DCS REST API

Ecosystem Version 161.3+
September 2022

Contents

Introduction

Enov8's REST-API allows you to automatically Create, Update & Read information. Which in turn allows you to integrate other platforms and automation scripts. For example CICD scripts to capture "version", or test automation scripts to capture "health".

Currently the following three types of requests are supported.

1. POST (Create)
2. PUT (Update)
3. GET (Read)
4. DELETE

REST-API Prerequisites

The following will be required before using the Enov8 REST-API

- The URL of your Enov8 instance
  e.g. https://company.enov8.com/ecosystem/api/

- An "API KEY" comprising of the following parameters
  - app_id (Application ID)
  - app_key (Application Key)

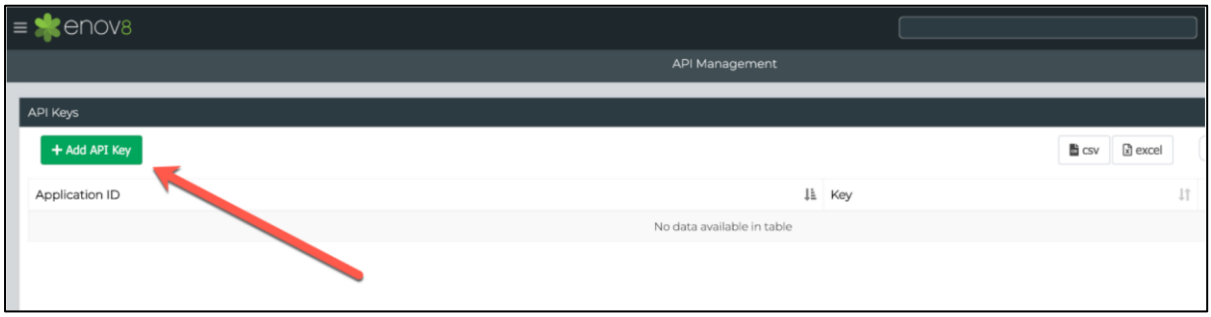
- An Enov8 platform user account with the required permissions
  - user_id

  Note: That a dedicated "Service User Account" is typically created.

Creating API Token

The following snapshots have been taken from the Ecosystem support portal.

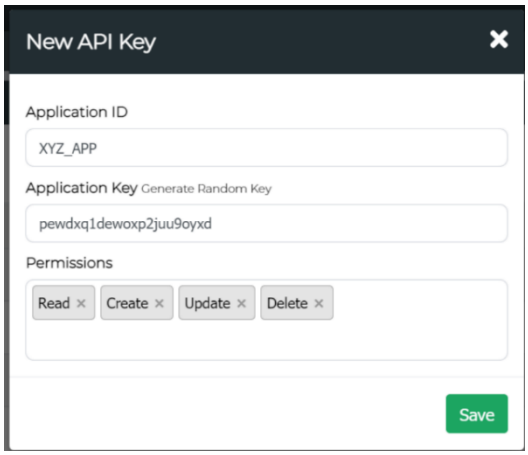
As a System Admin user login to the Enov8 platform and navigate to **Configuration Management > System Administration > API Management**. Click Add API Key



Enter your Application ID and click Generate Random Key for your Application Key.

Also select the permissions for the API Key (Read, Create, Update, Delete) Note: Retain these details for future use.

Click Save. You should receive a message saying API Key has been created successfully



Retrieving DCS Objects

Data related to DCS objects can be retrieved from Ecosystem by using a GET REST API call. This API request consists of a URL and header parameters.

URL Structure

{ecosystem api url}/DataConnection
{ecosystem api url}/DataConfiguration
{ecosystem api url}/DataExecutionResult

Header

The following parameters need to be passed as part of the request header to authenticate the request.

	KEY	VALUE
☑	app_id	YOUR API ID
☑	app_key	YOUR API KEY
☑	user_id	YOUR API USER ID

In addition to these parameters, other headers can be passed to further filter the results and restrict the objects to be retrieved. The Properties field can also be used to specify which keys should be included within each JSON object returned.

For Data Connection objects, some keys that can be used for filtering are Status and Type.

☑	Status	Active
☑	Type	MSSQL
☑	Properties	System ID;Connection Name

For Data Execution Results objects, Status and Type can also be used alongside Start Timestamp> which forces objects to have a timestamp after the specified datetime.

☑	Status	Passed
☑	Type	Profile
☑	Start Timestamp>	13-09-2022 15:17:33
☑	Properties	System ID;Status,Type,Start Timestamp

Response

If the request is successful, a status code of 200 will be returned. The response body returned for each DCS object type will have different fields, but every object will contain a unique System ID.

Data Connection example

For Data Connection objects, the response body will follow the format below:


```
{
    "System ID": "{eco_id}",
    "System User": "{user_name}",
    "System Time": "{sys_time}",
    "Connection Name": "{conn_name}",
    "Connection String": "{conn_string}",
    "Status": "Active",
    "Remediation Status": "",
    "Remediated By": null,
    "Type": "{conn_type}",
    "Refresh Date": null,
    "Profiling Date": "{profile_date}",
    "Masking Date": "{mask_date}",
    "Validation Date": "{validation_date}",
    "Connection Size": "{conn_size}",
    "Tag": "",
    "Label": "",
    "Agent": "",
    "System Component": "{sys_comps}",
    "Execution Result": "{data_execs}",
    "Data Configuration": "{data_configs}",
    "Data SQL": "{data_sql_names}",
    "Assigned To": "{assigned_user}",
    "Notification": "",
    "Organisation": "{org_name}"
}
```


However, if filtering is performed using the header included above, it will only contain the specified keys:


```
{
    "System ID": "{eco_id}",
    "Connection Name": "{conn_name}"
}
```

Data Execution Result example

For Data Execution Result objects, the response body will follow the format below:

```
{
    "System ID": "{eco_id}",
    "System User": "{user_name}",
    "System Time": "{sys_time}",
    "Percentage Completed": "{percen_comp}",
    "Status": "{scan_status}",
    "Remediation Status": "",
    "Remediated By ": null,
    "Type": "{scan_type}",
    "Priority": "",
    "Start Timestamp": "{start_time}",
    "End Timestamp": "{end_time}",
    "Executed By": null,
    "Data Configuration": "{config_name}",
    "Data Connection": "{conn_name}",
    "Results": "{scan_results}",
    "Logs": "{scan_logs}",
    "JSON": "{json_results}",
    "Source ID": "{source_id}",
    "Organisation": "{org_name}"
}
```

Using the header included above, the response body would contain only the specified keys:

```
{
    "System ID": "{eco_id}",
    "Status": "Passed",
    "Type": "Profile",
    "Start Timestamp": "13-09-2022 15:17:35"
}
```

Multiple objects found

If the API request finds multiple DCS objects for a given type, it will return a list of these objects instead:

```
[
    {
        "System ID": "{eco_id1}",
        "Connection Name": "{conn_name1}"
    },
    {
        "System ID": "{eco_id2}",
        "Connection Name": "{conn_name2}"
    }
]
```

Updating DCS Objects

Once the System ID for an object has been obtained, it can be updated using a PUT REST API call. This API request consists of a URL, header parameters and a request body.

URL Structure

{ecosystem api url}/DataConnection
{ecosystem api url}/DataConfiguration
{ecosystem api url}/DataExecutionResult

Header

The following parameters need to be passed as part of the request header to authenticate the request.

	KEY	VALUE
☑	app_id	YOUR API ID
☑	app_key	YOUR API KEY
☑	user_id	YOUR API USER ID

Body

The request body should contain the System ID of the DCS object as well as any other fields that should be updated.

For example, to update the Status of a Data Configuration object to Active the following body should be used:

```
{
    "System ID": "{data_config_id}",
    "Status": "Active"
}
```

Response

If updating a DCS object is successful, a status code of 200 and the following body will be returned:

```
{
    "success": true,
    "total_attempted": 1,
    "total_updated": 1,
    "result": [
        {
            "success": true,
            "System ID": "{eco_id}"
        }
    ],
    "value passed": "{passed_body}"
}
```

This contains the updated object's System ID as well as the body that was passed in the PUT request.

Creating DCS Objects

DCS objects can be created in Ecosystem using a POST REST API call. This API request consists of a URL, header parameters and a request body.

URL Structure

{ecosystem api url}/DataConnection
{ecosystem api url}/DataConfiguration
{ecosystem api url}/DataExecutionResult

Header

The following parameters need to be passed as part of the request header to authenticate the request.

	KEY	VALUE
☑	app_id	YOUR API ID
☑	app_key	YOUR API KEY
☑	user_id	YOUR API USER ID

Body

The request body for each DCS object will have a different JSON format. Here, the JSON required for creating a Data Connection object has been explained in detail.

Data Connection example

For creating a Data Connection object, the JSON format below should be used:

```
{
    "Connection Name": "{connection_name}",
    "Type": "{database_type}",
    "Status": "Active",
    "Assigned To": "{eco_id_of_group}",
    "Organisation": "{eco_id_of_org}",
    "Connection String": "{b64_encoded_json}"
}
```

Field	Description	Format / examples
Connection Name	Unique name given to Data Connection object	Any unique string
Type	Type of DataConnection object	Any supported type e.g., MySQL, MSSQL, File - Delimited
Status	Status of DataConnection, should be set to Active on creation	Should be Active, Inactive or Deleted
Assigned To	Assigned user or group	Ecosystem object ID e.g. ECO-000000000001
Organisation	Ecosystem organisation	Ecosystem object ID e.g. ECO-000000000001
Connection String	Base 64 encoded connection string	More detail below

Example payload:


```
{
    "Connection Name": "csv_file",
    "Type": "File - Delimited",
    "Status": "Active",
    "Assigned To": "ECO-000000004182",
    "Organisation": "ECO-000000003945",
    "Connection String":
        "eyAiRmlsZSBQYXRoIjogIkwzQmhkR2d2ZEc4dlptbHNaUzUwZUhRPSIsICJGaWx1IER1bGltaXR1ciI6ICJMQT09IiwgI
        iLCAiSGVhZGVyIENvdW50IjogIk1BPT0iLCAiRm9vdGVyIENvdW50IjogIk1BPT0iLCAiRmlsZSBRdW90ZSI6ICIiIH0="
}
```


Connection String field

The Connection String field contains a base 64 encoded JSON that is required for the creation of Data Connection objects. Each field in this JSON is also base 64 encoded. This field should have a different format depending on the Data Connection Type being added. To better understand this field, two examples have been provided.

MSSQL example

For MSSQL Data Connection objects, the Connection String JSON should be formatted as follows:

```
{
    "Server": "{b64_server_address}",
    "Username": "",
    "Password": "",
    "Database": "{b64_database}",
    "Port": "{b64_port}",
    "Schema": "{b64_schema}"
}
```

```
{
    "Server": "bG9jYWxob3N0",
    "Username": "",
    "Password": "",
    "Database": "dGVzdF9kYg==",
    "Port": "MTQzMw==",
    "Schema": "ZGJv"
}
```

This JSON would then be base 64 encoded to produce the Connection String field.

Note: Username and Password fields are empty strings here so that Windows authentication will be used. To ensure secure storage of passwords, Data Connection objects cannot be created with Password fields currently as they must be encrypted further prior to being stored in Ecosystem.

File - Delimited example

For File - Delimited Data Connection objects, the Connection String JSON should be formatted as shown below:

```
{
    "File Path": "{b64_file_path}",
    "File Delimiter": "{b64_delimiter}",
    "File Header": "{b64_whether_header}",
    "Header Count": "{b64_num_header_rows}",
    "Footer Count": "{b64_num_footer_rows}",
    "File Quote": "{b64_file_quote}"
}
```

```
{
    "File Path": "L3BhdGgvdG8vZmlsZS50eHQ=",
    "File Delimiter": "LA==",
    "File Header": "WWVz",
    "Header Count": "MA==",
    "Footer Count": "MA==",
    "File Quote": ""
}
```

As with the MSSQL type, this JSON should be base 64 encoded to produce the required Connection String.

Response

If the creation of a DCS object is successful, a status code of 201 and the following body will be returned:

```
{
    "success": true,
    "total_attempted": 1,
    "total_created": 1,
    "result": [
        {
            "success": true,
            "System ID": "{eco_id}"
        }
    ],
    "value passed": "{passed_body}"
}
```

This contains the created object's System ID as well as the body that was passed in the POST request.

Testing Connection

To check that a Data Connection object's details are correct, an API request can be sent to test the connection.

URL Structure

{ecosystem api url}/dcstestconnection

Header

The following parameters need to be passed as part of the request header to authenticate the request.

	KEY	VALUE
☑	app_id	YOUR API ID
☑	app_key	YOUR API KEY
☑	user_id	YOUR API USER ID

Body

There are two different request bodies that can be passed to test the connection to a Data Connection object. The first of these includes the Connection String of the Data Connection whilst the second uses the System ID for the Data Connection object in EcoSystem. To build a Base 64 encoded Connection String, see this section.

```
{
    "database_type": "{db_type}",
    "org_id": "{org_id}",
    "user_eco_id": "{user_id}",
    "connection_string": "{b64_encoded_json}"
}
```

```
{
    "database_type": "{db_type}",
    "org_id": "{org_id}",
    "user_eco_id": "{user_id}",
    "connection_id": "{dataconn_id}"
}
```

Response

If the Data Connection details are correct and the database / file can be accessed, a status code of 200 and the following response body will be returned:

```
{
    "jsonrpc": "2.0",
    "message": "Connection to Data Source was successful!",
    "success": true
}
```

Executing Profiling

DCS profiling can be executed using a POST REST API Call to Ecosystem. This API request consists of a URL, header parameters and a request body.

Obtaining required System IDs

To execute profiling via the Ecosystem REST API, the System ID is needed for several fields in the request body.

```
{
    "configuration_id": "{config_id}",
    "organisation_id": "{org_id}",
    "user_eco_id": "{user_id}",
    "assigned_to": "{assigned_group}",
    "connection_id": "{dataconn_id}"
}
```

For the Organisation, User and Assigned To fields, these System IDs can be obtained via the Ecosystem frontend search bar.

However, the System ID for DCS objects (Data Connection and Data Configuration) must be obtained via a GET request to Ecosystem or from the POST request used to create the object. For details on how to perform these requests, see the Retrieving Object ID and Creating Objects sections.

URL Structure

Given that all System IDs have been obtained, profiling can be executed using a POST request to the following URL.

{ecosystem api url}/rundcs

Header

As with the other requests, the following headers are required for authentication

	KEY	VALUE
☑	app_id	YOUR API ID
☑	app_key	YOUR API KEY
☑	user_id	YOUR API USER ID

Body

The request body should follow the JSON format shown below, where each field uses the object's Ecosystem System ID.

```
{
    "configuration_id": "{config_id}",
    "organisation_id": "{org_id}",
    "user_eco_id": "{user_id}",
    "assigned_to": "{assigned_group}",
    "connection_id": "{dataconn_id}"
}
```

Response

If the profiling execution is successful, a status code of 200 and the following body will be returned:

```
{
    "jsonrpc": "2.0",
    "message": "Data Compliance Script is being executed.",
    "success": true
}
```

Executing Masking

DCS masking can also be executed using a POST REST API Call to Ecosystem. This API request consists of a URL, header parameters and a request body.

Preparing masking config

Following the execution of profiling, a masking config will be automatically created in Ecosystem. This masking config must be updated prior to masking execution and the System ID must be obtained.

To obtain the System ID for this config, a request should be performed to retrieve the Data Configuration object as described in the Retrieving Object ID section. This request should contain additional headers to further filter the results, such as Status, Type and Data Connection name in the screenshot below.

☑	Status	Inactive
☑	Type	Mask
☑	Data Connection	{dataconn_name}

Once the System ID for the mask config has been obtained, the Status can be updated via a PUT request as described in the Updating Objects section.

URL Structure

Given that all System IDs have been obtained, masking is executed using a POST request to the following URL.

{ecosystem api url}/rundcs

Header

As with the other requests, the following headers are required for authentication

	KEY	VALUE
☑	app_id	YOUR API ID
☑	app_key	YOUR API KEY
☑	user_id	YOUR API USER ID

Body

The request body should follow the JSON format shown below, where each field uses the object's Ecosystem System ID.

```
{
    "configuration_id": "{config_id}",
    "organisation_id": "{org_id}",
    "user_eco_id": "{user_id}",
    "assigned_to": "{assigned_group}",
    "connection_id": "{dataconn_id}"
}
```

Response

If the masking execution is successful, a status code of 200 and the following body will be returned:

```
{
    "jsonrpc": "2.0",
    "message": "Data Compliance Script is being executed.",
    "success": true
}
```