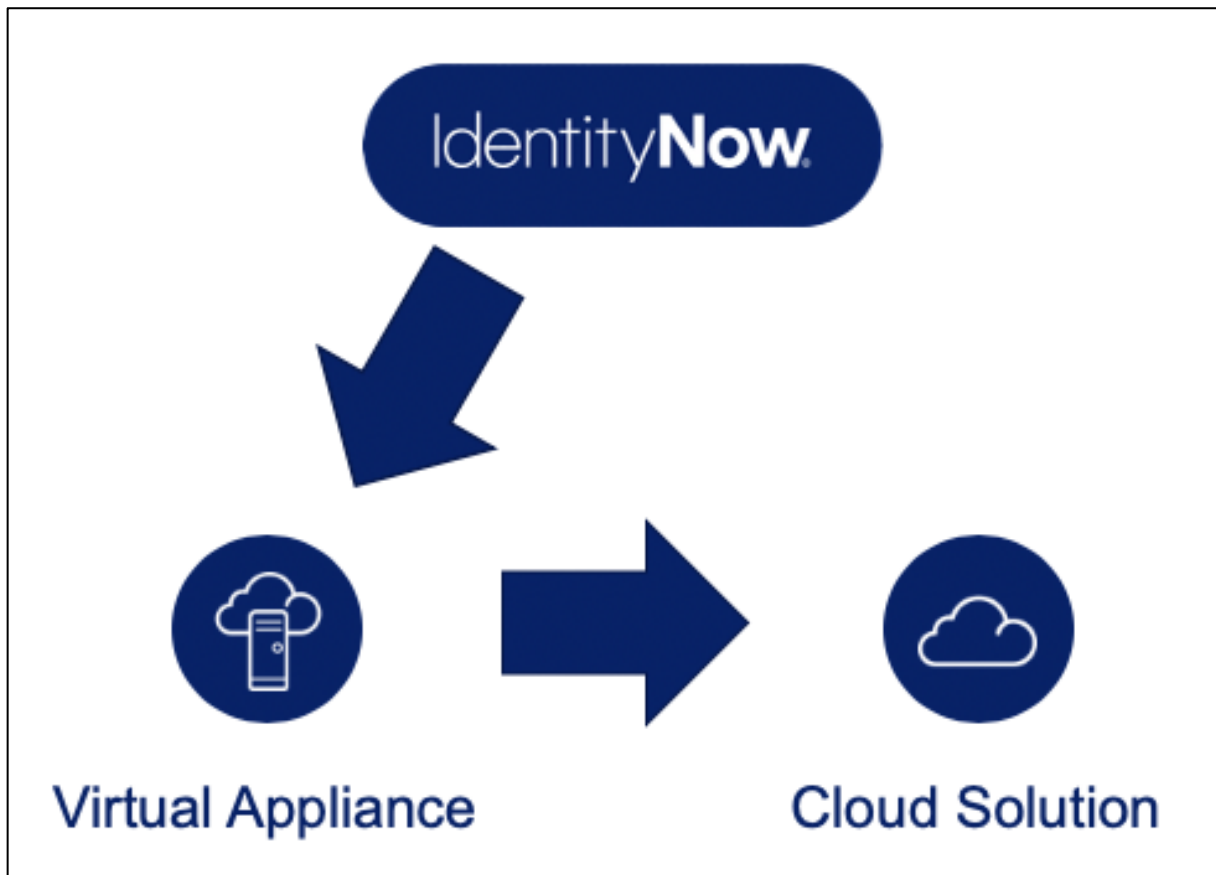


## Governing a Web Service using IdentityNow

In today's infrastructure there is usually one or more web service or cloud based system used by the company. To govern the access to this cloud based system, it is helpful to connect the existing web services to an existing Identity and Access Management tool.



*Graph.1: Architecture for the connection to the cloud solution.*

In this example I will show you how to simply add a system to an IdentityNow instance using the Web Services Connector provided by SailPoint.

### **Preconditions: API Documentation**

The precondition for connecting the cloud based solution to IdentityNow is a web facing application programming interface (API), which will disclose information about the accounts and entitlements on the system on hand.

This API is usually secured and you need to implement more complex security into this connector. To keep this example simple, this will not be covered by this article.

Here we will use an API which will produce JSON objects and has the following endpoints to receive data from the system.

**general** General Information ▼

**GET** `/ping` Check connectivity

**users** Information about user accounts ▼

**GET** `/users` Get user accounts

**GET** `/users/{id}` Get single user account

**GET** `/users/{id}/entitlements` Get user entitlements

**entitlements** Information about entitlement ▼

**GET** `/entitlements` Get entitlements


**GET** `/entitlements/{id}` Get entitlement attributes

These are the endpoints for the cloud solution, we are going to connect to IdentityNow (your solution could look similar to this).

### Test Connection

The first implemented operation is going to test the connection from your Virtual Appliance to your cloud solution. As seen in the list of endpoints, there is an endpoint available to check the connectivity with the route `/ping`. In your case this could be an endpoint to check the used authorization token.

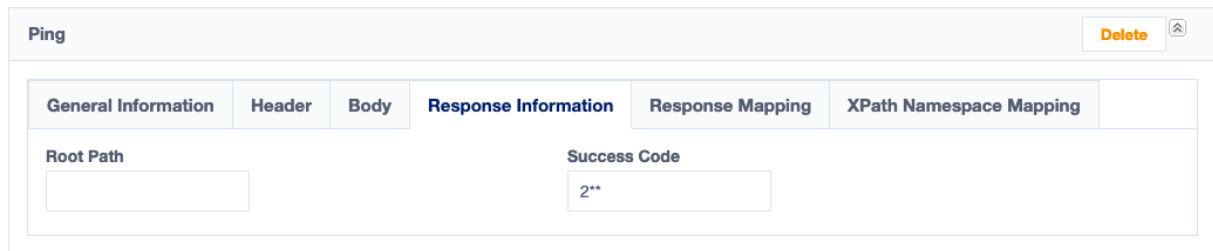
To implement this, we need to add a new operation from the type "Test Connection" and we will name it "Ping".

**Ping** Delete 

General Information	Header	Body	Response Information	Response Mapping	XPath Namespace Mapping
<p><b>Operation Name</b> <input type="text" value="Ping"/></p> <p><b>Operation Type</b> <span>Test Connection</span> ▼</p> <p><b>Context URL</b> <input type="text" value="/ping"/></p> <p><b>HTTP Method</b> <span>GET</span> ▼</p>					

Graph. 2: The configuration for Ping.

We will assume, that a successful connection will result in a response with the success code of 2\*\*. This needs to be added in the Response Information tab.



The screenshot shows a configuration window titled "Ping" with a "Delete" button in the top right corner. Below the title bar is a tabbed interface with the following tabs: "General Information", "Header", "Body", "Response Information" (which is selected), "Response Mapping", and "XPath Namespace Mapping". In the "Response Information" tab, there are two input fields: "Root Path" and "Success Code". The "Success Code" field contains the text "2\*\*".

*Graph. 3: Response Information for testing the connection.*

After successfully connecting the API to IdentityNow, the next step is to aggregate the entitlements.

### **Entitlement Aggregation**

Before we are able to implement the entitlements, we need to inspect the available endpoints to get the information about entitlements.

First we need to get a list of entitlements available. The following endpoint does give us an array of entitlements.

**GET**
**/entitlements** Get entitlements

Gets a list of all entitlements. Per page there is a limitation of 500 entitlements.

Parameters

Try it out

Name	Description
<b>offset</b> integer ( <i>formData</i> )	Items to be skipped  <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;">offset - Items to be skipped</div>

Responses

Response content type application/json

Code	Description
200	successful operation  <div style="display: flex; justify-content: space-between; font-weight: bold; font-size: 0.9em;"> <span>Example Value</span> <span>Model</span> </div> <div style="background-color: #333; color: #eee; padding: 10px; margin-top: 5px; border-radius: 4px;"> <pre> {   "items": [     {       "id": "91827364",       "name": "can log in"     }   ] }</pre> </div>
401	not authenticated
500	internal server error

Graph. 4: Endpoint for entitlement list.

To implement this entitlement list, we need to add a new operation with the type "Entitlement Aggregation".

The screenshot shows the 'Aggregate Entitlements' configuration form with the 'General Information' tab selected. The form contains the following fields:

- Operation Name:** Aggregate Entitlements
- Operation Type:** Entitlement Aggregation (dropdown menu)
- Context URL:** /entitlements
- HTTP Method:** GET (dropdown menu)

Graph. 5: Operation for entitlement aggregation.

As seen in the endpoint documentation, the entitlements will all be in an array named "items" and the response code will be 200 upon a successful request. Implementing this in the Response Information tab looks as follows:

The screenshot shows the 'Aggregate Entitlements' configuration form with the 'Response Information' tab selected. The form contains the following fields:

- Root Path:** \$.items
- Success Code:** 200

Graph. 6: Response information for entitlement aggregation.

The API implements a pagination which we will also need to implement. Gladly SailPoint offers a decent scripting language to realize this.

In this endpoint we get 500 items with every request. The path parameter offset will define, how many items we will skip for this request. So the logic for offset needs to be: 0 -> 500 -> 1000 -> ... until there are less than 500 items returned.

The implementation can look as follows:

**Aggregate Entitlements** Delete

General Information | Header | Body | Response Information | Response Mapping | XPath Namespace Mapping | **Paging**

Initial Page Offset:  Page Size:

**Paging Steps:**

```

$sysparm_limit$ = 500
TERMINATE_IF $RECORDS_COUNT$ < $sysparm_limit$
$sysparm_offset$ = $sysparm_offset$ + $sysparm_limit$
$endpoint.fullUri$ = $application.baseUrl$ + "/entitlements?offset=" + $sysparm_offset$
    
```

Graph. 7: Implementation for pagination.

In the first line we are defining the parameter with the limitation of items aggregated during every request.

The second line is the check for termination. If less than 500 entitlements are aggregated, there will be no more request.

The third line calculates the offset using the previous offset.

And the last line calculates the new application URL with the offset parameter set to the value to skip items.

After adding the information needed to make the request, we need to add the mapping for the data to entitlement attributes. In this example we will not show how to modify the schema in the source, to add and remove needed entitlement attributes.

Since we added the items array to the root path in the previous step, we are now able to access the values from the response directly in the response mapping.

**Aggregate Entitlements** Delete

General Information | Header | Body | Response Information | **Response Mapping** | XPath Namespace Mapping | Paging

Schema Attribute	Attribute Path	
<input type="text"/>	<input type="text"/>	+ Add
id	id	X
name	name	X

Graph. 8: Response mapping for entitlements

Just the name and the id for the entitlements will not be enough for governing them in IdentityNow. With a second request for every entitlement, we can add the attributes to the corresponding entitlement.

First we will check the documentation for the specifications on the endpoint.

GET /entitlements/{id} Get entitlement attributes

Gets all attributes to the entitlement.

**Parameters**
Try it out

Name	Description
<b>id</b> <span style="color: red; font-size: small;">* required</span> <b>integer(\$int64)</b> <small>(path)</small>	ID of entitlement  <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px; width: fit-content;">             id - ID of entitlement           </div>

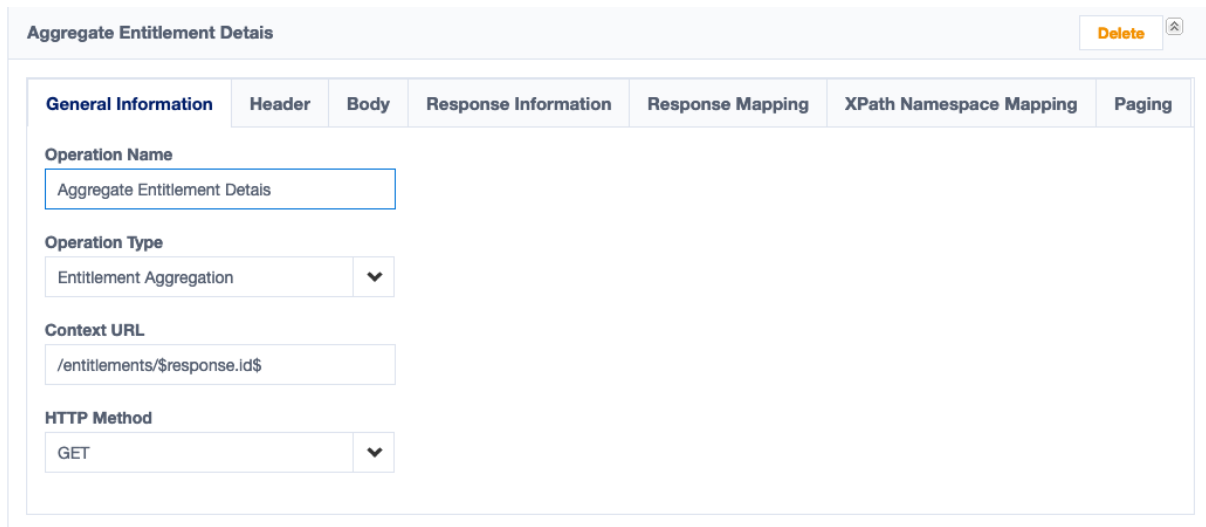
**Responses**

Response content type application/json ▼

Code	Description
200	successful operation  <div style="display: flex; justify-content: space-between; font-size: small; margin-top: 5px;"> <span>Example Value</span> <span>Model</span> </div> <div style="background-color: #333; color: white; padding: 10px; margin-top: 5px; border-radius: 5px;"> <pre> {   "id": "91827364",   "name": "can log in",   "description": "This is the entitlement for users to be able to log into the cloud solution",   "owner": "Max Mustermann" } </pre> </div>
401	not authenticated
500	internal server error

Graph. 9: Endpoint for entitlement details.

With this information we can add a second Entitlement Aggregation operation in IdentityNow. It is crucial that this operation is added after the first aggregation as we need to use the response of the first request to perform these requests.



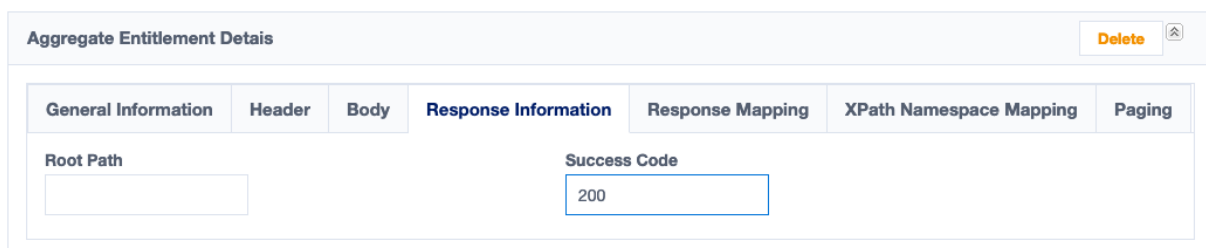
**Aggregate Entitlement Details** Delete

General Information	Header	Body	Response Information	Response Mapping	XPath Namespace Mapping	Paging
<p><b>Operation Name</b> Aggregate Entitlement Details</p> <p><b>Operation Type</b> Entitlement Aggregation</p> <p><b>Context URL</b> /entitlements/\$response.id\$</p> <p><b>HTTP Method</b> GET</p>						

Graph. 10: General information on entitlement detail aggregation.

Here we can see that the response information from the previous aggregation can be used to perform the detail aggregation for entitlements. In this case we will use the variable id in the URL which looks like this: \$response.id\$.

Since this request does not have a list of items, the Response Information will only contain the success code for the request.




**Aggregate Entitlement Details** Delete

General Information	Header	Body	Response Information	Response Mapping	XPath Namespace Mapping	Paging
<p><b>Root Path</b></p> <p><b>Success Code</b> 200</p>						

Graph. 11: Response information for detailed entitlement aggregation.

The only thing left is the Response Mapping for this request. Since we got the id and the name from the initial response, we only need to map the description and the owner of this entitlement.



**Aggregate Entitlement Details**
Delete 

General Information	Header	Body	Response Information	Response Mapping	XPath Namespace Mapping	Paging
Schema Attribute		Attribute Path				
<input type="text"/>		<input type="text"/>		<span style="color: #0056b3;">+ Add</span>		
description		description		<span style="color: #0056b3;">✕</span>		
owner		owner		<span style="color: #0056b3;">✕</span>		

*Graph. 12: Response Mapping for detailed entitlement aggregation.*

With that implementation, we are now able to aggregate entitlements and their detailed information.

The next step is to add accounts to the connector.

### **Account Aggregation**

Account aggregation will work similarly to entitlement aggregation. The example API uses an endpoint to get a list of all users and an endpoint to get more details for a single user. In addition to the entitlements, there is an endpoint to get all entitlements one users has.

The first operation we need to implement is the Account Aggregation to get the list of users. The following endpoint shows the endpoint in detail.

GET
/users
Get user accounts

Gets a list of user accounts. Per page there is a limitation of 500 users.

Parameters

Try it out

Name	Description
<b>offset</b> <b>integer</b> (formData)	Items to be skipped  <div style="border: 1px solid #ccc; padding: 2px; background-color: #fff9c4; display: inline-block;">offset - Items to be skipped</div>

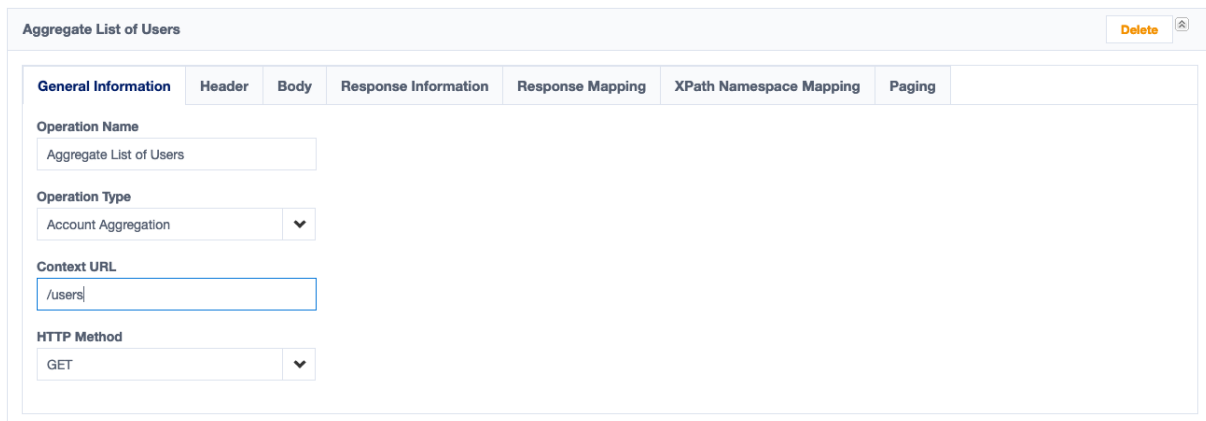
Responses

Response content type application/json

Code	Description
200	successful operation  <div style="display: flex; justify-content: space-between; font-size: 0.8em; font-weight: bold; margin-bottom: 5px;"> <span>Example Value</span> <span>Model</span> </div> <div style="background-color: #333; color: #eee; padding: 10px; border: 1px solid #444; font-family: monospace; font-size: 0.9em;"> <pre> {   "items": [     {       "id": "MM911835",       "name": "Max Mustermann"     }   ] }                     </pre> </div>
401	not authenticated
500	internal server error

Graph. 13: API endpoint to get list of users.

We will first add a new operation in IdentityNow with the operation type Account Aggregation.



*Graph. 14: General configuration for account aggregation.*

Since the response JSON has the same structure as the endpoint for the entitlements, this will be the same.

Pagination is also handled as it is in the entitlement aggregation, however the code needs to be altered to the users endpoint and not the entitlements endpoint.

After adding the initial list accounts we need to add the details for each account. The API also offers an endpoint for the details for each user account.

GET
/users/{id}
Get single user account

Gets detailed attributes for a single user account.

Parameters

Try it out

Name	Description
<b>id</b> <span style="color: red; font-size: small;">* required</span> <b>integer(\$int64)</b> <small>(path)</small>	ID of user  <div style="border: 1px solid #ccc; padding: 2px; margin-top: 5px; display: flex; align-items: center;"> <span style="font-size: small; margin-right: 5px;">id - ID of user</span> <input style="flex-grow: 1; border: none;" type="text"/> </div>

Responses

Response content type application/json v

Code	Description
200	successful operation  <div style="display: flex; justify-content: space-between; font-size: small; margin-bottom: 5px;"> <span style="border-bottom: 1px solid #ccc; padding-bottom: 2px;">Example Value</span> <span style="border-bottom: 1px solid #ccc; padding-bottom: 2px;">Model</span> </div> <div style="background-color: #333; color: #eee; padding: 10px; border-radius: 5px; margin-bottom: 5px;"> <pre style="margin: 0;">{   "id": "MM911835",   "name": "Max Mustermann",   "firstname": "Max",   "lastname": "Mustermann",   "department": "IT",   "costcenter": 12341234 }</pre> </div>
401	not authenticated
500	internal server error

Graph. 15: API endpoint for user account details.

With the list of id's we can now get the details for every user by adding an operation Account Aggregation using the \$response.id\$ variable to make the call.

The screenshot shows the 'Aggregate User Attributes' configuration page with the 'General Information' tab selected. The fields are as follows:

<b>Operation Name</b>	Aggregate User Attributes
<b>Operation Type</b>	Account Aggregation
<b>Context URL</b>	/users/\$response.id\$
<b>HTTP Method</b>	GET

Graph. 16: Operation for detailed user account aggregation.

The Response Information will only contain the success code 200 for a successful call.

The response mapping needs to be configured for the returned JSON object. It is also possible to add only certain values to be written to IdentityNow. In this example we want to have everything, except for the cost center.

The screenshot shows the 'Aggregate User Attributes' configuration page with the 'Response Mapping' tab selected. The configuration table is as follows:

Schema Attribute	Attribute Path	
		+ Add
name	name	✘
firstname	firstname	✘
lastname	lastname	✘
department	department	✘

Graph. 17: Response mapping for detailed user account aggregation.

With this configuration we are now able to aggregate all user accounts with the needed attributes we want to govern in IdentityNow.

Adding the entitlements, that each user has is similar to this process. First, we will have a look at the corresponding API endpoint documentation.

GET **/users/{id}/entitlements** Get user entitlements

Gets a list of all entitlements inherited by the user.

**Parameters**
Try it out

Name	Description
<b>id</b> <span style="color: red;">★ required</span> <b>integer(\$int64)</b> (path)	ID of user  <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;">id - ID of user</div>

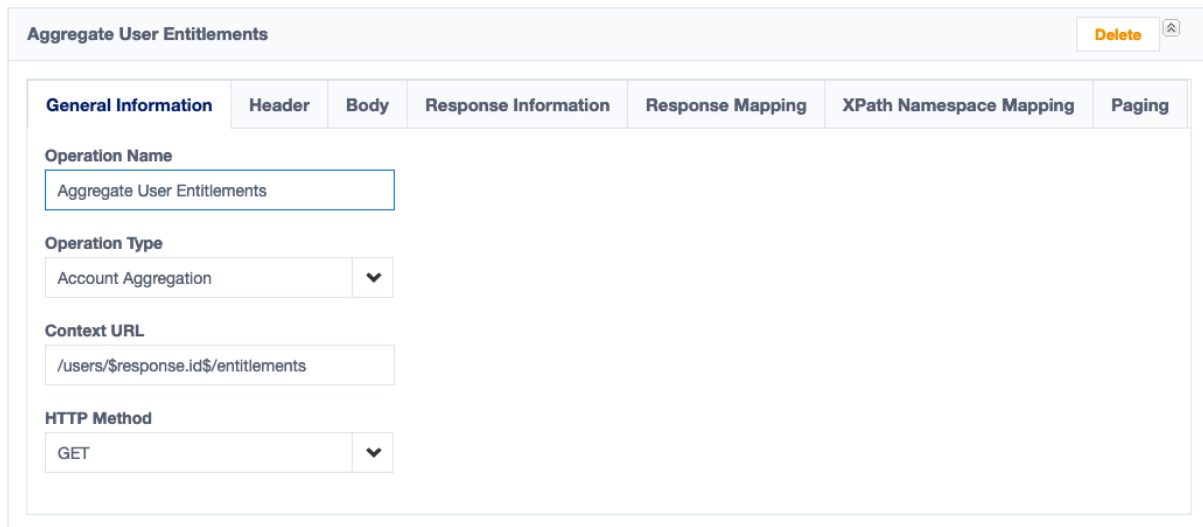
**Responses**
Response content type

application/json

Code	Description
200	successful operation  <div style="display: flex; justify-content: space-between; font-weight: bold; font-size: 0.9em;"> <span>Example Value</span> <span>Model</span> </div> <div style="background-color: #333; color: #eee; padding: 10px; margin-top: 5px; border: 1px solid #444;"> <pre> {   "entitlements": [     {       "id": "91827364",       "name": "91827364"     }   ] }           </pre> </div>
401	not authenticated
500	internal server error

Graph. 18: API endpoint to get a list of entitlements inherited by the user.

We will first need to add a third operation from the type Account Aggregation. This Operation also needs to be added after the initial aggregation of the user list.



**Aggregate User Entitlements** Delete

General Information	Header	Body	Response Information	Response Mapping	XPath Namespace Mapping	Paging
<p><b>Operation Name</b> Aggregate User Entitlements</p> <p><b>Operation Type</b> Account Aggregation</p> <p><b>Context URL</b> /users/{response.id}/entitlements</p> <p><b>HTTP Method</b> GET</p>						

Graph. 19: Operation for user account entitlement aggregation.

The Response Information will be the same as for the detailed account attributes aggregation. However, this time we are not able to add entitlements as the root path because a list of entitlements in one attribute is needed.

To achieve this, we can add the following mapping.



**Aggregate User Entitlements** Delete

General Information	Header	Body	Response Information	Response Mapping	XPath Namespace Mapping	Paging						
<table border="1"> <thead> <tr> <th>Schema Attribute</th> <th>Attribute Path</th> <th></th> </tr> </thead> <tbody> <tr> <td>entitlements</td> <td>entitlements[*].id</td> <td>X</td> </tr> </tbody> </table>							Schema Attribute	Attribute Path		entitlements	entitlements[*].id	X
Schema Attribute	Attribute Path											
entitlements	entitlements[*].id	X										

Graph. 20: Response Mapping for user entitlement aggregation.

With this configuration we are now able to test the connection to the API, aggregate the entitlements and the accounts. After this step, we are now able to govern the data within the cloud solution.

Sometimes it can also be useful to aggregate a single account manually. For that case we will now add the Get Object operation.

## Get Object

Luckily, we can use a lot of the configuration, that we used in the account aggregation operation. We will use the API endpoints

- /users/{id} and
- /users/{id}/entitlements.

We will add two new operations from the type Get Object calling these endpoints. This time we do not have the response variable available, however we can use the native identity.

Graph. 21: Get Object configuration for attributes.

Adding the configuration for the Response Information and the Response Mapping from the Account Aggregation operation, we can move on to the entitlements for the user object.

Graph. 22: Get Object configuration for entitlements.



For this operation we will also add the configuration from the previous account aggregation operation.

Now the connection between the cloud solution and IdentityNow is set up and we are able to connect the API, aggregate entitlements, aggregate accounts and fetch updates on single identities.

---

This article did not cover security or provisioning for the cloud solution. Also, modifications to the IdentityNow source schema are not covered. The source code for the swagger documentation of the API can be found at <https://gist.github.com/DigitalPhilosopher/ca2b926e37ecd072f4fc14a7dd01d1ef>.

For more information on IdentityNow, you can visit <https://www.sailpoint.com/>.

For help on implementing your Identity and Access Management System, you can reach us at <https://www.kogit.de/unternehmen/kontakt-standort/>.