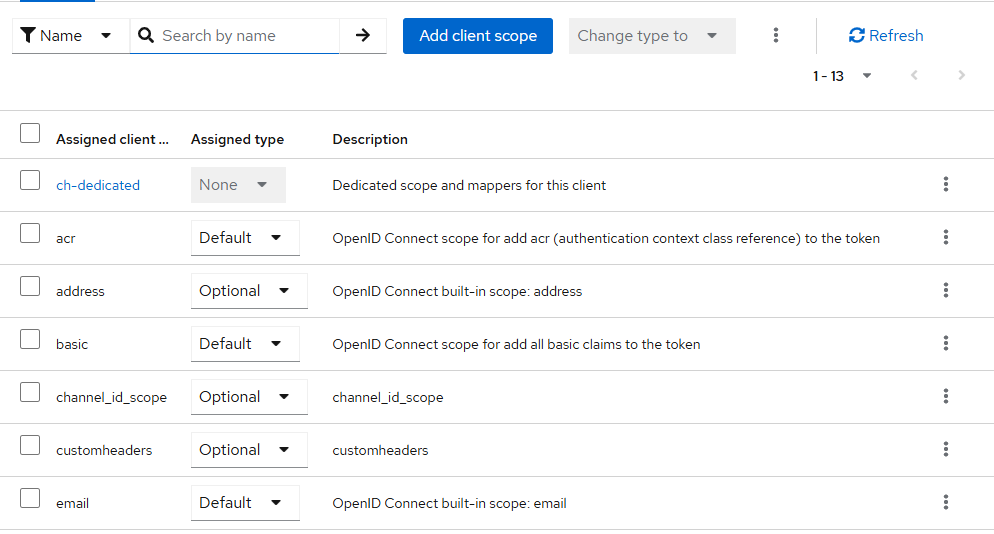
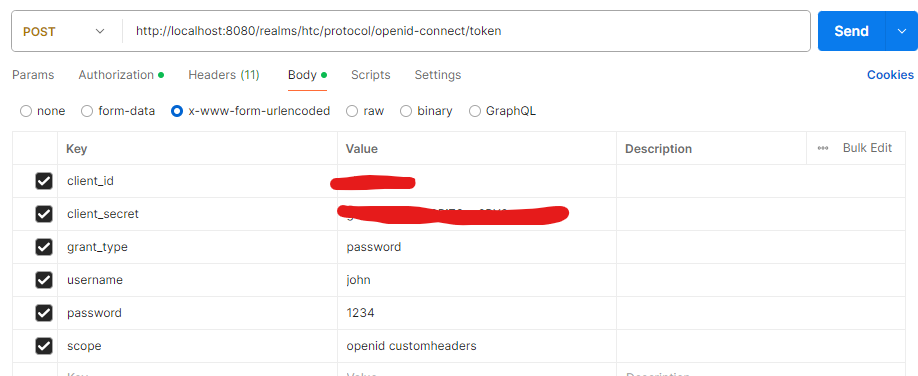


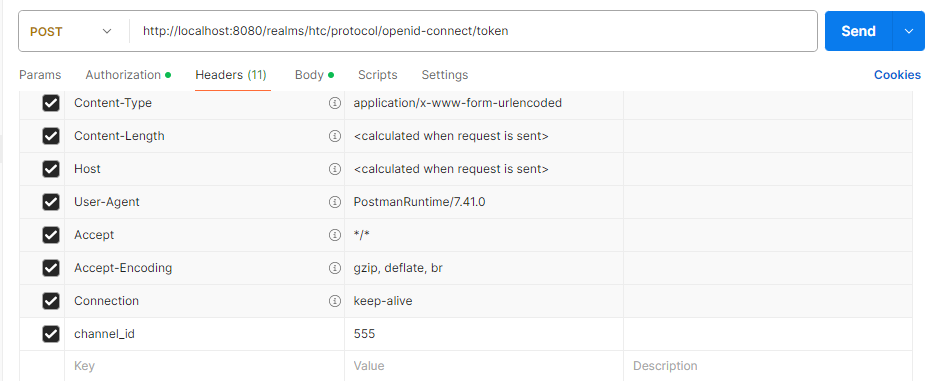
Fig. 5: Setting assigned type of client scope to optional

**User interaction:**

Step 1: Postman client configuration

Create a new http request in postman and configure as below . add the created scope name in scope and mapper names in headers section





**Flow in SPI**

1. Since we select our custom SPI as mapper, it implements methods that fetches the header parameters from protocol/openid/token point and its corresponding value.
2. We then implement another method to add the results obtained in previous step as claim in jwt