Using the Methods and Resources

The following topics explain in general how to use the various methods and resources.

Trying a Method and Interpreting the Results

You can use the API Explorer to test the various methods. This topic explains the general process, with an explanation of the response the system returns. See the topics on each method type for specific method-related techniques.

The **Try It Out!** button for each method/resource interacts directly with the system. GET retrieves actual data, POST/PUT creates or modifies real resources, and DELETE removes real objects. You are making real configuration changes in the system, although the changes are not immediately deployed. To make the changes active, use the POST /operational/deploy resource to start a deployment job.

You can find the **Try It Out!** button after the **Response Message** section when you open a method/resource. For some method/resources, you must enter an object ID to test it. In this case, you typically need to first do a GET on the parent resource. For more information, see Finding the Object ID (objId) and Parent ID.

For POST/PUT, you also need to fill in the required values in the JSON model.

After you click **Try It Out!**, the API Explorer adds the results to the page after the button. The response includes the following sections:

Curl

The **curl** command used to make the call. For example, clicking **Try It Out!** on the GET /object/networks resource returns something like the following. The "v" element in the path changes with each new version of the API.

```
curl -X GET --header 'Accept: application/json'
'https://ftd.example.com/api/fdm/latest/object/networks'
```

Note This does not include the **Authorization: Bearer** header, which would be required in an API call from your client.

Request URL

The URL to issue from your client to make the request. For example, for GET /object/networks:

```
https://ftd.example.com/api/fdm/latest/object/networks
```

Response Body

The object that the system returns to your client. If a resource can include multiple objects (such as /object/network), you get a list of items on a GET request. POST/PUT/DELETE responses will be about a single object.

The specific content returned is based on the resource model. For example, GET /object/networks returns a list of objects, with each object looking something like the following (the initial indication of an items list is also shown). Note that the links/self value indicates the URL you would use to refer to this object; the object ID is included in the URL.

```
{
  "items": [
    {
      "version": "900f8558-7d19-11e7-bf7b-3dcaf0c58345",
      "name": "AIM_SERVERS-205.188.1.132",
      "description": null,
      "subType": "HOST",
      "value": "205.188.1.132",
      "isSystemDefined": true,
      "id": "900fac69-7d19-11e7-bf7b-d9417b20e59e",
      "type": "networkobject",
      "links": {
        "self": "https://ftd.example.com/api/fdm/latest/
object/networks/900fac69-7d19-11e7-bf7b-d9417b20e59e"
      }
    },
```

GET requests also include a paging section, which is explained in GET: Obtaining Data from the System, on page 3.

Response Code

The numeric HTTP status code returned for the HTTP call. These are the standard HTTP status codes, which you can find in RFCs or Wikipedia (such as https://en.wikipedia.org/wiki/List_of_HTTP_status_codes). For example, 200 (OK) indicates a successful GET/PUT/POST call, and 204 a successful DELETE call.

Response Headers

These are the packet headers in the HTTP response. For example, GET /object/networks might have headers such as the following:

```
{
  "date": "Thu, 10 Aug 2017 19:19:16 GMT",
  "content-encoding": "gzip",
```

```
  "x-content-type-options": "nosniff",
  "transfer-encoding": "chunked",
  "connection": "Keep-Alive",
  "vary": "Accept-Encoding",
  "x-xss-protection": "1; mode=block",
  "pragma": "no-cache",
  "server": "Apache",
  "x-frame-options": "SAMEORIGIN",
  "strict-transport-security": "max-age=31536000 ; includeSubDomains",
  "content-type": "application/json;charset=UTF-8",
  "cache-control": "no-cache, no-store, max-age=0, must-revalidate",
  "accept-ranges": "bytes",
  "keep-alive": "timeout=5, max=99",
  "expires": "0"
}
```


GET: Obtaining Data from the System

Use the GET method to read information from the device.

If a resource can contain multiple objects, you get a list of objects in the response. You can include query parameters on the URL to control the number of objects returned. The default is to return 10 objects from the start of the object list.

The following procedure explains the general approach for making GET calls in the API Explorer. Use the example code for your API client.

Procedure

Step 1 In API Explorer, open a GET method (first, open the group to see the methods and resources).

Step 2 If the method you want to use requires an object or parent ID in the URL, use a parent method to obtain the needed ID.

For example, GET /objects/networks/{objId} requires the ID of a specific object. Use the GET /objects/networks method to get a list of network objects, then look for the id value for the object you want to examine. Note that in this case, the information returned in the GET /object/networks call will be the same as what you see in GET /objects/network/{objId}. See Finding the Object ID (objId) and Parent ID.

Step 3 In the **Parameters** section, configure the following options:

- **objId**—The object ID is always required if it is needed in the URL. For example, 900fac69-7d19-11e7-bf7b-d9417b20e59e.
- **parentId**—The parent ID is equivalent to object ID, but is simply for a parent that is higher up in the hierarchy. For example, GET /policy/intrusion returns a list of intrusion policies, whereas GET /policy/intrusion/{parentId}/intrusionrules returns the rules defined within one of those policies. You would get the parent ID from GET /policy/intrusion.
- **offset**—For resources that support multiple objects, how far into the list to start returning objects. The default, 0, indicates the start of the list.
- **limit**—The maximum number of objects to return in the response. The default is 10. The maximum limit is 1000; if you enter an invalid value, it is automatically changed to 1000.

- **sort**—How to sort the objects returned in the response. The default sort is alphabetical by the **name** value. To change the sort, enter the name of the attribute within the resource to sort on. For example, you can use **sort=value** in network objects to sort on the value (that is, IP address) attribute. To sort in reverse order, include a minus sign, for example, **sort=-name**.
- **filter** (not available for all resources)—Return items that match the filter criteria only. The format of the filter value is {key}{operator}{value}, where the key is an attribute name and value is the string to filter on. There are no spaces between the items. The fields you can filter on are listed in the description of the **filter** parameter in the API Explorer; the fields differ per object. If an object supports filtering on multiple fields, you can include multiple values on the **filter** parameter, separated with a semi-colon (;). For example, you could filter on gid:1;sid:105 for GET /policy/intrusionpolicies/{parentId}/intrusionrules. The allowed operators are:
  - **:** for equal to. For example, **filter=name:Canada**.
  - **!** for not equal to. For example, **filter=name!Canada**.
  - **~** for similar to. For example, **filter=name~United**.
- **filter=fts~***string* (not available for all resources)—Return items that match the filter only. The **fts~** option is for full-text search. All attributes in the object are searched for the string. You can include a partial string; using the asterisk * as a wildcard to match one or more characters is optional. Do not include the following characters, they are not supported as part of the search string: ?~!{}<>:%. The following characters are ignored: ;#&.

  For example, you could find all network objects that have 10. as the first octet using GET /object/networks?filter=fts~10. Note that it takes 3-5 seconds to index newly-created or updated objects, so you need to pause before immediately doing a full-text search on new or changed objects.
- **filter=fetchZeroHitCount:**{**true** | **false**} (available for access rules only)—If you specify **includeHitCounts=true**, you can use this filter option to either include (**true**) or exclude (**false**) rules that have not been hit, that is, whose hit count is zero. The default is **true**.
- **includeHitCounts** (available for access rules only)—Whether to include hit count information for the rules in the policy. Specify **includeHitCounts=true** to get hit counts. Specify **false** (the default) to exclude hit counts. The hit count information is returned in the hitCount attribute in the returned object.
- **time_duration** (available for trending reports only)—How many seconds into the past the report should include. For example, 1800 returns a report for the past 30 minutes.

Note Whereas {objId} and {parentId} are part of the URL path, add the **offset**, **limit**, **sort**, **filter**, **includeHitCounts**, and **time_duration** parameters after a **?** character to the end of the URL.

Step 4 Click the **Try It Out!** button and examine the response.

For successful calls (return code 200), the response body includes one object or a list of objects, depending on the call you made. For information on the general structure and content of the response, see Trying a Method and Interpreting the Results, on page 1.

GET requests include a paging section. If there are more objects than were returned for the call, the **prev** and **next** values indicate how to get the previous or next set of objects. The **count** value indicates the overall number of objects. The **limit** value indicates how many items are returned in the response. The **offset** value indicates the starting position of the returned objects, with 0 indicating the start of the list.

```
"paging": {
```

```
    "prev": [],
    "next": [
      "https://ftd.example.com/api/fdm/latest/object/networks?limit=10&offset=10"
    ],
    "limit": 10,
    "offset": 0,
    "count": 22,
    "pages": 0
  }
```

POST: Create a New Object

Use the POST method to create a new object for a type of resource. For example, use POST to create a new network object.

The following procedure explains the general approach for making POST calls in the API Explorer. Use the example code for your API client.

Procedure

Step 1 In API Explorer, open a POST method (first, open the group to see the methods and resources).

Step 2 Click **Model** under the **Response Class** heading and read about the data types and values for the resource attributes.

Step 3 Under the **Parameters** heading, configure the following options if they are available:

- **parentId**—The ID of the parent object that will contain this object. For example, when adding an SSL rule, the ID of the SSL Decryption policy.
- **at**—For objects that reside in a parent that organizes the objects in an ordered list, such as SSL decryption policies, the location to insert the object. Use an integer to indicate the location, with 0 being the start of the list. The default is to insert the new object at the end of the list.

Note Whereas {objId} and {parentId} are part of the URL path, add the **at** parameter after a **?** character to the end of the URL.

Step 4 Also under the **Parameters** heading, click the JSON model shown in the **Data Type** > **Example Value** column for the **body** parameter.

Clicking in this box loads the JSON model into the Value column for the **body** parameter. For example, clicking the box for the POST /object/networks resource loads the following body:

```
{
  "name": "string",
  "description": "string",
  "subType": "HOST",
  "value": "string",
  "type": "networkobject"
}
```

Step 5 Fill in the required values for the **body** JSON object attributes.

For enum values, be certain to read the allowed values under **Response Class** > **Model**. For example, you can create a network object for a subnet (rather than a host) address by filling in the values and changing the default value for **subType**:

```
{
  "name": "new_network_object",
  "description": "A subnet object created using the REST API.",
  "subType": "NETWORK",
  "value": "10.100.10.0/24",
  "type": "networkobject"
}
```

Step 6 Click the **Try It Out!** button and examine the response.

Examine the **curl** command used to update the system. Note the additional headers. When you create your API client, you need to also include these header fields and values. For example, the **curl** command to create the sample object is the following. Note the Content-Type and Accept headers.

```
curl -X POST --header 'Content-Type: application/json' \
   --header 'Accept: application/json' -d '{ \
   "name": "new_network_object", \
   "description": "A subnet object created using the REST API.", \
   "subType": "NETWORK", \
   "value": "10.100.10.0/24", \
   "type": "networkobject" \
 }' 'https://ftd.example.com/api/fdm/latest/object/networks'
```

For successful calls (return code 200), the response body includes the complete object that you created, including other system-generated values such as **version** and **id**. The version and ID values are especially important, as you need them if you subsequently use PUT to change the object. For information on the general structure and content of the response, see Trying a Method and Interpreting the Results, on page 1.

The response body also includes the **links/self** value, which is the URL for the object you created. For example, the following is the response body for the sample object.

```
{
  "version": "f6d8da48-7ed5-11e7-9bfd-d96183b5f5f1",
  "name": "new_network_object",
  "description": "A subnet object created using the REST API.",
  "subType": "NETWORK",
  "value": "10.100.10.0/24",
  "isSystemDefined": false,
  "id": "f6d8da49-7ed5-11e7-9bfd-27136f5686ad",
  "type": "networkobject",
  "links": {
    "self": "https://ftd.example.com/api/fdm/latest/object/networks/
f6d8da49-7ed5-11e7-9bfd-27136f5686ad"
  }
}
```

PUT: Modify an Existing Object

Use the PUT method to change the attributes of an existing object. For example, use PUT to modify the address contained within an existing network object without changing the object's ID.

The PUT method replaces the entire object. You cannot simply change one attribute. Thus, you must ensure that your JSON object includes the old values that you want to preserve.

The following procedure explains the general approach for making PUT calls in the API Explorer. Use the example code for your API client.

Before you begin

Use the GET method for the parent resource to obtain a copy of the existing state of the object, as described in GET: Obtaining Data from the System, on page 3.

You must have the correct values for at least the following parameters, as well as any user-supplied values that you do not want to change.

- **version**
- **id**

Procedure

Step 1 In API Explorer, open a PUT method (first, open the group to see the methods and resources).

Step 2 Under the **Parameters** heading, configure the following options:

- **objId**—The **id** value for the object. For example, 900fac69-7d19-11e7-bf7b-d9417b20e59e.
- **parentId**—For objects that reside within another object, the ID of the parent object that will contain this object. For example, when modifying an SSL rule, the ID of the SSL Decryption policy.
- **at**—For objects that reside in a parent that organizes the objects in an ordered list, such as SSL decryption policies, the location to insert the object. Use an integer to indicate the location, with 0 being the start of the list. The default is to insert the object at the end of the list.

Note Whereas {objId} and {parentId} are part of the URL path, add the **at** parameter after a **?** character to the end of the URL.

Step 3 Also under the **Parameters** heading, click the JSON model shown in the **Data Type** > **Example Value** column for the **body** parameter.

Clicking in this box loads the JSON model into the Value column for the **body** parameter. For example, clicking the box for the PUT /object/networks resource loads the following body. Note that this is slightly different from the POST version for the same resource: the PUT body includes the **version** attribute.

```
{
  "version": "string",
  "name": "string",
  "description": "string",
  "subType": "HOST",
  "value": "string",
```

```
  "type": "networkobject"
}
```

Step 4 Fill in the required values for the **body** JSON object attributes.

Be certain to replicate old values that you do not want to change.

For enum values, be certain to read the allowed values under **Response Class** > **Model**. Repeat the old value unless you are changing the object to a different subtype. For example, the default PUT model for a network object has HOST for **subType**, but if you are changing a subnet object, make sure you change **subType** to NETWORK.

For example, to update the subnet IP address in a network object, repeat all the old values for all attributes except **value**. Enter the new subnet address in **value**.

```
{
  "version": "f6d8da48-7ed5-11e7-9bfd-d96183b5f5f1",
  "name": "new_network_object",
  "description": "A subnet object created using the REST API.",
  "subType": "NETWORK",
  "value": "10.100.11.0/24",
  "type": "networkobject",
}
```

Step 5 Click the **Try It Out!** button and examine the response.

Examine the **curl** command used to update the system. Note the additional headers. When you create your API client, you need to also include these header fields and values. For example, the **curl** command to update the sample object is the following. Note the Content-Type and Accept headers.

```
curl -X PUT --header 'Content-Type: application/json' \
--header 'Accept: application/json' -d '{ \
   "version": "f6d8da48-7ed5-11e7-9bfd-d96183b5f5f1", \
   "name": "new_network_object", \
   "description": "A subnet object created using the REST API.", \
   "subType": "NETWORK", \
   "value": "10.100.11.0/24", \
   "type": "networkobject" \
 }' 'https://ftd.example.com/api/fdm/latest/object/
networks/f6d8da49-7ed5-11e7-9bfd-27136f5686ad'
```

For successful calls (return code 200), the response body includes the complete object that you updated. Note that the version value changes, but the object ID (and thus the link/self) stays the same. The version changes every time you modify an object. For information on the general structure and content of the response, see Trying a Method and Interpreting the Results, on page 1.

Note If you did not make any changes to the object, that is, the object being updated is the same as its previous version, the system does not process the request and instead and sends back a 204 code saying that nothing has changed for that resource.

For example, the following is the response body for updating the sample object.

```
{
  "version": "96f5f3cc-7ede-11e7-9bfd-9b7d8a92863f",
  "name": "new_network_object",
  "description": "A subnet object created using the REST API.",
```

```
  "subType": "NETWORK",
  "value": "10.100.11.0/24",
  "isSystemDefined": false,
  "id": "f6d8da49-7ed5-11e7-9bfd-27136f5686ad",
  "type": "networkobject",
  "links": {
    "self": "https://ftd.example.com/api/fdm/latest/object/networks/
f6d8da49-7ed5-11e7-9bfd-27136f5686ad"
  }
}
```

DELETE: Remove a User-Created Object

Use the DELETE method to remove an object that you, or another user, created. For example, use DELETE to remove a network object that you no longer use.

You cannot delete system-defined objects or objects that are required to exist.

You also cannot delete an object that is currently being used by another object, such as a network object that is used in an access rule. For in-use objects, first modify all objects that use it, then delete the object.

The following procedure explains the general approach for making DELETE calls in the API Explorer. Use the example code for your API client.

Before you begin

Use the GET method for the parent resource to obtain a copy of the existing state of the object, as described in GET: Obtaining Data from the System, on page 3.

You must have the object ID (the **id** value) to delete the object.

Procedure

Step 1 In API Explorer, open a DELETE method (first, open the group to see the methods and resources).

Step 2 Under the **Parameters** heading, enter the **id** value for the object in the **objId** field. For example, f6d8da49-7ed5-11e7-9bfd-27136f5686ad.

If the object resides within a container, you also need to enter the ID of the parent object into the **parentId** field.

Step 3 Click the **Try It Out!** button and examine the response.

Examine the **curl** command used to delete the object from the system. Note the additional headers. When you create your API client, you need to also include these header fields and values. For example, the **curl** command to delete the sample object is the following. Note the Accept header.

```
curl -X DELETE --header 'Accept: application/json'
'https://ftd.example.com/api/fdm/latest/object/
networks/f6d8da49-7ed5-11e7-9bfd-27136f5686ad'
```

For successful calls (return code 204 “No Content”), you get an empty response body. This is the expected result.