Connectors TLS v1.2/Two-Way SSL

Last Modified on 03/30/2021 8:54 am EDT

This article describes the process for doing Two-Way SSL in Connector Builder. Two-Way SSL is also sometimes referred to as SSL, Mutual TLS/SSL, Two-Way TLS, or TLS v1.2.

Not sure what that is? If your API says you need to send a signed certificate for each request, there is a chance this article will help.

For this example, I will be building authentication for ADP WorkforceNow and making a GET request following these steps:

1. Create a Java KeyStore formatted certificate.
2. Configure the connector to accept our certificate, and finish authentication.
3. Create an instance.
4. Validate with a GET call.

At a more granular level:

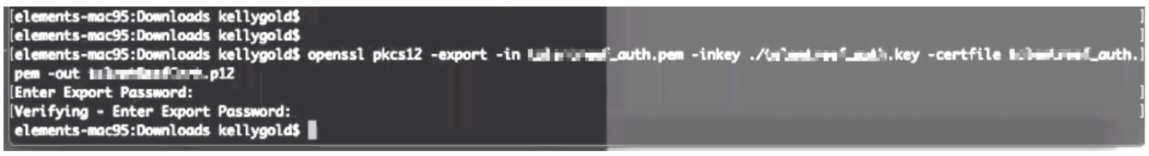
1. Gather files needed create a keystore. I have a `.pem` file and a `.key` file, but you may instead/also have a `.cer` file.
2. Convert the `.pem` file to a `.P12` (pks12) file
3. Create the Java KeyStore from the `.P12` (pks12) file.
4. Base64 encode the Java KeyStore (.jks) file.
5. Create an connector with custom authentication
6. Add a custom resource and set "API Type" to "On Provision"
7. Exchange the Basic Auth token with the Bearer token
8. make a GET Call.

Step 1: Create a Java KeyStore formatted certificate

You'll need a couple of files to create your certificate. Usually, this is a `.cer`, `.pem`, or `.key` file (or a combination thereof). For the ADP APIs, I have a `.pem` and a `.key` file

- In the directory with your files (`.cer`, `.pem`, `.key`), run the following command. Make sure to replace the file names with yours!

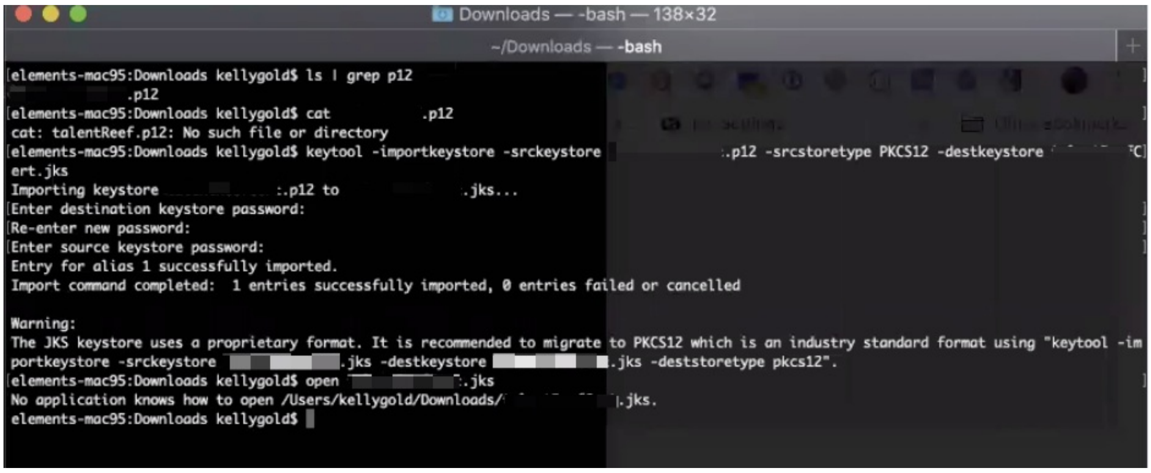
  -
    ```
    openssl pkcs12 -export -in MyPemFile.pem -inkey myKeyFile.key -certfile
    myPemFile.pem -out myPKS12OutputFile.p12
    ```
  - When prompted, create a password.



```
elements-mac95:Downloads kellygold$
elements-mac95:Downloads kellygold$
elements-mac95:Downloads kellygold$ openssl pkcs12 -export -in [redacted]_auth.pem -inkey ./[redacted].key -certfile [redacted]_auth.
pem -out [redacted].p12
Enter Export Password:
Verifying - Enter Export Password:
elements-mac95:Downloads kellygold$
```

- Convert the P12 (pks12) file to a `.jks` file (Java KeyStore formatted certificate)
- Run the following command to convert the KeyStore, enter passwords when prompted (save this). Make sure to enter your file names!

-
  ```
  keytool -importkeystore -srckeystore myP12File.p12 -srcstoretype PKCS12 -destkeystore myJKSCert.jks
  ```



```
elements-mac95:Downloads kellygold$ ls | grep p12
[redacted].p12
elements-mac95:Downloads kellygold$ cat [redacted].p12
cat: talentReef.p12: No such file or directory
elements-mac95:Downloads kellygold$ keytool -importkeystore -srckeystore [redacted].p12 -srcstoretype PKCS12 -destkeystore [redacted]C
ert.jks
Importing keystore [redacted].p12 to [redacted].jks...
Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
Entry for alias 1 successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standard format using "keytool -im
portkeystore -srckeystore [redacted].jks -destkeystore [redacted].jks -deststoretype pkcs12".
elements-mac95:Downloads kellygold$ open [redacted].jks
No application knows how to open /Users/kellygold/Downloads/[redacted].jks.
elements-mac95:Downloads kellygold$
```

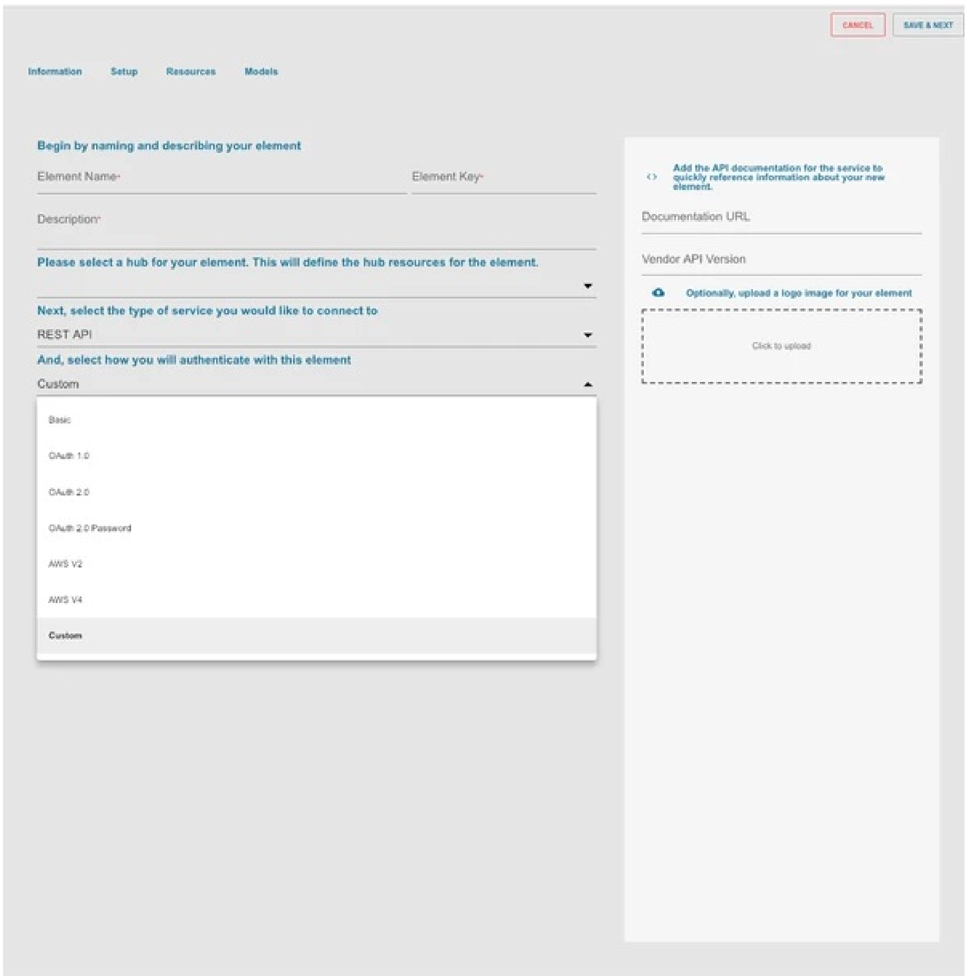
You should now have a `.jks` file in the same directory.

Step 2: Configure the connector to accept our certificate, and finish authentication

The ADP connector authentication is as follows

1. Make a POST to the /Token Endpoint (correct; do not authorize for ADP).
    1. Include a Basic HTTP Auth header with ClientID + Client Secret Base64 encoded with a colon between - ClientID:ClientSecret
    2. Include the certificate on the request.
2. Set the returned Access_Token body parameter as the Authorization Bearer Token for ongoing requests.

- Create a connector with custom authentication



CANCEL | SAVE & NEXT

Information | Setup | Resources | Models

Begin by naming and describing your element

Element Name*

Element Key*

Description*

Please select a hub for your element. This will define the hub resources for the element.

Next, select the type of service you would like to connect to

REST API

And, select how you will authenticate with this element

Custom

- Basic
- OAuth 1.0
- OAuth 2.0
- OAuth 2.0 Password
- AWS V2
- AWS V4
- **Custom**

Add the API documentation for the service to quickly reference information about your new element.

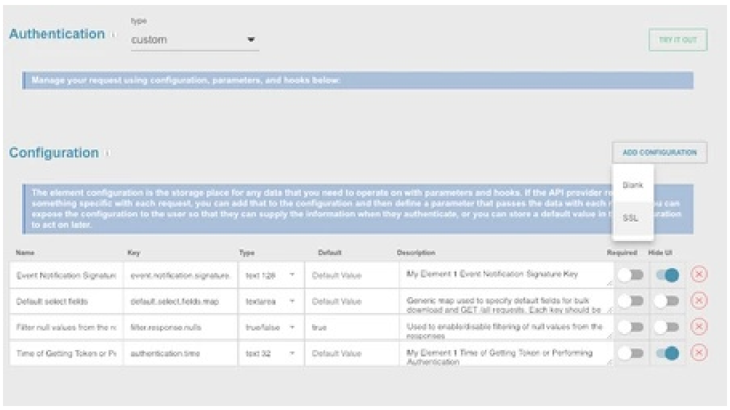
Documentation URL

Vendor API Version

Optionally, upload a logo image for your element

Click to upload

- In Setup>Configuration, select "ADD CONFIGURATION" and select "SSL". This should add two configuration items to our connector (`Base64.encoded.jks` and `jks.password`)



Authentication

type: custom

TRY IT OUT

Manage your request using configuration, parameters, and hooks below.

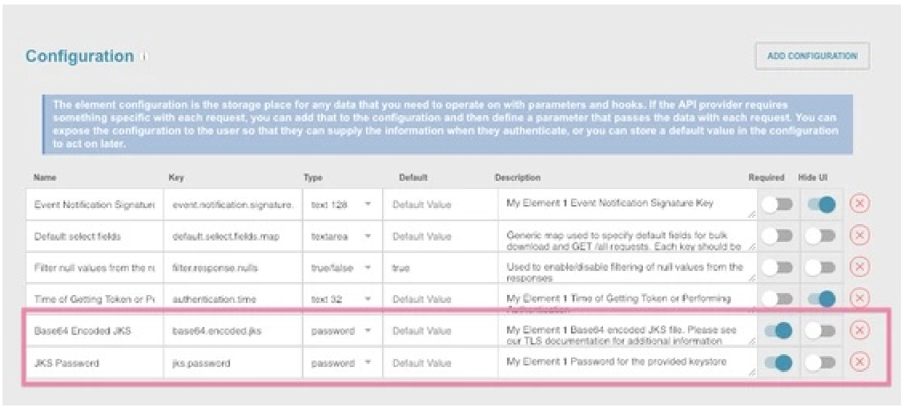
Configuration

ADD CONFIGURATION

- Blank
- SSL

The element configuration is the storage place for any data that you need to operate on with parameters and hooks. If the API provider r… something specific with each request, you can add that to the configuration and then define a parameter that passes the data with each r… …u can expose the configuration to the user so that they can supply the information when they authenticate, or you can store a default value in t… …ration to act on later.

Name	Key	Type	Default	Description	Required	Hide UI
Event Notification Signatur	event.notification.signature.	text 128	Default Value	My Element 1 Event Notification Signature Key	○	●
Default select fields	default.select.fields.map	textarea	Default Value	Generic map used to specify default fields for bulk download and GET /all requests. Each key should be	○	○
Filter null values from the r	filter.response.nulls	true/false	true	Used to enable/disable filtering of null values from the responses	○	○
Time of Getting Token or Pe	authentication.time	text 32	Default Value	My Element 1 Time of Getting Token or Performing Authentication	○	●



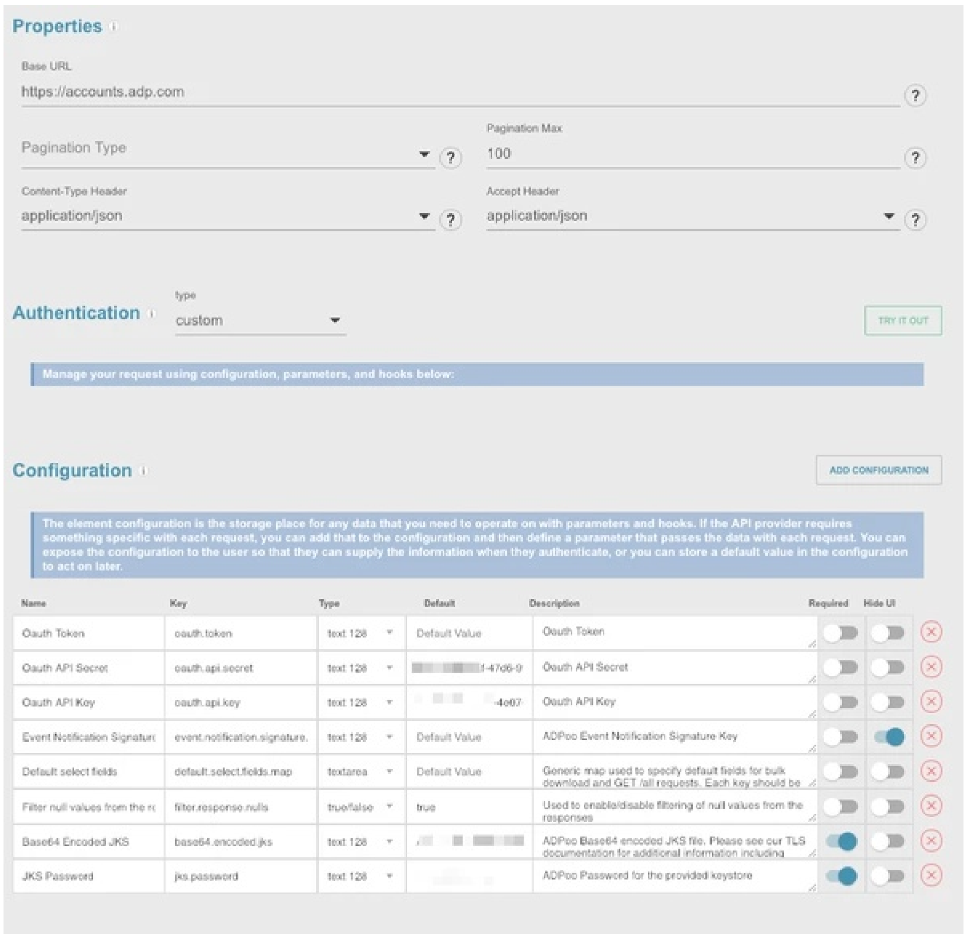
Configuration

ADD CONFIGURATION

The element configuration is the storage place for any data that you need to operate on with parameters and hooks. If the API provider requires something specific with each request, you can add that to the configuration and then define a parameter that passes the data with each request. You can expose the configuration to the user so that they can supply the information when they authenticate, or you can store a default value in the configuration to act on later.

Name	Key	Type	Default	Description	Required	Hide UI
Event Notification Signatur	event.notification.signature.	text 128	Default Value	My Element 1 Event Notification Signature Key	○	●
Default select fields	default.select.fields.map	textarea	Default Value	Generic map used to specify default fields for bulk download and GET /all requests. Each key should be	○	○
Filter null values from the r	filter.response.nulls	true/false	true	Used to enable/disable filtering of null values from the responses	○	○
Time of Getting Token or Pe	authentication.time	text 32	Default Value	My Element 1 Time of Getting Token or Performing Authentication	○	●
Base64 Encoded JKS	base64.encoded.jks	password	Default Value	My Element 1 Base64 encoded JKS file. Please see our TLS documentation for additional information	●	○
JKS Password	jks.password	password	Default Value	My Element 1 Password for the provided keystore	●	○

- Next, we'll need to add configuration for our ClientID, ClientSecret, and Access Token. You may choose to put default values for these for testing, although note that for a production-ready connector, they will need to be parameterized.



Properties

Base URL: https://accounts.adp.com

Pagination Type:

Pagination Max: 100

Content-Type Header: application/json

Accept Header: application/json

Authentication type: custom

TRY IT OUT

Manage your request using configuration, parameters, and hooks below:

Configuration

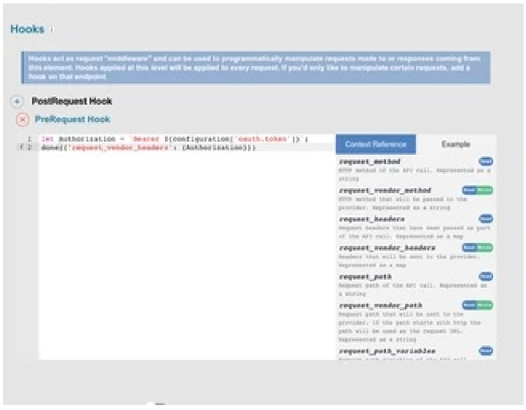
ADD CONFIGURATION

The element configuration is the storage place for any data that you need to operate on with parameters and hooks. If the API provider requires something specific with each request, you can add that to the configuration and then define a parameter that passes the data with each request. You can expose the configuration to the user so that they can supply the information when they authenticate, or you can store a default value in the configuration to act on later.

Name	Key	Type	Default	Description	Required	Hide UI
Oauth Token	oauth.token	text 128	Default Value	Oauth Token	○	○
Oauth API Secret	oauth.api.secret	text 128	[redacted]1-47d6-9	Oauth API Secret	○	○
Oauth API Key	oauth.api.key	text 128	[redacted]-4e07	Oauth API Key	○	○
Event Notification Signatur	event.notification.signature.	text 128	Default Value	ADPoo Event Notification Signature Key	○	●
Default select fields	default.select.fields.map	textarea	Default Value	Generic map used to specify default fields for bulk download and GET /all requests. Each key should be	○	○
Filter null values from the re	filter.response.nulls	true/false	true	Used to enable/disable filtering of null values from the responses	○	○
Base64 Encoded JKS	base64.encoded.jks	text 128	[redacted]	ADPoo Base64 encoded JKS file. Please see our TLS documentation for additional information including	●	○
JKS Password	jks.password	text 128	[redacted]	ADPoo Password for the provided keystore	●	○

  - Add a Global-Pre-Hook setting your Oauth Token to be sent in the Headers on a request

    -
      ```
      let Authorization = `Bearer ${configuration['oauth.token']}`;
      ```
      ```
      done({'request_vendor_headers': {Authorization}})
      ```



Hooks

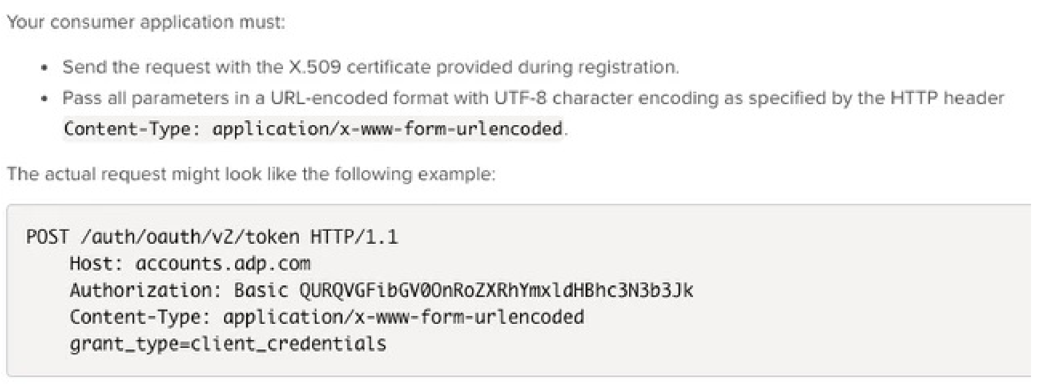
Hooks act as request "middleware" and can be used to programmatically manipulate requests made to or responses coming from this element. Hooks applied at this level will be applied to every request. If you'd only like to manipulate certain requests, add a hook on that endpoint.

PostRequest Hook

PreRequest Hook

```
let Authorization = `Bearer ${configuration['oauth.token']}`;
done({'request_vendor_headers': {Authorization}})
```

- Now that we've set up our configuration, we need to add an authentication resource and a bit of Javascript to make everything work.
- On Resources tab, Add a new resource (POST in ADP's case). This API should reflect the first call made to the vendor during authentication. Since we chose Custom, it's up to the developer to define how/what to call to retrieve the access token.
  - Set the "Resource Type" to "On Provision", which will call this API when we provision a connector instance.
  - Note the vendor requirements from the ADP API documentation:


Your consumer application must:

- Send the request with the X.509 certificate provided during registration.
- Pass all parameters in a URL-encoded format with UTF-8 character encoding as specified by the HTTP header `Content-Type: application/x-www-form-urlencoded`.

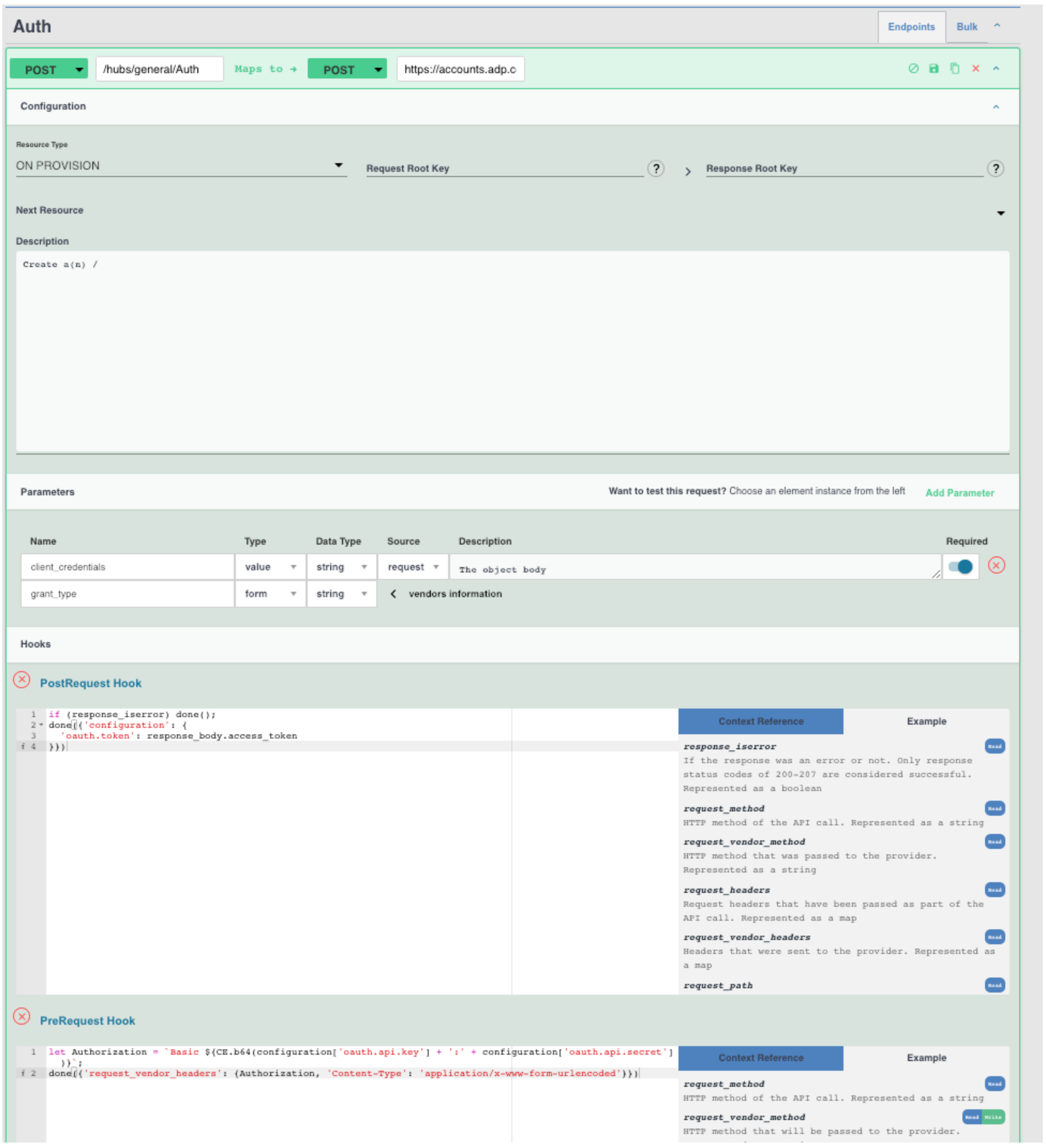
The actual request might look like the following example:

```
POST /auth/oauth/v2/token HTTP/1.1
    Host: accounts.adp.com
    Authorization: Basic QURQVGFibGV0OnRoZXRhYmxldHBhc3N3b3Jk
    Content-Type: application/x-www-form-urlencoded
    grant_type=client_credentials
```

- We still need to tell ADP our Grant-Type. To do this modify the Body "Parameter" on the connector resource we've created.
  - (See the finished product for example)
- Last, add a bit of Pre-Hook Javascript to 1. Set the Basic Auth header on the provisioning request (1st request), and 2. Set the oauth.api.key to the `access_token` returned in the response.

  -
    ```
    let Authorization = `Basic ${CE.b64(configuration['oauth.api.key'] + ':' + configuration['oauth.api.secret'])}`;
    done({'request_vendor_headers': {Authorization, 'Content-Type': 'application/x-www-form-urlencoded'}})
    ```
- Post-Hook Javascript:

  -
    ```
    if (response_iserror)
    done();
    done({'configuration':
    {
    'oauth.token':
    response_body.access_token
    }})
    ```
- `The Finished Auth Resource`



Auth

Endpoints | Bulk

POST /hubs/general/Auth Maps to → POST https://accounts.adp.c

Configuration

Resource Type: ON PROVISION

Request Root Key ? > Response Root Key ?

Next Resource

Description

```
Create a(n) /
```

Parameters

Want to test this request? Choose an element instance from the left | Add Parameter

Name	Type	Data Type	Source	Description	Required
client_credentials	value	string	request	The object body	✓
grant_type	form	string		vendors information	

Hooks

PostRequest Hook

```
if (response_iserror) done();
done({'configuration': {
  'oauth.token': response_body.access_token
}})
```

Context Reference | Example

```
response_iserror
If the response was an error or not. Only response
status codes of 200-207 are considered successful.
Represented as a boolean

request_method
HTTP method of the API call. Represented as a string

request_vendor_method
HTTP method that was passed to the provider.
Represented as a string

request_headers
Request headers that have been passed as part of the
API call. Represented as a map

request_vendor_headers
Headers that were sent to the provider. Represented as
a map

request_path
```

PreRequest Hook

```
let Authorization = `Basic ${CE.b64(configuration['oauth.api.key'] + ':' + configuration['oauth.api.secret']
  )}`;
done({'request_vendor_headers': {Authorization, 'Content-Type': 'application/x-www-form-urlencoded'}})
```

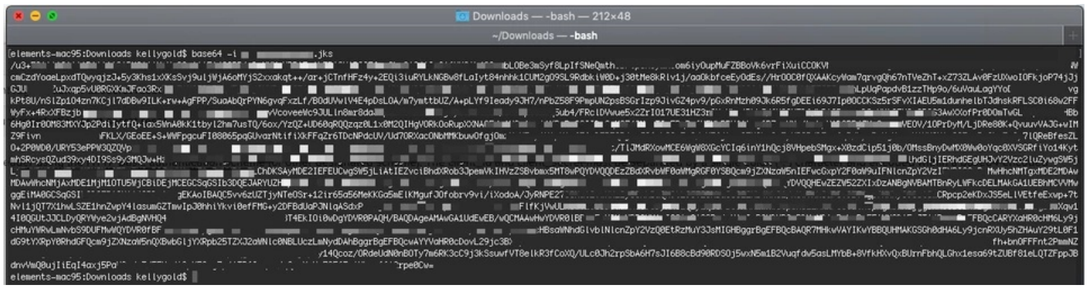
Context Reference | Example

```
request_method
HTTP method of the API call. Represented as a string

request_vendor_method
HTTP method that will be passed to the provider.
```

Step 3: Create an instance

- To create an instance (in this case, for ADP), enter the following:
  - Base64 JKS (java KeyStore) file
    - To base64 encode your keystore file, navigate to the directory and run:
      `base64 -`
    - `i myJKSCert.jks`

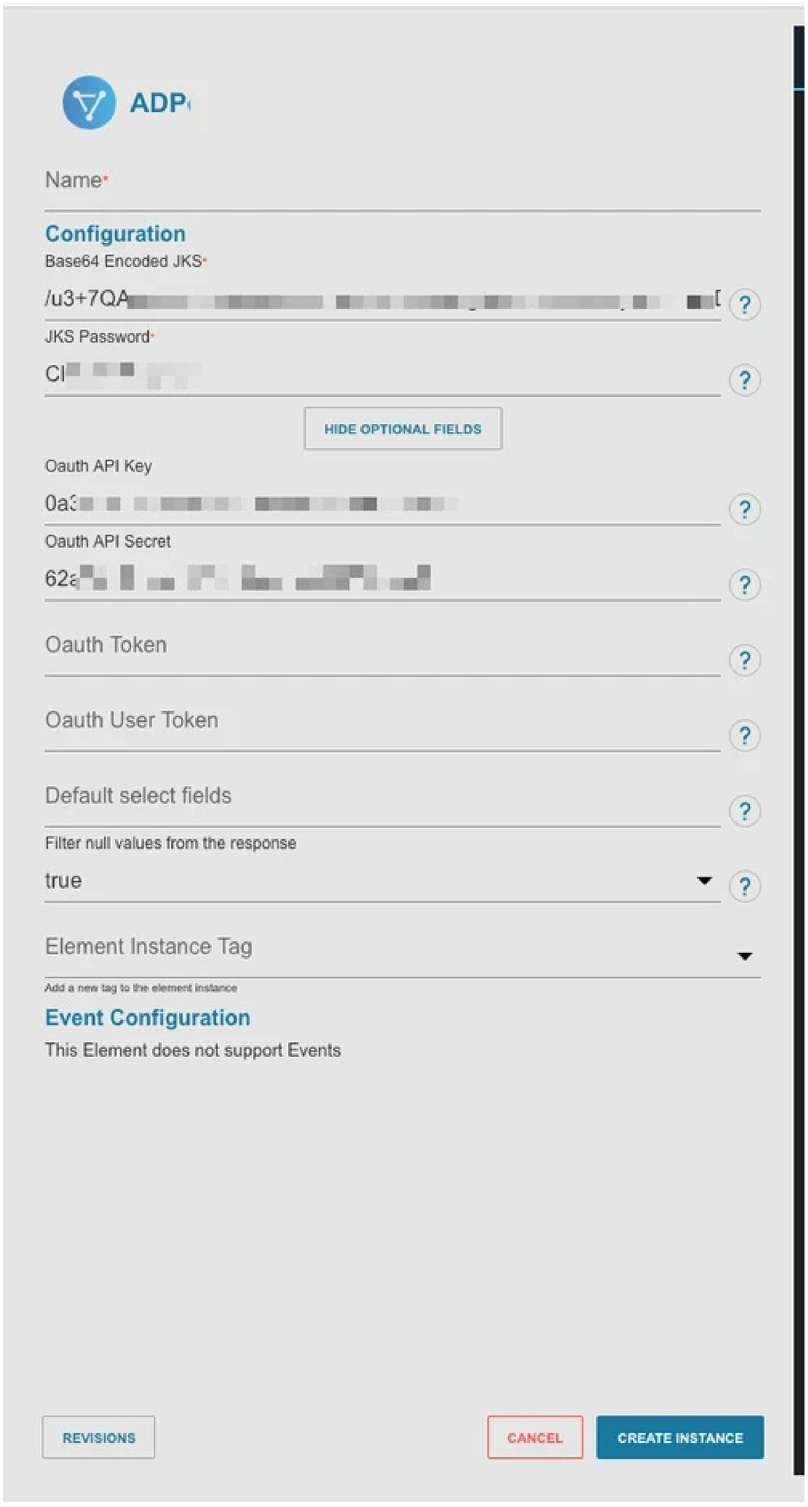


Downloads — -bash — 212×48

~/Downloads — -bash

```
elements-mac95:Downloads kellygold$ base64 -i [redacted].jks
[redacted]
elements-mac95:Downloads kellygold$
```

    - JKS Password set when you created the KeyStore (jks) file
  - ClientID
  - ClientSecret



Name*

Configuration

Base64 Encoded JKS*

/u3+7QA[redacted]

JKS Password*

Cl[redacted]

Oauth API Key

0a[redacted]

Oauth API Secret

62[redacted]

Oauth Token

Oauth User Token

Default select fields

Filter null values from the response

true

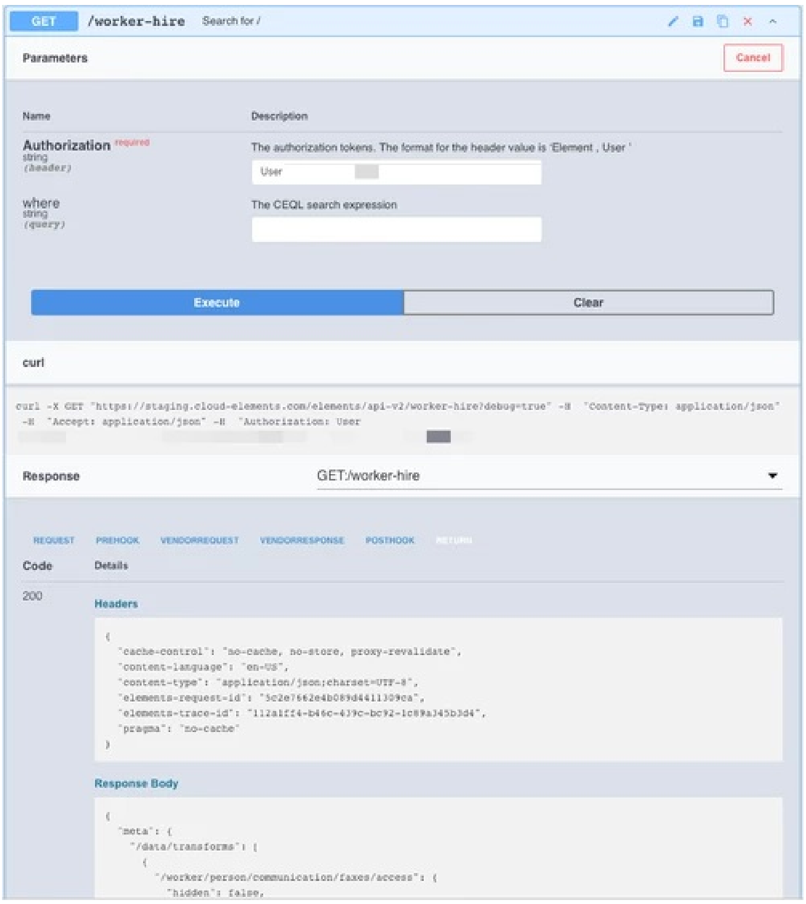
Element Instance Tag

Add a new tag to the element instance

Event Configuration

This Element does not support Events

Step 4: Validate result by making a GET call



Authorization required
string
(header)

The authorization tokens. The format for the header value is 'Element , User '

User [redacted]

where
string
(query)

The CEQL search expression

curl

```
curl -X GET "https://staging.cloud-elements.com/elements/api-v2/worker-hire?debug=true" -H  "Content-Type: application/json"
 -H  "Accept: application/json" -H  "Authorization: User
[redacted]
```

Response GET:/worker-hire

Code Details

200

Headers

```
{
  "cache-control": "no-cache, no-store, proxy-revalidate",
  "content-language": "en-US",
  "content-type": "application/json;charset=UTF-8",
  "elements-request-id": "5c2e7662e4b089d4411309ca",
  "elements-trace-id": "112a1ff4-b46c-439c-bc92-1c89a345b3d4",
  "pragma": "no-cache"
}
```

Response Body

```
{
  "meta": {
    "/data/transforms": [
      {
        "/worker/person/communication/faxes/access": {
          "hidden": false,
```