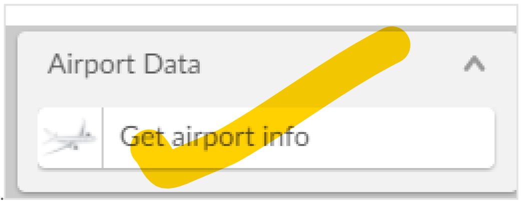
Define the operations

In the previous topic, you created a basic OpenAPI Specification structure for the Airport-Data API. Now you will add an operation to define an action for retrieving the airport information.

Summary

Operations are the functions calls available in the API, and are displayed as actions and events in the action group for this custom connector. Each operation you declare maps to an action in the action toolbox.



.

Note: You do not need to define every operation of the API within the OpenAPI Specification. Only define the operations you want to use.

Operations as paths and methods

Operations in the OpenAPI Specification are defined in two parts:

- The path to access the operation or resource on the host
- The HTTP method to use when requesting the resource, such as GET or POST

Each combination of path and method creates a unique operation, but not all combinations are necessarily supported by the API. Operations must be defined to match the API exactly: you cannot substitute an HTTP method or change the path.

In the Workflow Designer, the actions are listed in the action toolbox in the order you define them.

Define the operations

To define an operation, you define:

- The path to the resource
- The HTTP method to access the resource
- The operation details, including a description of the operation and what format it returns

Step 1: Add the paths object

All operations in an OpenAPI Specification are enclosed within the **paths** object. The **paths** object is usually added after the basic OpenAPI Specification structure, but before the closing brace at the end of the file. Make sure you include the opening and closing braces of the object.

Copy

```
{
  "swagger": "2.0",

  "info": {
    "version": "1.0.0",
    "title": "Airport Data"
    "description": "Retrieves location information based on airport codes"
  },
  "host": "www.airport-data.com",
  "basePath": "/api",
  "schemes": [
    "http"
  ],

  "produces": [
    "application/json"
  ],

  "paths" : {

  }

}
```

Step 2: Add the operation path

Within the **paths** object, the resource path to access the operation is added as another object. In this example, the resource path is **/ap_info.json**. Note that the braces of the path object completely enclose the resource path object.

Copy

```
"paths" : {
  "/ap_info.json": {

  }
}
```

Defining more than one path

Step 3: Add the HTTP method

Within the resource path object, the HTTP method to use for this operation is added as another object. In this example, the method is **get**. Notice that the braces of the resource path object completely enclose the HTTP method object.

Copy

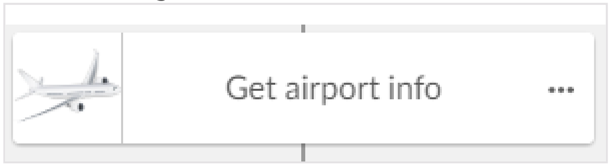
```
"paths" : {
  "/ap_info.json": {
    "get": {


    }
  }
}
```

Defining more than one HTTP method

Step 4: Add the operation details

Now that the operation path and method have been specified, add the details about this operation inside the HTTP method object:

- The **summary** is the name of the operation, and is used as the name of the action in the Workflow designer.



- The **description** is a brief explanation of what the operation does. This is for documentation purposes in the OpenAPI Specification only, and does not appear in your workflow.
- The **operationId** is a unique identifier for the operation within the OpenAPI Specification. This identifier must be unique for the OpenAPI Specification to be valid.
- The **produces** array defines the formats this API can return information in. It is included in case this operation overrides the global types provided in the basic OpenAPI Specification structure.

Copy

```
"paths" : {
  "/ap_info.json": {
    "get": {
      "summary": "Get airport info",
      "description": "Get airport info",
      "operationId": "getAirportInfo",
      "produces": [
        "application/json"
      ],

    }
  }
}
```

Note the comma after the closing square bracket of the **produces** array. This is required, because you will add more information to the HTTP method object in the next topic.

The OpenAPI Specification

The OpenAPI Specification now contains an operation that can be used as an action in the Workflow designer. Next you will add the parameters that the API needs to process the request.

Copy

```
{
  "swagger": "2.0",

  "info": {
    "version": "1.0.0",
    "title": "Airport Data"
    "description": "Retrieves location information based on airport codes"
  },

  "host": "www.airport-data.com",
  "basePath": "/api",
  "schemes": [
    "http"
  ],

  "produces": [
    "application/json"
  ],

  "paths" : {
    "/ap_info.json": {
      "get": {
        "summary": "Get airport info",
        "description": "Get airport info",
        "operationId": "getAirportInfo",
        "produces": [
          "application/json"
        ],

      }

    }

  }

}
```

Next steps

Define the parameters