Ecwid Authenticate a Connector Instance

Last Modified on 03/30/2020 11:46 pm EDT

On this page

You can authenticate with Ecwid to create your own instance of the Ecwid connector through the UI or through APIs. Once authenticated, you can use the connector instance to access the different functionality offered by the Ecwid platform.

Authenticate Through the UI

Use the UI to authenticate with Ecwid and create a connector instance. If you authenticate with Ecwid via OAuth 2.0, all you need to do is add a name for the instance, whereas authenticating via custom authorization will require you to input your Ecwid Store ID, Oder API Secret, and Product API Secret you recorded in API Provider Setup. After you create the instance, you'll log in to Ecwid to authorize SAP Cloud Platform Open Connectors to access your account. For more information about authenticating a connector instance, see Authenticate a Connector Instance (UI).

After successfully authenticating, we give you several options for next steps. Make requests using the API docs associated with the instance, map the instance to a common resource, or use it in a formula template.

Authenticate Through API

To provision your Ecwid connector, use the /instances API.

Step 1. Call the /instances API

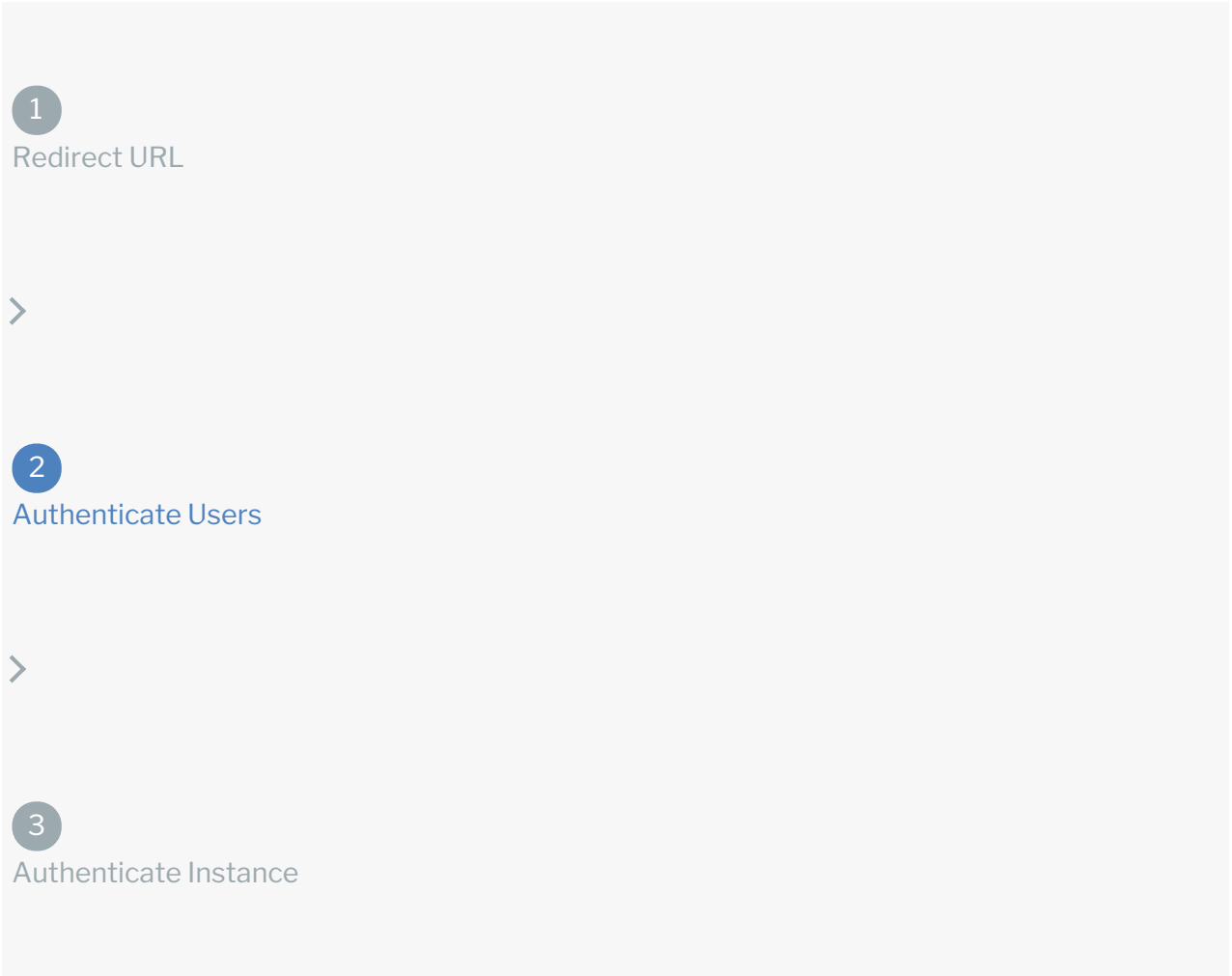
- **HTTP Headers**: Authorization- User , Organization
- **HTTP Verb**: POST
- **Request URL**: /instances
- **Request Body**: Required – see below
- **Query Parameters**: none

Ecwid now has two types of authentication- Oauth2 or custom.

Oauth2

```
curl -X GET \
'https://api.openconnectors.us2.ext.hana.ondemand.com/elements/api-v2/elements/oauth/url?apiKey=&apiSecret=&callbackUrl=' \
```

Authenticating Users and Receiving the Authorization Grant Code



Provide the `oauthUrl` in the response from the previous step to the users. After users authenticate, provides the following information in the response:

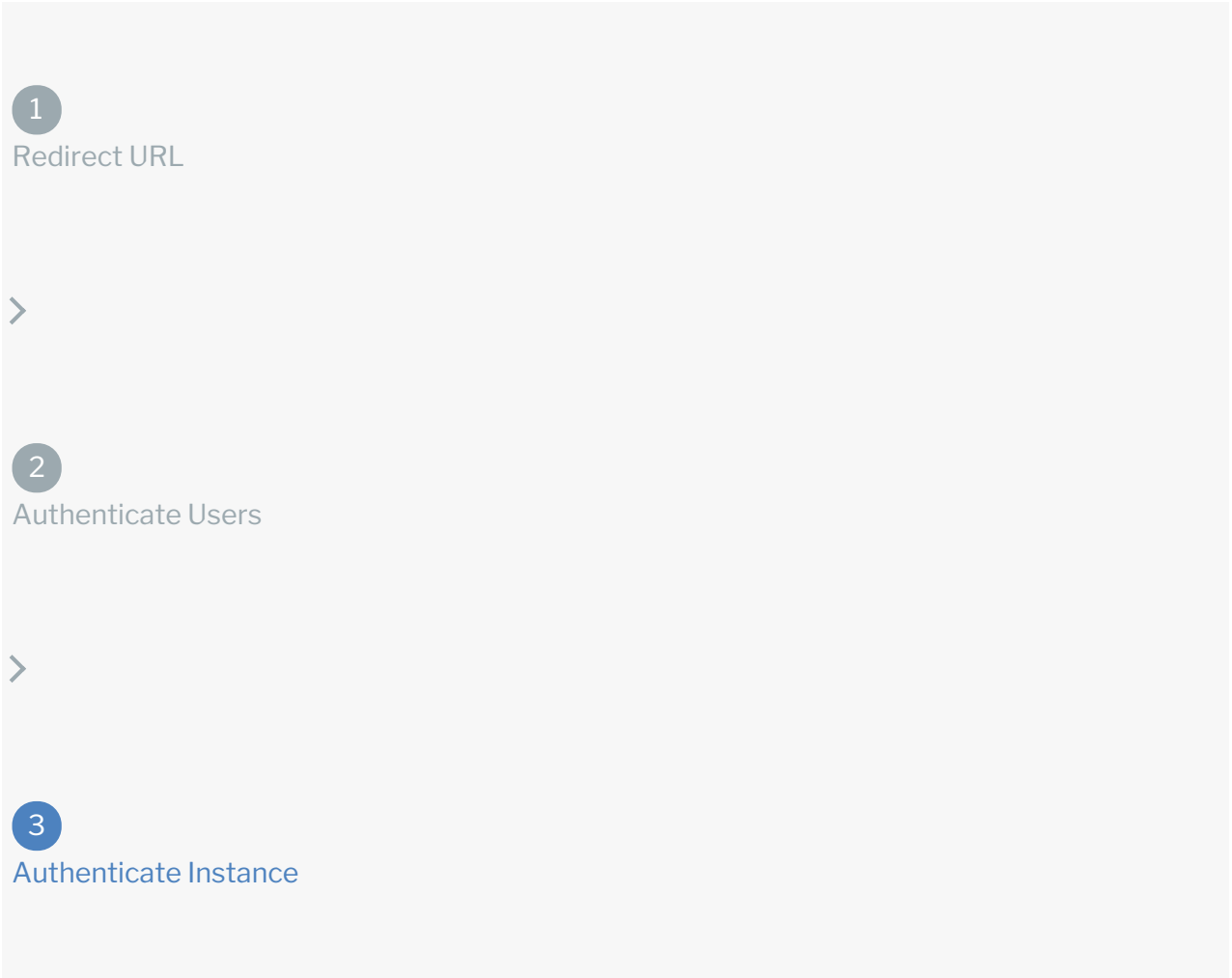
- code
- state

Response Parameter	Description
	The authorization grant code returned from the API provider in an OAuth 2.0

Response Parameter	Description
code	authentication workflow. SAP Cloud Platform Open Connectors uses the code to retrieve the OAuth access and refresh tokens from the endpoint.
state	A customizable identifier, typically the connector key (`) .

> **Note:** If the user denies authentication and/or authorization, there will be a query string parameter called `error` instead of the `code` parameter. In this case, your application can handle the error gracefully.

Authenticating the Connector Instance



Use the `code` from the previous step and the `/instances` endpoint to authenticate with and create a connector instance. If you are configuring events, see the Events section.

> **Note:** The endpoint returns a connector instance token and id upon successful completion. Retain the token and id for all subsequent requests involving this connector instance.

To authenticate a connector instance:

1. Construct a JSON body as shown below (see Parameters):

```
{
  "element": {
    "key": ""
  },
  "providerData": {
    "code": ""
  },
  "configuration": {
    "oauth.api.key": "< app >",
    "oauth.api.secret": "< app >",
    "oauth.callback.url": "< app  >"
  },
  "tags": [
    ""
  ],
  "name": ""
}
```

2. Call the following, including the JSON body you constructed in the previous step:

```
POST /instances
```

> **Note:** Make sure that you include the User and Organization keys in the header. See the Overview for details.

3. Locate the `token` and `id` in the response and save them for all future requests using the connector instance.

Example Request

```
curl -X POST \
  https://api.openconnectors.us2.ext.hana.ondemand.com/elements/api-v2/inst
ances \
  -H 'authorization: User , Organization ' \
  -H 'content-type: application/json' \
  -d '{
  "element": {
    "key": ""
  },
  "providerData": {
    "code": "xxxxxxxxxxxxxxxxxxxxxx"
  },
  "configuration": {
    "oauth.api.key": "Rand0MAP1-key",
    "oauth.api.secret": "fak3AP1-s3Cr3t",
    "oauth.callback.url": "https;//mycoolapp.com",
  },
  "tags": [
    "Docs"
  ],
  "name": "API Instance"
}'
```

Custom Authentication

In order to create an Ecwid instance with custom Authentication (note this is a legacy version of Ecwid so if you do not already have your store ID, order API Key, and Product API Key you should use oauth2), you will need the Store ID, Order API Key, and Product API Key. For instructions on how to retrieve those credentials, please see our Ecwid Endpoint Setup. NOTE: Ecwid currently supports the the GET, PUT/PATCH, DELETE API calls. POST is not available at this time.

Description: token is returned upon successful execution of this API. This token needs to be retained by the application for all subsequent requests involving this connector instance.

A sample request illustrating the /instances API is shown below.

HTTP Headers:

```
Authorization: User , Organization
```

This instance.json file must be included with your instance request. Please fill your information to provision. The “key” into SAP Cloud Platform Open Connectors Ecwid is “ecwid”. This will

need to be entered in the “key” field below depending on which Connector you wish to instantiate.

```
{
  "element": {
    "key": "ecwid"
  },
  "configuration": {
    "ecwid.order.key": "",
    "ecwid.product.key": "",
    "ecwid.store.id": ""
 },
  "name": ""
}
```

Here is an example cURL command to create an instance using /instances API.

Example Request:

```
curl -X POST
-H 'Authorization: User , Organization '
-H 'Content-Type: application/json'
-d @instance.json
'https://api.openconnectors.us2.ext.hana.ondemand.com/elements/api-v2/instances'
```

If the user does not specify a required config entry, an error will result notifying her of which entries she is missing.

Below is a successful JSON response:

```
{
  "id": 12345,
  "name": "test",
  "token": "dsPr6AheLIS8pt7rp8E81bSKEkx9Ftr+9Y",
  "element": {
    "id": 42,
    "name": "Ecwid",
    "key": "ecwid",
    "description": "Ecwid is everything you need to sell anywhere.",
    "image": "elements/provider_ecwid.png",
    "active": true,
    "deleted": false,
    "typeOauth": false,
    "trialAccount": false,
    "configDescription": "If you do not have a Ecwid account, you can creat
e one at Ecwid Signup"
  },
  "provisionInteractions": [],
  "valid": true,
  "disabled": false,
  "maxCacheSize": 0,
  "cacheTimeToLive": 0,
  "cachingEnabled": false
}
```


Note: Make sure you have straight quotes in your JSON files and cURL commands. Please use plain text formatting in your code.

Instance Configuration

The content in the `configuration` section or nested object in the body posted to the `POST /instances` or `PUT /instances/{id}` APIs varies depending on which connector is being instantiated. However, some configuration properties are common to all connectorss and available to be configured for all connectors. These properties are -

- `event.notification.enabled` : This property is a `boolean` property, and determines if event reception (via `webhook` or `polling`) is enabled for the connector instance. This property defaults to *false*.
- `event.vendor.type` : When `event.notification.enabled` property is set to *true*, this property determines the mechanism to use to receive or fetch changed events from the service endpoint. The supported values are `webhook` and `polling` . Most connectors support one mechanism or the other, but some like Salesforce.com support both mechanisms. This property is *optional*.
- `event.notification.type` : This property can be used to determine how an event

notification should be sent to the consumer of the connector instance, in most cases your application. Currently, `webhook` is the only supported value for this property. This means that when an event is received by the connector instance, it will get forwarded to the provided `event.notification.callback.url` via a `webhook` to you. This property is *optional*.

- `event.notification.callback.url` : As mentioned above, the value of this property is an `http` or `https` URL to which we will post the event for consumption by your application. This property is *optional*.
- `filter.response.nulls` : This property defaults to *true*, i.e., it's `boolean` property, and determines if `null` values in the response `JSON` should or should not be filtered from the response returned to the consuming application. By default, all `null` values are filtered from the response before sending the response to the consuming application.