Working with Generic API Task of Cisco UCS Director, Release 6.6

First Published: 2018-04-27

Why Generic API Task

Cisco UCS Director has a repository of out-of-the box tasks that are built around specific operations for a particular device. These tasks have pre-defined inputs and pre-defined outputs. These tasks are developed by developers who understand Cisco UCS Director framework and also have domain expertise around the device for which the task is meant for.

In certain situations, these out-of-the box tasks doesn't meet your requirements. Say, for example:

1. If new parameters are added to the device specific API to enhance the existing functionality, you have to wait till the next release for the task enhancement from the Cisco UCS Director development team.
2. If the device exposes a new functionality, you have to wait till the next release to get the additional task for a given connector to utilize the new functionality.
3. If you want to orchestrate the device operations which is currently not supported by Cisco UCS Director as a connector, you have to wait for the release of Cisco UCS Director in which the new device is supported as a connector.
4. If you want to trigger a Web service by taking run-time parameters as the tasks are executed in Cisco UCS Director, this approach is currently not supported.

To overcome these situations, Cisco UCS Director provides:

- A custom task framework where you can utilize CloupiaScript to write any device specific logic and achieve the desired result.

  Custom tasks provide the required flexibility but at the same time you must have good understanding about the Cisco UCS Director exposed APIs and on how to write CloupiaScript (which is mainly JavaScript).

- Generic tasks such as, SSH task, which allows you to automate operations on device which can be accessed through SSH. As SSH is a standard way of interacting with devices, you can easily automate such operations without depending on the out-of-the-box tasks.

Most of the devices which require the use of custom tasks expose their functionalities via APIs (XML/JSON). The standard way of consuming any API is a normal HTTP/HTTPs-based call which is provided by clients such as Postman, in a very simple way.

In the Release 6.6, Generic API task is introduced as an out-of-box task to help you to consume any API in a standard HTTP/HTTPs manner.

The pre-defined structure of a HTTP/HTTPs based API includes:

1. IP address
2. Port
3. URL path

4 Headers

5 Request body

6 Authentication parameters

What you can expect from Generic API Task

The generic API task:

1 Offers a simple interface to consume any API.

2 Allows you to define run-time variables which can be evaluated during run-time or as task input.

3 Provides ways to output the response or part of the response, as needed.

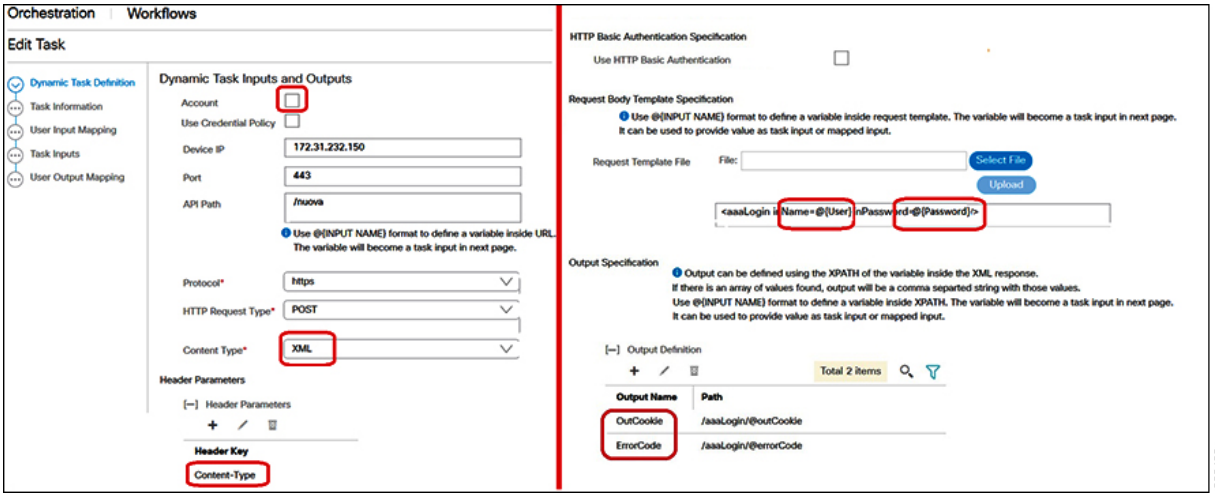
4 Supports rollback operation.

5 Consumes information about the devices that are managed by Cisco UCS Director.

Generic API Task Structure

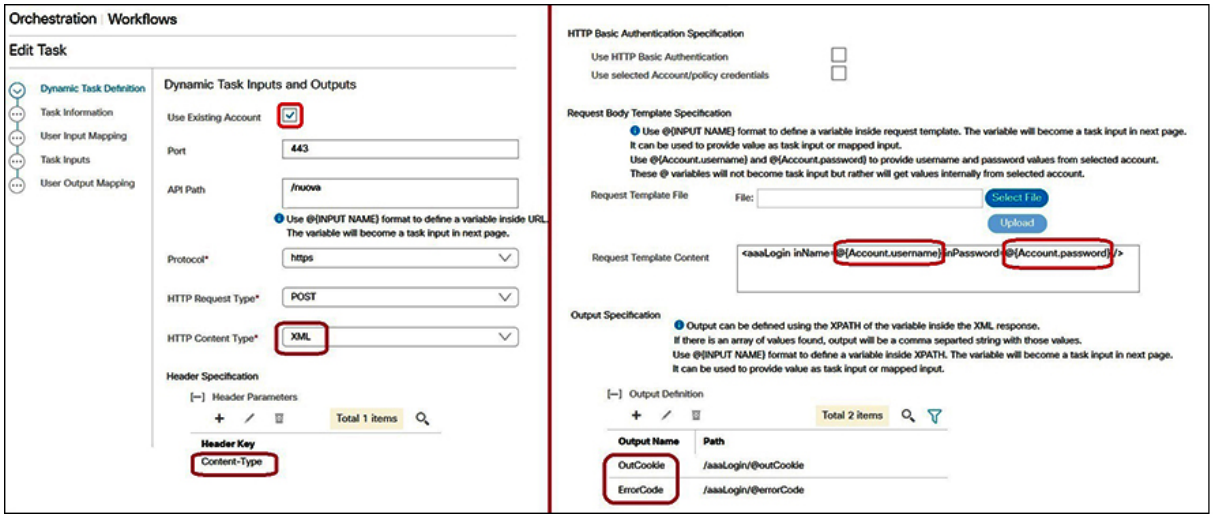
The generic API task has an additional page to define the API structure and required variables. The figure 1 provides the generic API structure of devices not managed by Cisco UCS Director.

Figure 1: Generic API Task – Unmanaged Device



The figure 2 provides the generic API structure of devices managed by Cisco UCS Director.

Figure 2: Generic API Task – Managed Device



In this page, you can:

1. Define API to make GET, POST, DELETE and PUT methods based HTTP calls.
2. Consume JSON or XML based APIs.
3. Opt to enable HTTP-based authentication to internally append an authorization header with username and password encoded in Base64 format.
4. Check the **Use selected Account/policy credentials** check box to use existing credentials. Uncheck this box to provide the credential information as task input.
5. Define outputs for the task using standards such as XPath and JSONPath. You can find out the XPath or JSONPath of the node or element value in the response and define the same as task output.
6. Opt to enable or disable to output the complete API response.

Defining Outputs

You can choose to define outputs based on the XPath or JSONPath of the variables in the API response. You can also choose to output the complete response as an output or hide the complete response during the task execution.

Example 1: Executing a Login API for the Cisco UCS Manager Account

When you execute a login API for the Cisco UCS Manager account using a generic API task, the successful response for the API is as follows:

```
<aaaLogin
    response="yes"
    outCookie="<real_cookie>"
    outRefreshPeriod="600"
    outPriv="aaa,ext-lan-policy,ext-lan-qos,ext-san-policy,operations,
                pod-policy,pod-qos,read-only"
```

```
    outDomains="mgmt02-dummy"
    outChannel="noencssl"
    outEvtChannel="noencssl">
</aaaLogin>
```

If you want to use the value of **outCookie** attribute from this response as an input parameter for further operations, you have to define an output variable as:

- **Output Name :** Cookie
- **Path :** /aaaLogin/@outCookie

Where, **/aaaLogin/@outCookie** is the XPath for the outCookie attribute.

Example 2: Making a JSON based Login REST API call to an APIC device

The JSON response of a successful login API call to an APIC device is as follows:

Response:

```
{
  "imdata" : [{
      "aaaLogin" : {
        "attributes" : {
          "token" :
             "GkZl5NLRZJl5+jqChouaZ9CYjgE58W/pMccR+LeXmdO0obG9NB
             Iwo1VBo7+YC1oiJL9mS6I9qh62BkX+Xddhe0JYrTmSG4JcKZ4t3
             bcP2Mxy3VBmgoJjwZ76ZOuf9V9AD6Xl83lyoR4bLBzqbSSU1R2N
             IgUotCGWjZt5JX6CJF0=",
          "refreshTimeoutSeconds" : "300",
          "lastName" : "Washington",
          "firstName" : "George"
        },
        "children" : [{
    ...
    ...
}
```

The token attribute is required to perform any other API calls on the device. Hence, you have to fetch the value for the token attribute.

As the output is a JSON response, find the JSONPath for the token attribute. The JSONPath is similar to XPath where we traverse the element from the parent element.

The JSONPath for the token attribute from the response is as follows:

```
JSONPath: $.imdata[0].aaaLogin.attributes.token
```

You have to define the output in the Generic API task as:

- **Output Name :** Cookie
- **Path :** $.imdata[0].aaaLogin.attributes.token

Example 3: Using XPath or JSONPath in-built Filtering Capability

Using the in-built filtering capability of XPath and JSONPath, you can define the XPath or JSONPath of an element and set some filtering criteria.

Let's take the following JSON response which displays a list of employee details:

Response:

```
[
  {
    'name': 'John',
    'address1': null,
    'empId': 10,
    'salary': 0.0
```

```
  },
  {
    'name': 'George',
    'address1': null,
    'empId': 11,
    'salary': 0.0
  },
  {
    'name': Frank',
    'address1': null,
    'empId': 12,
    'salary': 0.0
  }
]
```

If you want to fetch the ID of employee Frank, you can use the filtering criteria as in the following JSONPath expression:

```
$.[?(@.name=="Frank")].empId
```

You have to define the task output using the JSONPath expression as:

- **Output Name :** Employee_Id
- **Path :** $.[?(@.name=="Frank")].empId

Note If there are more than one employee with the name Frank, you will get the comma separated employee IDs as [13, 14, 20].

Similar to this example, you can use the XPath filtering capabilities to filter XML response and output the desired node or element value from the response.

Defining Variables

You can define variables in a task definition as follows:

```
@{INPUT VARIABLE}
```

For example, you can define the user name and employee name as: `@{Username}, @{Employee Name}`

Variables can be used in the following places:

- **API Path—**You can define variables in the API path which will form the URL for the API call, and these variables become the task input. The API path is evaluated during the task execution.
- **Request Content—**You can define variables in the request body content and these variables become the task input. The body content is evaluated during the task execution.
- **Output Path—**You can define variables in the XPath or JSONPath expression output, and these variables become the task input. The output path is evaluated during the task execution and it is used to find the required output from XML or JSON response.

These variables become the task input and the values for the variables can be input either as a mapped input or task input.

Sample: Defining Variables in an API Path

In this example, let's see how you can execute the getEmployeeDetails API (available at the following URL) to fetch employee details by passing run-time employee ID using the generic API task.

```
https://[IP]:[Port]/getEmployeeDetails/[Employee ID]
```

To execute the API using the generic API task, the employee ID must be set as a variable and the value must be provided at run-time. The IP address and port values are also required but can be static in nature.

You can define the API structure using the generic API task as follows:

- **IP Address :** xx.xx.xx.xx
- **Port :** 443
- **API Path :** /getEmployeeDetails/@{Employee_ID}

The @{Employee_ID} notation creates a task input named as Employee_ID. You can map the input to either workflow input or define the value statically as task input.

Sample: Defining Variables in a Request Content

In most of the HTTP API calls, a body content is sent as a payload. The payload contains the information required to perform the basic operation, such as, Post, Put, and so on.

Let's take an example of getting a session ID from Cisco UCS Manager which can further be used as a parameter to perform other operations.

The API path required to get session ID is /nuova. The request content that needs to be sent to the URL is:

```
<aaaLogin inName= [User Name] inPassword= [Password] />
```

The API structure defined using a generic API task to retrieve a session ID is as follows:

- **IP Address :** xx.xx.xx.xx
- **Port :** 443
- **API Path :** /nuova
- **Request Content:** <aaaLogin inName= @{User_Name} inPassword= @{Password.encrypt} />

Here, the User_Name and Password variables are the task input and the values can be provided either statically or as mapped input.

The .encrypt keyword is used to make sure that the generated task input type for the defined variable is password and the password will not be visible in the task or during the task execution.

By default, the task input type of the defined variable is generic text.

For devices managed in Cisco UCS Director, the request content can also be defined using the following keywords:

```
<aaaLogin inName= @{Account.username} inPassword= @{Account.password} />
```

Or

```
<aaaLogin inName= @{CredPolicy.username} inPassword= @{CredPolicy.password} />
```

The **@{Account.username}** and **@{Account.password}** keywords are used to retrieve user name and password information from an existing account of Cisco UCS Director. The generic API task fetches the information for the selected account internally and use the information during the API execution. These keywords don't appear as the task input in the next page, as the values are fetched internally from Cisco UCS Director.

The **@{CredPolicy.username}** and **@{CredPolicy.password}** keywords can be used when the user wants to provide credential information using a credential policy defined in Cisco UCS Director. These keywords can be used for devices which are not managed by Cisco UCS Director, but the device credential information is input from the credential policy in Cisco UCS Director. These keywords don't appear as the task input in the next page, as values are fetched internally from Cisco UCS Director.

Sample: Defining Variables in the Output Path

In certain situations, you may want to use variables in the XPath or the JSONPath of the element available in the response. For example, let's take the following JSON response of an API execution that displays the employee details.

Response:

```
[
  {
    'name': 'John',
    'address1': null,
    'empId': 10,
    'salary': 0.0
  },
  {
    'name': 'George',
    'address1': null,
    'empId': 11,
    'salary': 0.0
  },
  {
    'name': Frank',
    'address1': null,
    'empId': 12,
    'salary': 0.0
  }
]
```

To output the employee ID of an employee, the following JSONPath is used:

```
$.[?(@.name=="Frank")].empId
```

If you want to pass the employee name as a variable to get the employee name at run-time and to get the employee ID for the same as task output, you can define variables in the output path definition using the **@{Variable_Name}** notation.

The JSONPath to find employee ID for the employee name that is passed as a variable is as follows:

```
$.[?(@.name=="@{Employee_Name}")].empId
```

The **Employee_Name** variable is the task input, and it can get value as a task input or a mapped input. The variable value is evaluated at run-time during the task execution, and then the JSONPath is evaluated based on the JSON response. Finally, the evaluated JSONPath value is output as the task output.

In the same way, you can define the variable in the XPath definition while defining the output of the generic API task.

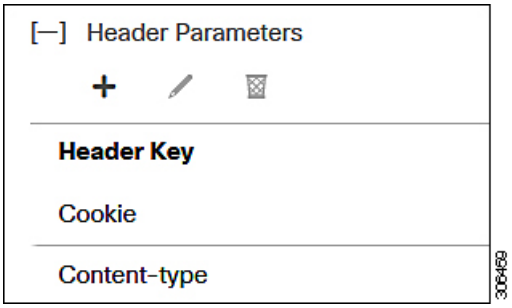
Defining Headers

Generic API task allows you to define the HTTP header that is required for HTTP/HTTPS based API execution. The header parameters are consumed as task input in the next page automatically, and the parameters get value statically or as a mapped input during run-time.

If you want to pass the header value with some portion of it as static and rest of the portion as a variable, use the dollar ($) concept available in Cisco UCS Director to define the value (with static and variable portion) for the header.

For example, in the following figure, there are two header parameters required for the API execution:

Figure 3: Task – Header Parameters



These header parameters are automatically set as task input, and the values for the parameters can be provided either statically or as a mapped input.

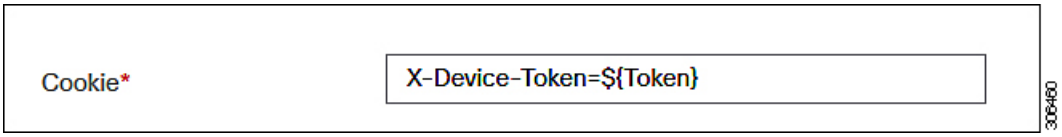
Say, if the value required for the Cookie parameter is as follows:

```
X-Device-Token=[TOKEN_VALUE]
```

Where, the X-Device-Token is a fixed part and TOKEN_VALUE is a value which can be a variable.

To define the value for the Cookie task input, use the dollar symbol as in the following figure:

Figure 4: Cookie Task Input



HTTP Basic Authentication

In certain situations, the API execution may require passing of authentication parameters using the HTTP basic authentication method.

HTTP Basic Authentication is a standard way to authenticate clients with server. The authentication information is passed using an Authorization header parameter that contains the word `Basic` followed by a space and a Base64-encoded string `username:password`.

Generic API task allows you to define basic authentication mechanism in one of the following ways:

- Allows you to provide the credential information (username and password) as task input. In this case, two task inputs are auto-generated (`Auth_Username` and `Auth_Password`) and you can provide the required values as task input or mapped input.
- Allows you to use the existing device credential or information in a credential policy for basic authentication parameter. In this case, the required information is fetched internally from Cisco UCS Director and no input is required from user.

Note Generic API task automatically adds the required header for HTTP Basic authentication (Authorization) with required formatted value while making the API call. You need not define this header manually.

Defining Rollback API

To define rollback API for the generic API task, perform the following:

1. Define the minimal required rollback API definitions, such as, URL, headers, API path, and request body.
2. Use @Variable notation to define variables which appears as task input for the task.
3. Use the @Variables notation in the rollback URL and rollback body content.
4. Use the keywords (such as, Account.username, Account.password, CredPolicy.username and CredPolicy.password) in the rollback URL and rollback body content.

Defining Rollback Workflow

To define rollback workflow for the generic API task, perform the following:

1. Choose a workflow that has to be triggered during the rollback of a task.
2. Configure the workflow inputs and provide information about the values to be passed to the workflow during rollback.
3. You can choose to take a workflow input value as task input. Value can be provided as task input or mapped input.
4. You can choose to provide the workflow input value using pre-defined keywords (such as, Account.username, Account.password, DeviceCred.username, and DeviceCred.password).
5. You can also choose to use the outputs defined in the task as input value for the rollback workflow.