

Mapping and Transforming Fields

Last Modified on 08/30/2021 1:01 pm EDT

Transformations refer to the conversion of the fields in an API provider's resource to match the fields in a common resource. Transformations are the result of the process of creating a common resource, and then mapping fields in an connector instance resources to the common resource. See [Creating Common Resources](#) for the steps to set up resources for transformation.

Map Fields to a Common Resource

Before you can transform fields, you need to map the fields for each connector instance to a common resource. The common resource fields are on the left and the connector instance resource fields are on the right.

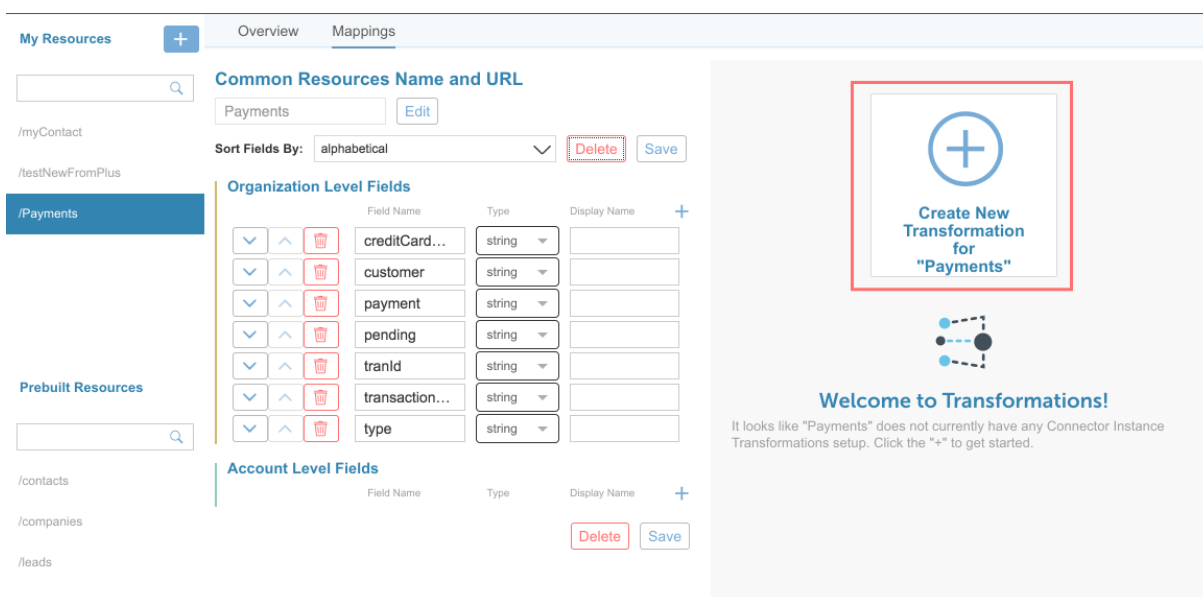
Note: We provide a default `id` field, which you can choose to map to an `id` field in the connector's resource, delete, or rename to an entirely different field. If you cloned your resource, you will see more fields than just the `id` field.

You can map fields one at a time, or you can add several fields to the common resource at once, and then map them later. These instructions describe mapping a single field at a time.

This section describes mapping to a common resource at the account level using the UI. You must be an account level user to map to account level fields. Go here for [instructions about using the APIs](#).

To map fields:

1. Navigate to the Transformations page.
2. On the Transformations page, click **Create New Transformation**.



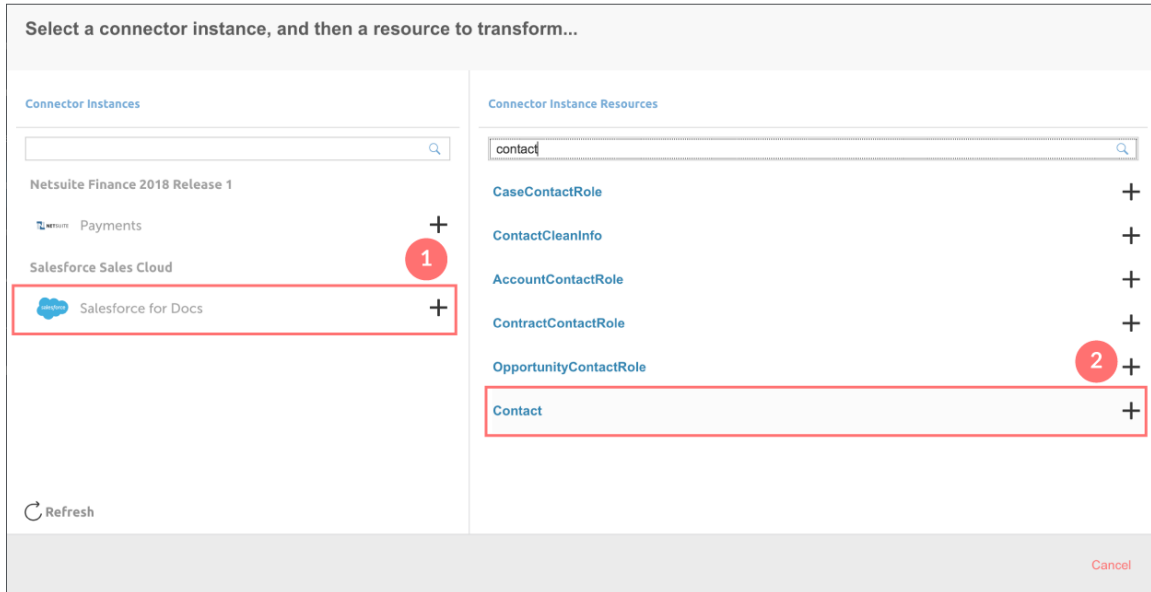
The screenshot shows the 'Mappings' page in a web application. The page is divided into several sections:

- My Resources:** A search bar and a list of resources including `/myContact`, `/testNewFromPlus`, and `/Payments` (which is highlighted).
- Common Resources Name and URL:** A search bar containing 'Payments' and an 'Edit' button.
- Sort Fields By:** A dropdown menu set to 'alphabetical', with 'Delete' and 'Save' buttons.
- Organization Level Fields:** A table with columns for Field Name, Type, and Display Name. The fields listed are: creditCard..., customer, payment, pending, tranId, transaction..., and type. Each row has a trash icon and a plus icon.
- Account Level Fields:** A table with columns for Field Name, Type, and Display Name. There are 'Delete' and 'Save' buttons.
- Prebuilt Resources:** A search bar and a list of resources including `/contacts`, `/companies`, and `/leads`.
- Overlay:** A large overlay on the right side of the screen with a red border. It contains a blue plus sign in a circle, the text 'Create New Transformation for "Payments"', a diagram of a connector instance, and the text 'Welcome to Transformations!' followed by a message: 'It looks like "Payments" does not currently have any Connector Instance Transformations setup. Click the "+" to get started.'

3. Select the Connector Instance, and then select the Connector Instance Resource.

The Resources available to that Connector Instance appear in the Connector Instance Resources column after you

select a resource.



Tip: Use the search fields to find what you are looking for in long lists.

4. Beginning with the default field **id**, select a field on the right to map to **id**.

Note: If you do not see the fields that you expect, switch on **Load metadata by id**, and use the id of an actual object in the resource. We'll then use the metadata associated with that object to populate the list of fields. See [Transforming Custom Objects](#) for details.

5. Click **+** next to the Account Level Fields to add another field.

Note: You can add fields at the instance level as well, but these steps focus on creating an account level common resource.

6. Enter a name for the field, and then choose the data type if the field is something other than a string.
7. Optionally add a **Display Name** to more clearly identify a field with how it appears in an API provider's UI.
8. Select the corresponding field on the right to map to the new field.

Note: You can type in the field to filter.


9. Continue adding resources until you finish, and then click **Save**.
10. To map the resource to another instance, click **Transformations** in the breadcrumbs at the top of the page.

Tips

- If a common resource has account-level fields overridden by instance-level fields, the instance-level overrides will appear in the Account Level Fields list, although they apply only to the specific instance they're mapped to.

After you add an instance level field and click Save, the new instance level field will also appear in the above Account Level Fields list, mirroring what you just added in Instance Level Fields. The new instance level field's place in the Account Level Fields section indicates that it is applied in the particular instance, as the instance level overrides a corresponding account-level field.

- If you made a mistake and don't want to include a field in a common resource, click . If you still want the field but want to remove the mapping, click .

- If you need to map a custom field, click  , and then type a name. See [Transforming Custom Objects](#) for details.
- We use dot notation to show sub-objects in the connector instance resources. If you need to create sub-objects in your common resource, use dot notation. Examples include address.city, address.state, and address.street. See [Working With Nested Objects](#) for details.
- Use Javascript to Manage Complex Objects. See [Javascript in Transformations](#) for details.

Map and Transform Fields with the APIs

Map Fields to Create a Default Transformation

You can use several APIs to map resources depending on the level at which you want to map them. This section describes mapping fields for an account-level default transformation. The result is a default transformation for all instances of a specific connector.

To map fields:

1. Construct a JSON body as shown below. For descriptions of each parameter, see [Transformation JSON Parameters](#).

```
{
  "level": "account",
  "vendorName": "",
  "fields": [
    {
      "path": "",
      "type": "",
      "vendorPath": "",
      "vendorType": ""
    }
  ]
}
```


2. Call the following, including the JSON body from the previous step:

```
POST /accounts/elements/{keyOrId}/transformations/{objectName}
```

Note: Replace {keyOrId} with the connector key or id and replace {objectName} with the name of the common resource.

Transformation JSON Parameters

Parameter	Description	Required (Y/N)
level	The access level of the transformation, either <code>account</code> or <code>instance</code> .	N. Default depends on endpoint.
vendorName	The name of the resource that contains the fields that you want to map to the common resource.	N
fields	An object containing the field names and data types of the common resource and the vendor resource.	N

 **Tip:** To get a list of fields in a resource, call `GET hubs/{hub}/objects/{RESOURCE}/metadata` .

path Parameter	Description	Y Required (Y/N)
vendorPath	The name of the field in the vendor resource.	Y
type	The data type of the field in the common resource. Data types can be <code>boolean</code> , <code>string</code> , <code>date</code> , and <code>number</code> .	N
vendorType	The data type of the field at the vendor. Unless the format is <code>date</code> , you do not need to include this parameter. If the format is <code>date</code> , also include a mask.	N

cURL Example

```
curl -X POST \
  https://api.openconnectors.us2.ext.hana.ondemand.com/elements/api-v2/accounts/elements/sfdc/trans
  formations/myContacts \
  -H 'authorization: User sAfK7LJGNz5ZHcNrvdJvLI=f03WbTbH6aRKc0HJ3oOIi, Organization 58168435e3b9
  959a929eb04b6218b9a2' \
  -H 'content-type: application/json' \
  -d '
  {
    "level": "account",
    "vendorName": "Contact",
    "fields": [
      {
        "type": "string",
        "path": "FirstName",
        "vendorPath": "FirstName"
      },
      {
        "type": "string",
        "path": "id",
        "vendorPath": "Id"
      },
      {
        "type": "string",
        "path": "LastName",
        "vendorPath": "LastName"
      },
      {
        "type": "date",
        "path": "birthdate",
        "vendorPath": "Birthdate",
        "vendorType": "date",
        "configuration": [
          ]
        ]
      }
    ]
  }'
```

Map Fields at the Instance Level

Using the `/instances` endpoint, you can map fields at the instance level. Mapping fields is a two-step process that includes creating an instance level resource, and then mapping fields to it.

To create an instance level common resource and map fields to it:

1. Construct a JSON body for the instance level resource as shown below (see [New Common Resource JSON Parameters](#)):

```
{
  "fields": [
    {
      "type": "",
      "path": ""
    }
  ]
}
```

2. Create the common resource. Make the following API call with the JSON body from the previous step, replacing `{id}` with the instance id, and replacing `{objectName}` with the name of the common resource:

```
POST /instances/{id}/objects/{objectName}/definitions
```

3. Construct a JSON body to map fields to the new common resource as shown below. For descriptions of each parameter, see [Transformation JSON Parameters](#).

```
{
  "level": "instance",
  "vendorName": "",
  "fields": [
    {
      "path": "",
      "type": "",
      "vendorPath": "",
      "vendorType": ""
    }
  ]
}
```

4. Map fields to the common resource. Call the following, including the JSON body from the previous step:

```
POST /instances/{id}/transformations/{objectName}
```

Note: Replace `{id}` with the instance id and replace `{objectName}` with the name of the common resource.

cURL Example

Step 1: Create the common resource

```
curl -X POST \
  https://api.openconnectors.us2.ext.hana.ondemand.com/elements/api-v2/instances/{id}/objects/{objectName}/definitions \
  -H 'authorization: User sAfK7LJGNz5ZHcNrvdJvLI=f03WbTbH6aRKc0HJ3oOIi, Organization 58168435e3b9959a929eb04b6218b9a2' \
  -H 'content-type: application/json' \
  -d '{
    "fields": [
      {
        "type": "string",
        "path": "title"
      }
    ]
  }'
```

Step 2: Map fields to the common resource

```
curl -X POST \
  https://api.openconnectors.us2.ext.hana.ondemand.com/elements/api-v2/instances/{id}/transformations/{objectName} \
  -H 'authorization: User sAfK7LJGNz5ZHcNrvdJvLI=f03WbTbH6aRKc0HJ3oOIi, Organization 58168435e3b9959a929eb04b6218b9a2' \
  -H 'content-type: application/json' \
  -d '{
    {
      "vendorName": "Contact",
      "level": "instance",
      "fields": [
        {
          "path": "title",
          "type": "string",
          "vendorPath": "Title"
        }
      ]
    }
  }'
```

Map Fields at the Account Level

Using the `/accounts` endpoint, you can map fields at the account level. Mapping fields is a two-step process that includes creating an account level resource, and then mapping fields to it.

To create an account level common resource and map fields to it:

1. Construct a JSON body for the account level common resource as shown below (see [New Common Resource JSON Parameters](#)):

```
{
  "fields": [
    {
      "type": "",
      "path": ""
    }
  ]
}
```

2. Create the common resource. Make one of the following API calls with the JSON body from the previous step, replacing `{id}` with the account id, and replacing `{objectName}` with the name of the common resource:

```
POST /accounts/objects/{objectName}/definitions
```

Note: Use this API call to create a common resource at the default account level.

```
POST /accounts/{id}/objects/{objectName}/definitions
```

Note: Use this API call to specify an account by id.

3. Construct a JSON body to map fields to the new common resource as shown below. For descriptions of each parameter, see [Transformation JSON Parameters](#).

```
{
  "level": "instance",
  "vendorName": "",
  "fields": [
    {
      "path": "",
      "type": "",
      "vendorPath": "",
      "vendorType": ""
    }
  ]
}
```

4. Map fields to the common resource. Call the following, including the JSON body from the previous step:

```
POST /accounts/{id}/elements/{keyOrId}/transformations/{objectName}
```

Note: Replace `{id}` with the instance id, replace `{keyOrId}` with the connector key or id, and replace `{objectName}` with the name of the common resource.

cURL Example

Step 1: Create the common resource (default account)

```
curl -X POST \
  https://api.openconnectors.us2.ext.hana.ondemand.com/elements/api-v2/accounts/objects/{objectName}/definitions \
  -H 'authorization: User sAfK7LJGNz5ZHcNrvdJvLI=f03WbTbH6aRKc0HJ3oOIi, Organization 58168435e3b9959a929eb04b6218b9a2' \
  -H 'content-type: application/json' \
  -d '{
  "fields": [
    {
      "type": "string",
      "path": "title"
    }
  ]
}'
```

Step 1: Create the common resource (specific account)

```
curl -X POST \
  https://api.openconnectors.us2.ext.hana.ondemand.com/elements/api-v2/api-v2/accounts/{id}/objects/{objectName}/definitions \
  -H 'authorization: User sAfK7LJGNz5ZHcNrvdJvLI=f03WbTbH6aRKc0HJ3oOIi, Organization 58168435e3b9959a929eb04b6218b9a2' \
  -H 'content-type: application/json' \
  -d '{
  "fields": [
    {
      "type": "string",
      "path": "title"
    }
  ]
}'
```

Step 2: Map fields to the common resource

```
curl -X POST \
  https://api.openconnectors.us2.ext.hana.ondemand.com/elements/api-v2/accounts/{id}/elements/{keyOrIdtransformations}/{objectName} \
  -H 'authorization: User sAfK7LJGNz5ZHcNrvdJvLI=f03WbTbH6aRKc0HJ3o0Ii, Organization 58168435e3b9959a929eb04b6218b9a2' \
  -H 'content-type: application/json' \
  -d '{
    "vendorName": "Contact",
    "level": "account",
    "fields": [
      {
        "path": "title",
        "type": "string",
        "vendorPath": "Title"
      }
    ]
  }'
```

Javascript in Transformations

You can use custom Javascript when the basic object mapping does not meet your needs. For example, you might need to break a single address object into its component parts (address.city, address.state, address.street, and address.zip).

Note:

- For all scripts, Javascript `strict` mode is enforced.
- ES6 is supported.
- The function parameters are immutable, meaning they cannot be assigned to directly. To change an object or value passed into the function, first copy it to your own local variable, and then make the necessary changes.

To access the custom Javascript functionality:

- Click  .

functions include the parameters and functions in the following table:

Common Resource Custom JS Parameters and Functions

Parameter	Description
transformedObject	The transformed object, with any mappings already taking place.
originalObject	The original object, with no transformations or mappings taking place on it.
fromVendor	Is the transformation being executed coming back from the vendor (on an API response) ?
done	The callback function needed to call at the end of your JS. Call <code>done</code> to terminate a given step.

Libraries

- CE: Our custom library that provides some common functionality. It is not necessary to `require` this library, it is available by default.
 - `CE.randomString()` : Generate a random string (approx. 10 characters long).
 - `CE.randomEmail()` : Generate a random email address.

- `CE.md5(str)` : Create an MD5 hash from a string value. Takes a `string` as a parameter. Returns a `string` .
- `CE.b64(str)` : Encode a string in base64. Takes a `string` as a parameter. Returns a `string` .
- `CE.decode64(str)` : Decode a string from base64, using UTF-8 encoding. Takes a `string` as a parameter. Returns a `string` .
- `CE.hmac(algo)(enc)(secret, str)` : HMAC hash a string (*str*) using the provided secret (*secret*), algorithm (*algo*), and encoding (*enc*). See https://nodejs.org/api/crypto.html#crypto_class_hmac for more information about the algorithm and encoding parameters.
- `CE.hmac[algo][enc](secret, str)` : This is a set of convenience functions that allow HMAC hashing using some common algorithms and encodings. For example, `CE.hmacSha1Hex(secret, str)` will create an HMAC SHA1 hash of the provided string, using the provided secret, and return a hex string. You can replace *algo* and *enc* with the following values: *algo*: `Sha1` , `Sha256` , `Md5` *enc*: `Hex` , `base64`
- Lodash: The popular `lodash` library. To use this library, simply `require` it in your script. It is possible to use the library modules, as well, such as `lodash/fp` .
- Util: The standard Node `util` library. To use, `require` it in your script.

Note: While `Node global URL Origin` is supported, the module is not maintained by the platform.

Examples

- Adding fields to a resource when a certain endpoint does not provide them:

```
function (originalObject, transformedObject, fromVendor, done) {
  transformedObject.isCreatedThisYear = (fromVendor && transformedObject.createdDt > '2016-01-01');
  done(transformedObject);
}
```

- Two endpoints identify priority differently: one users numbers (1 or 2) and the other descriptions (low or high).

```
function (originalObject, transformedObject, fromVendor, done) {
  if (!fromVendor) done(transformedObject); // only care when returning data from the vendor

  transformedObject.priority = transformedObject.priorityNumber === 1 ? 'low' : 'high'; // we prefer our priority to be the string representation, so we convert the endpoints "priorityNumber" field to the appropriate string representation here.

  done(transformedObject);
}
```

- Combining `FirstName` and `LastName` fields.

```
function (originalObject, transformedObject, fromVendor, done) {
  if transformedObject.Name = originalObject.FirstName + ' ' + originalObject.LastName;
  done(transformedObject);
}
```

Transforming Custom Objects


If you do not see an object that you expect in the instance resources you can either use an id of an actual object to load its metadata or manually enter the object name. This sometime happens for custom objects you created at the endpoint.

To load object metadata:

1. Switch on **Load metadata by id**.
2. Enter an actual id associated with an object in the resource.
3. Click **Load**.

We'll use the metadata associated with that object to populate the list of fields.

To manually map a custom object:

1. Click  next to the field.

The list becomes a text entry field.

2. Enter the name of the object.

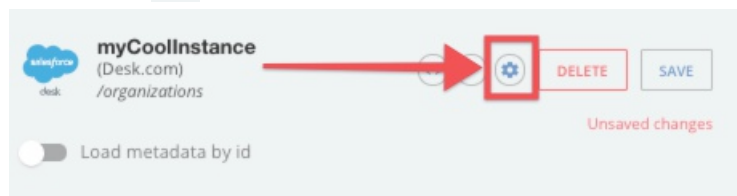


Removing Fields During Transformation

We pass through all fields in the JSON on both requests and responses. However, you can choose to remove all unmapped fields or specific fields from requests or responses.

To remove unmapped fields:

1. On the Transformations page, click  next to your connector instance.



2. Switch **Remove Unmapped Fields** to on.
3. Click **Save**.

To remove fields from requests or responses:

1. On the Transformations page, click  next to the field.



	Field Name	Type	Display Name	+
  	address.city	string	City	city
  	address.country	string	Country	country
  	address.geocodeA	string	Address Geocodin	geocodeAccuracy
  	address.latitude	number	Latitude	latitude

2. Switch on or off the sliders for the requests or responses.

For example, in the following configuration, we remove the data.id field from the response.

Field Settings for id <==> Id

Remove Field on Requests

Should we remove this field if it exists in the JSON on a request? By default, we pass them through.

Remove Field on Responses

Should we remove this field if it exists in the JSON on a response? By default, we pass them through.

Default Request Value

Set a static default value in case no value exists for this field in the request

Default Response Value

Set a static default value in case no value exists for this field in the response


Cancel

Save

Transforming Data Types

You can transform the data types on vendor objects. In most cases, you only need to select a new data type, but for dates you also provide a mask, or date format.

To change data types:

1. On the Transformations page, click  next to the field.
2. Select a type from the list.
3. If you select date, add a date format to the **Date Mask**.

Field Settings for date <==> CreatedDate

Date Mask

Specify a date mask to transform this date field value to the specified mask. Leaving it blank will return the date value as is.

Setting Default Values

If no values exist for a specific field, but you do not want to remove it, you can set a default value on both the response and request.

To set a default value:


1. On the Transformations page, click  next to the field.


2. Click **Default Request Value** or **Default Response Value**, and then type the value.
3. Click **Save**.

Testing Your Transformations

After you set up your mapping, you can test your transformations.

To test a transformation:

1. On the Transformations page, click .
2. Enter required information such as the object's Id.
3. Review the transformed response body. This is the response containing only the fields in your common resource.
4. Click **Original** to see the entire response JSON payload.
5. Test a Put or Patch by selecting the appropriate method, and then entering the JSON request.


 **Tip:** Copy the JSON payload from Transformed.

6. Click **Run**.

Adding Your Common Resource to the API Docs

You can add the common resource you create to the instances of each affected connector.

To add a common resource to API docs:













1. On the Transformations page, click  next to your connector instance.
2. Switch **Add to API Docs** on.
3. Click **Save**.

To confirm that the common resource is added to your API docs:

1. Go to an connector instance.
2. Hover over the instance card, and the click **API Docs**.
3. Scroll to your common resource.

Working with Nested Objects

We display sub-objects in dot notation. You can also use dot notation to nest objects in your common resource. For example, you might want to create nested address fields like those shown in the example below:

Account Level Fields			
	Field Name	Type	Display Name
  	address.street	string	
  	address.city	string	
  	address.state	string	
  	address.zip	string	

The JSON result of this nested object:

```
{
  "Address": {
    "city": "Cambridge",
    "state": "MA",
    "street": "1234567 Elm St",
    "zip": "99999"
  }
}
```

Map Complex Objects

You can use regular expressions as values for the JSON body parameters.

Examples:

- Get the value of the name field from the Products array where id = 4.

```
{
  "fields": [
    {
      "path": "AppleIpadName",
      "vendorPath": "Products[id=4].name"
    }
  ]
}
```

- Get the value of the name field from the Products array where id = 2.

```
{
  "fields": [
    {
      "path": "AppleNewProductName",
      "vendorPath": "details.Products[?(@.id=2)].name"
    }
  ]
}
```

- Get the touchId value from the Products array with id=2 which is inside the features object.

```
{
  "fields": [
    {
      "path": "AppleTouchProductId",
      "vendorPath": "Products[?(@.features.touchId=true)].id"
    }
  ]
}
```

- Get the Id value of the Products array at index 0.

```
{
  "fields": [
    {
      "path": "AppleProductId",
      "vendorPath": "Products[0].id"
    }
  ]
}
```

- Get a count of the Products array.

```
{
  "fields": [
    {
      "path": "AppleProductsCount",
      "vendorPath": "Products[*].size()"
    }
  ]
}
```

Using the OCNQL Query Override Field

SAP Open Connectors provides a query override field for any connectors whose native search specifications are different from the standard fields implemented on the resource in SAP Open Connectors. For example, if you are using the Hubspot Marketing connector and want to search for a customer's email, you would query on `person.email`, as opposed to simply "email," a more commonly used field. This means that when mapping Hubspot Marketing fields in a transformation, you must override the `EMAIL` field with `PERSON.EMAIL` in order to successfully use the search; otherwise, you will receive an invalid expression type error.

If the Query Field Name is left blank, the mapped field value will be used.

To use the query override field with a common resource, complete these steps:

1. Log in to SAP Open Connectors and click Common Resources.
2. On the My Resources tab, hover over the common resource and click Mappings.
3. On the Mappings tab, find the field whose name you want to override and click the Field Settings button.
4. On the Field Settings drawer, enter the overriding value to query at the endpoint in the Query Field Name field, and then click Save.
5. To test the query override, click the Try It Out button.

Known Connectors

All connectors support the OCNQL query override field. However, some connectors may need to use it more frequently than others based on the naming of their fields. See the connector's documentation for additional information.

- [Marketo](#)
 - [Hubspot Marketing](#)
-