

FMC API Programming with Python Lab Manual

May 20, 2020

Version 0.1

**Cisco Systems, Inc.  
Corporate Headquarters  
170 West Tasman Drive  
San Jose, CA 95134-1706 USA**[**http://www.cisco.com**](http://www.cisco.com) **Tel: 408 526-4000 Toll Free: 800 553-NETS (6387)  
Fax: 408 526-4100**

Contents

[Contents 2](#_Toc40869415)

[Lists of Tables and Figures 3](#_Toc40869416)

[About FMC API Programming with Python 4](#_Toc40869417)

[Approvals 4](#_Toc40869418)

[Revision History 4](#_Toc40869419)

[1 Module 1: Introduction and Lab Environment 5](#_Toc40869420)

[1.1 Course Introduction 5](#_Toc40869421)

[1.2 Course Prerequisites 5](#_Toc40869422)

[1.3 Lab Environment 5](#_Toc40869423)

[1.4 FMC API Explorer 5](#_Toc40869424)

[1.5 REST API Basics 5](#_Toc40869425)

[2 Module 2: JSON 6](#_Toc40869426)

[3 Module 3: API Explorer In Depth 7](#_Toc40869427)

[4 Module 4: Postman 8](#_Toc40869428)

[5 Module 5: Introduction to Python 9](#_Toc40869429)

[6 Module 6: API Requests with Python 10](#_Toc40869430)

[7 Module 7: Building a Code Framework 11](#_Toc40869431)

[8 Module 8: AC Policy and AC Rule GET Operations 12](#_Toc40869432)

[9 Module 9: POSTing an AC Policy 13](#_Toc40869433)

[10 Module 10: GETting and POSTing Objects 14](#_Toc40869434)

[11 Module 11: POSTing AC Rules 15](#_Toc40869435)

[12 Module 12: Converting Rules from Text 16](#_Toc40869436)

[13 Module 13: AC Rule Advanced Operations 17](#_Toc40869437)

[14 Module 14: Deployment 18](#_Toc40869438)

[15 Appendix A: Acronym Listing 19](#_Toc40869439)

[Trademarks and Disclaimers 20](#_Toc40869440)

Lists of Tables and Figures

**No table of figures entries found.**

**No table of figures entries found.**

About FMC API Programming with Python Lab Manual

|  |  |
| --- | --- |
| Author | Richard Clendenning |
| Change Authority | Cisco Systems Advanced Services, Security & Collaboration IDT, Implementation Americas |

Approvals

|  |  |
| --- | --- |
| Area of Responsibility | Name or User ID |
| Document Owner | Richard Clendenning |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Name or User ID | Comments |
| 0.1 | June 6, 2020 | Richard Clendenning | Document creation |
|  |  |  |  |

# Lab 1: Lab Environment

## Introduction

In this lab you will log into the FMC GUI and the API Explorer.

## FMC GUI Login

Please perform this step:

1. Log into the FMC GUI with your credentials as provided in the table below.

|  |  |
| --- | --- |
| Command | Description |
| FMC URL | https://10.2.3.4 |
| FMC username | First letter of first name and entire last name  E.g., for John Smith, the username is jsmith |
| FMC password | Cisco123 |

Table 1 – FMC login information

## Enabling the FMC REST API

In the FMC, the REST API is enabled by default. However, if you are intending to use the REST API, you should confirm that it is enabled. Please perform the following steps.

1. Navigate to System>Configuration>REST API Preferences>Enable REST API
2. If the “Enable REST API” checkbox is unchecked, check it.
3. Click “Save”. A "Save Successful" dialog will display when the REST API is enabled.

## Introduction to the FMC API Explorer

Please perform these steps:

1. Log into the FMC API Explorer with your credentials as provided in the table below.

|  |  |
| --- | --- |
| Command | Description |
| API Explorer URL | https://10.2.3.4/api/api-explorer |
| API Explorer username | First letter of first name and entire last name, then -api  E.g., for John Smith, the username is jsmith-api |
| FMC password | Cisco123 |

Table 1 – API Explorer login information

Note that while you can log into the API Explorer using the FMC GUI credentials, it is recommended (both in this course and in general practice) to use FMC GUI accounts exclusively for FMC GUI access and API Explorer accounts exclusively for API Explorer access. This allows one user to be logged into both at the same time and provides better accountability.

On login, you should see the Cisco Firepower Management Center Open API Specification as shown in the figure below.

A screenshot of a cell phone

Description automatically generated

Figure 1 – FMC OpenAPI Specification

1. Click Legacy Explorer at the top right to see the look of API Explorer prior to implementing the Open API Specification.
2. Click OpenAPI Docs at the top right to again see the Open API Specification. We will work exclusively with the Open API Specification going forward.

## Related Links

API Explorer has several helpful links. Please perform these steps:

1. Right-click on the REST API Quick Start Guide link and select Open Link in New Tab.
2. Click on the most recent version of the Firepower Management Center REST API Quick Start Guide.
3. In the Guide’s Table of Contents, click on the “About the API Explorer” link.
4. Skim this section of the Guide, which has information about the Legacy API Explorer and on obtaining sample code with CodeGen. Note that in the Legacy API Explorer you can obtain code snippets in Python or Perl directly from the web interface. Alternatively, you can download CodeGen and use it to produce code snippets from the FMC OpenAPI specification.
5. In the left navigation bar, note the links to the sections on Connecting with a Client and Objects in the REST API. We will cover these topics in this course, but these sections are a great reference.

## GETting Access Policies

In this exercise we will retrieve the Access Policies that are on the FMC.

The API Explorer interface has expandable categories for accessing endpoints.

Perform the following steps.

1. In API Explorer, click on the Policy category to expand it.

A screenshot of a cell phone

Description automatically generated

Figure 2 API Explorer Categories

1. Click on the first GET under the expanded Policy category.

A screenshot of a cell phone

Description automatically generated

Figure 2 GET for access policies

1. Click the “Try it out” button at the top right.
2. Click the Execute button.
3. Note how the API Explorer provides two ways that you could format this request yourself, using the Curl utility or a browser.

A screenshot of a cell phone

Description automatically generated

Figure 2 Curl GET request and Request URL

1. Note the server response code and the response body, which provides the list of policies in JSON format.

A screenshot of a cell phone

Description automatically generated

Figure GET response for access policies

In the above example output, there are two policies, named “ChildPolicy” and “ParentPolicy”.

# Module 2: JSON

The scenario for this lab exercise is as follows:

You have made a request to an FMC REST API for the Access Control rules that are in a certain policy. In the body of the response, the API returns a JSON object with one key-value pair. The key is “items” and the value is an array of two Access Control rules.

Each Access Control rule is a JSON object that has two key-value pairs as follows:

|  |  |  |
| --- | --- | --- |
| Item # | Key | Value |
| 1 | “name” | “Allow DNS” |
| “log” | false |
| 2 | “name” | “Allow SSH” |
| “log” | true |

Table 1: Data for Module 2 JSON exercise

Please perform this step:

1. Write the JSON that represents the data returned by the FMC API.

# Module 3: API Explorer In Depth

The API Explorer optional parameters are explained, as well as Curl and other aspects of API Explorer.

But most importantly, I want to get all AC policies, then get a particular policy, then get all the rules for that policy, then get a particular rule for a policy.

I also want to explore objects here.

## Requesting the Access Policies

Perform the following steps:

1. In Module 1, Section 1.6, the access policies were retrieved. Repeat those steps to again retrieve the list of access policies.
2. Copy the id of a particular policy from the response. We will use this in the next section.

## Requesting a Particular Access Policy

A request for all the access policies only returns the name and id of each access policy. To get more details about a particular policy, we must request only that policy.

Perform the following steps:

1. In API Explorer, under Policy, search for the following endpoint and expand it.

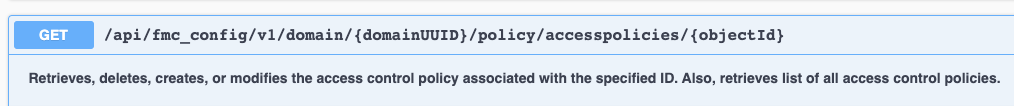


Figure GET request for a particular access policy

1. Click Try it out at the right.
2. In the field for “Identifier for access control policy”, paste in the access control policy id that you copied in the previous section.
3. Click Execute.
4. In the response object, there are these keys:
   1. metadata
   2. type
   3. links
   4. rules
   5. defaultAction
   6. name
   7. description
   8. id

Often, the information we want about a policy is the rules. As you can see, the response does not provide any information about the rules apart from a link which is the endpoint for a request that will return the policy’s rules.

## Requesting an Access Policy’s Rules

To request a list of the rules in an Access Control Policy, follow these steps:

1. Under Policy, expand the GET to retrieve a policy’s access rules.

A screenshot of a cell phone

Description automatically generated

Figure GET request for the access rules of a policy

1. At the right, click Try it out.
2. The container id is the id of the access policy whose rules we are requesting. Copy any access policy id obtained when we requested the list of all policies, and paste it into the containerUUID field.
3. Click Execute.
4. The server response should be a 200 and the Response body should provide a list of access rules that are in the policy.

# Module 4: Postman

In this module we will create a collection and several requests in Postman to become familiar with interacting with the Postman API.

## Creating a Request

Perform the following steps:

1. Download and install Postman.
2. Open Postman.
3. Click on New Collection at the left.
4. Name it FMC Access Policy. No description is needed.
5. Click Create.
6. In the left navigation bar, ensure that Collections is selected at the top, and find the FMC Access Policy collection.
7. Click the ellipsis for the FMC Access Policy collection and select Add Request.
8. Name the request “Get Access Token”, and click Save to FMC Access Policy at the bottom.

## Requesting the Access Token

Perform the following steps:

1. In the left navigation bar, expand the FMC Access Policy collection.
2. Click on the GET Access Token request to edit it.
3. By default, the method is GET. Change the method to POST.
4. Set the URL as follows:

**https://<management\_center\_IP\_or\_name>/api/fmc\_platform/v1/auth/generatetoken**

1. Below the URL, click Authorization.
2. Under Type, select Basic Auth.
3. Enter your api username.
4. Enter your api password.
5. At the right, click Save.
6. At the right, click Send.
7. A successful response has a status code of 204 No Content. The response has no body, but the access token and the refresh token are in the headers of the response. Click on Headers.

A screenshot of a cell phone

Description automatically generated

Figure GET response for access policies

1. The above diagram shows the X-auth-access-token. You will need this token when making requests in the upcoming steps.
2. Note that the refresh token is provided below the access token (X-auth-refresh-token). We will not use it in this exercise, but you should be aware that a request for a token provides both the access token and the refresh token.

## Requesting the Access Policies

Perform the following steps:

1. Add another request to the FMC Access Policy collection, and name it GET Access Policies.
2. Click on the request to edit it, and change the method to GET.
3. In API Explorer, find the Request URL for requesting the list of Access Policies, and copy it as the URL in the Postman request.
4. Click the Authorization tab.
5. Under type, select No Auth. We will authenticate through a header and not through one of the Postman authorization types.
6. Click the Headers tab.
7. In the Key field, enter “X-auth-access-token”.
8. For the value, enter the access token that was obtained in the previous section.
9. Click Save, and then Send.
10. A successful response has a status of 200 OK. If you receive an error that the token is invalid, try re-sending the token request and use the new token in the request for access policies.
11. Identify the names of several policies in the response, to ensure you understand how to interpret the output.

## Requesting a Particular Access Policy

A request for all the access policies only returns the name and id of each access policy. To get more details about a particular policy, we must request only that policy.

Perform the following steps:

1. Add another request to the FMC Access Policy collection, and name it GET Access Policy.
2. Method: GET
3. In API Explorer, find the Request URL for requesting the list of rules in an access policy. Refer to Section 3 as needed. Copy and paste the URL into the URL field.
4. Configure authorization for the request as described in section 4.3, steps 6-10.
5. Click Save, and then Send.
6. A successful response has a status of 200 OK. If you receive an error that the token is invalid, try re-sending the token request and use the new token in the request for access policies.
7. Identify the names of several rules in the response.

# Module 5: Introduction to Python

## VS Code: Installing the Python Extension

Visual Studio Code (VS Code) is a popular environment for Python development. To set up this environment, perform the following steps:

1. Create a folder called Development.
2. Open Visual Studio Code.
3. In Visual Studio Code, open the Development folder.
4. At the far left, there are six icons:
   1. Explorer
   2. Search
   3. Source Control
   4. Run
   5. Extensions

Click on the Extensions icon.

1. In the Search Extensions in Marketplace text box, type Python.
2. The first extension in the results should be a Python extension that provides linting, debugging, and other features and has over 20 million downloads. Click the Install button for this extension.

A screenshot of a cell phone

Description automatically generated

Figure The Python extension

## Running a Python Program

1. Click the Explorer icon at the top right.
2. Click the Development folder you created.
3. Click the new folder icon and create a new folder named Mod5.
4. In the Mod5 folder, create a new file named hello\_world.py.
5. Double-click the file to edit it.
6. Enter this line: print(“Hello, world!”)
7. Save the file.
8. Press Ctrl-~ to open a terminal in VS Code.
9. In the terminal window, run the program by typing python hello\_world.py

The figure below shows the program in the top half, and in the bottom have the terminal and the execution of the program.

A screenshot of a cell phone

Description automatically generated

Figure Initial Python program

## Learning Python from w3schools.com

### Python Variables, Data Types, Strings, and Operators

1. Execute the examples in the table below from the w3shools.com/python web site. The first example is located at <https://www.w3schools.com/python/python_variables.asp>.

|  |  |  |
| --- | --- | --- |
| Page | Section | Examples to Try |
| Python Variables | Creating Variables | All three “Try it Yourself” |
| Python Variables | Output Variables | All three “Try it Yourself” |
| Python Data Types | Getting the Data Type | The one “Try it Yourself” |
| Python Strings | String Methods | Instead of “Try it Yourself”, demonstrate these methods with your own code:   * isalpha() * isdigit() * split() * upper()   For example, the first one, isalpha(). Start with any string:  a = “Hello, World!” print a.isalpha()  Since some characters are not letters in the alphabet (the comma, the space, and the !), it should print False.  Now remove those characters:  a = "Helloworld" print a.isalpha()  Now, the result should be True |
| Python Operators | Python Arithmetic Operators | Try the Multiplication |
| Python Operators | Python Assignment Operators | Try the -= operator |

Table 1: Python variables, data types, strings, and operators examples from w3schools.com

### Python Lists, Dictionaries, Ifs, and For Loops

1. Execute the examples in the table below from the w3shools.com/python web site.

|  |  |  |
| --- | --- | --- |
| Page | Section | Examples to Try |
| Python Lists | Access Items | Try the one example |
| Python Dictionaries | Change Values | Try the one example |
| Python Dictionaries | Loop Through a Dictionary | Try all four examples |
| Python If … Else | Else | Try both examples |
| Python For Loops | Python For Loops | Try the one example |
| Python For Loops | The break statement | Try both examples |
| Python For Loops | The continue statement | Try the one example |

Table 1: Python lists, dictionaries, ifs, and for loop examples from w3schools.com

### Python Functions and JSON Conversions

1. Execute the examples in the table below from the w3shools.com/python web site. The first example is located at <https://www.w3schools.com/python/python_variables.asp>.

|  |  |  |
| --- | --- | --- |
| Page | Section | Examples to Try |
| Python Functions | Return Values | Try the one example |
| Python JSON | Parse JSON - Convert from JSON to Python | Try the one example |
| Python JSON | Convert from Python to JSON | Try all three examples |

Table 1: Python functions and JSON conversion examples from w3schools.com

## Installing Python Requests

Perform this step:

1. Use pip to install requests with this command:

pip install requests

# Module 6: API Requests with Python

In this module we will use Python to request an authorization token and to issue a GET for Access Policies.

Perform this step:

1. Create a Mod6 folder inside the Development folder.

## Obtaining Sample Code

1. In API Explorer, click on Legacy Explorer.
2. In the left menu, click on Policy.
3. For the top endpoint, click on GET.
4. At the right, click on the Export operation in.. button and select Python script.
5. Copy the script.
6. In VS Code, inside the Mod6 folder you created, create a new file called get\_ac\_policies.py.
7. Paste the Python code into the get\_ac\_policies.py file.
8. Save the file.

## Configuring the Program

All the lines of this program are explained in Module 6 of the Course Manual. You may want to review the code there before proceeding.

To configure the program with the correct FMC address and API username and password, complete the following steps:

1. Verify that the server variable assignment below the import statements contains the correct IP or hostname for your FMC. If it does not, supply the correct FMC address.
2. Enter your api username where the variable username is assigned.
3. Enter your api password where the variable password is assigned.
4. Save the file.

## Turning Off SSL Verification

1. Read the notes about ssl verification which are found around lines 24 – 30 of the script and are copied below.

|  |
| --- |
| Item # |
| # 2 ways of making a REST call are provided:  # One with "SSL verification turned off" and the other with "SSL verification turned on".  # The one with "SSL verification turned off" is commented out. If you like to use that then  # uncomment the line where verify=False and comment the line with =verify='/path/to/ssl\_certificate'  # REST call with SSL verification turned off:  # r = requests.post(auth\_url, headers=headers, auth=requests.auth.HTTPBasicAuth(username,password), verify=False)  # REST call with SSL verification turned on: Download SSL certificates from your FMC first and provide its path for verification. |
|
|

Table 1: Comment about ssl verification

1. We will be turning off ssl verification. Follow the instructions in the comment to:
   1. Uncomment the line that ends “verify=False”
   2. Comment the line that ends “verify='/path/to/ssl\_certificate'
2. Save the file.

## Executing the Program

1. Execute the program at the command line:

**$python get\_ac\_policies.py**

You should receive an InsecureRequestWarning, followed by the list of access policies in JSON.

## Executing the Program with Parameters

The program provides for supplying the username and password as parameters. We will test that now.

1. Change the username and password in the program to be incorrect. For example, change them both to “a”.
2. Save the file.
3. Execute the program again, but this time supply the username and password as parameters. For example:

**$python get\_ac\_policies.py rclenden C1sco.123**

You should receive an InsecureRequestWarning, followed by the list of access policies in JSON, the same as before.

1. Edit the program to change the username and password back to the valid credentails.

## Eliminating the InsecureRequestWarnings

The InsecureRequestWarning is provided by Python to caution you that you are attempting a secure connection to a site whose certificate cannot be verified. Generally, it is important to verify the identity of a server to which you are connecting, but for these labs the warning is unnecessary and we will eliminate it.

The warning is shown in the table below.

|  |
| --- |
| InsecureRequestWarning |
| /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/urllib3/connectionpool.py:979: InsecureRequestWarning: Unverified HTTPS request is being made to host '10.81.127.36'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings |
|
|

Table 1: InsecureRequestWarning notification

As you can see, the warning is produced by line 979 of a file called connectionpool.py (for Python version 3.8). We will comment out the code that produces this warning.

1. Using a text editor, open the connectionpool.py file on your machine. Follow the path that is given in your warning message to locate the file on your machine.
2. Comment out the lines that begin with if not conn.is\_verified:

The result should look like the lines in the table below.

|  |
| --- |
| InsecureRequestWarning |
| # if not conn.is\_verified:  # warnings.warn(  # (  # "Unverified HTTPS request is being made to host '%s'. "  # "Adding certificate verification is strongly advised. See: "  # "https://urllib3.readthedocs.io/en/latest/advanced-usage.html"  # "#ssl-warnings" % conn.host  # ),  # InsecureRequestWarning,  # ) |
|
|

Table 1: Python code that produces a warning when an insecure connection is attempted

1. Save the file.
2. Execute the program again. You should see the policies but no InsecureRequestWarning.

## Requesting Access Control Rules

To create a request for an access policy’s rules, we could use code provided by API Explorer, as we did in the previous section to obtain a list of access policies. However, we will instead modify the get\_ac\_policies.py script to arrive at a script that requests a policy’s rules.

1. Copy the get\_ac\_policies.py file, and name the copy “get\_ac\_rules.py”.
2. Read through the new get\_ac\_rules.py script and consider what needs to change to request the access rules of a policy.
3. What must change is the value of the variable api\_path. Review section 3.3 to see the endpoint for requesting a policy’s rules. Select the id of any access policy, and use it as you change the value of the api\_path variable in the program so that it has the endpoint path for requesting the policies access rules.
4. Save the file.
5. Execute the program. You should see the listing of that access policy’s rules.

# Module 7: Building a Code Framework

## Making a re-usable module

Perform these steps:

1. Create a Mod7 folder inside the Development folder.
2. Create a Mod7\_1 folder inside the Mod7 folder.

### Creating a get\_object() function

Perform the following steps:

1. Copy get\_ac\_policies.py, paste it into the Mod7\_1 folder, and name the copy get\_7\_1\_v1.py.
2. Change the code in get\_7\_1\_v1.py so that it is a function named get\_object that receives two parameters:
   1. fmc\_info
   2. endpoint
3. Remove all comments from get.py.
4. The fmc\_info parameter will be a dictionary with these keys:
   1. server
   2. username
   3. password

Change the code in the function so the assignments for server, username, and password are obtained from the fmc\_info parameter.

1. Change the assignment of the url variable to use the endpoint variable. Note that the endpoint argument will only have the last part of the path, such as policy/accesspolicies.
2. Add json\_resp = None above the try block.
3. Remove this from the try block:

print(json.dumps(json\_resp,sort\_keys=True,indent=4, separators=(',', ': ')))

1. At the end of the function, return the JSON response from the server.

Try to perform the above steps without looking at the solution. Here are some hints if you are struggling:

To create a function:

def get\_object(fmc\_info, endpoint):

and then indent all lines after this line, with four spaces.

To assign the server, username, and password:

server = fmc\_info['server']

username = fmc\_info['username']

password = fmc\_info['password']

To construct the url:

path\_base = "/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/"

url = server + path\_base + endpoint

### Creating a Main File

1. Start a new file that will use the get\_object() function. Create the file inside the Mod7\_1 folder and name it main\_7\_1\_v1.py.
2. Copy the following code into the main\_7\_1\_v1.py file.

import get\_7\_1\_v1 as get

fmc\_info = {

'server': 'https://10.81.127.36',

'username': 'api',

'password': 'superpass'

}

endpoint = 'policy/accesspolicies'

print (get.get\_object(fmc\_info, endpoint))

1. Execute the main\_7\_1\_v1.py program.
2. Verify that the program prints the list of access policies.

### Expanding the Main File

1. Copy the main\_7\_1\_v1.py program as main\_7\_1\_v2.py.
2. To demonstrate the reusability of the get\_object() function, add two more lines to the main\_7\_1\_v2.py program, to print the access rules by calling the get\_object() function with a different endpoint. See the lines below.

import get\_7\_1\_v1 as get

fmc\_info = {

'server': 'https://10.81.127.36',

'username': 'api',

'password': 'superpass'

}

endpoint = 'policy/accesspolicies'

print (get.get\_object(fmc\_info, endpoint))

endpoint = 'policy/accesspolicies/005056A4-5126-0ed3-0000-042949673459/accessrules'

print (get.get\_object(fmc\_info, endpoint))

1. **IMPORTANT: Replace the UUID in the second endpoint assignment above with the UUID of one of the access policies in the FMC you are working with.**
2. Run the main\_7\_1\_v2.py program.
3. Verify that the program prints all access policies, and then prints the access rules for the policy that you specified.

This exercise shows that the get\_object(fmc\_info, endpoint) code is re-usable.

## Parsing the Response

Perform these steps:

1. Create a Mod7\_2 folder inside the Mod7 folder.
2. Copy the files as indicated in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| From Folder | File | To Folder | File |
| Mod7\_1 | get\_7\_1\_v1.py | Mod7\_2 | get\_7\_2\_v1.py |
| Mod7\_1 | main\_7\_1\_v2.py | Mod7\_2 | main\_7\_2\_v1.py |

Table 1: Files for Module 7.2

### A Wrapper for Name and ID

The get\_object() function returns the response, but we really want just the name and id of each policy. In the future we could possibly need other parts of the response, so we’ll leave get\_object() alone and create a wrapper.

Perform these steps:

1. In get\_7\_2\_v1.py add this function for returning the names and IDs of the items:

def get\_names\_and\_ids(fmc\_info, endpoint):

full\_response = get\_object(fmc\_info, endpoint)

items = full\_response['items']

item\_list = []

for item in items:

item\_list.append({'name': item['name'], 'id': item['id']})

return item\_list

1. In get\_object, we want to return json\_resp, so replace the last line of the file with this line:

return json\_resp

1. Change the first line of main\_7\_2\_v1 to use the new function:

from get\_7\_2\_v1 import get\_names\_and\_ids

1. Replace the function calls in main\_7\_2\_v1 to use the new function:

endpoint = 'policy/accesspolicies'

print (get\_names\_and\_ids(fmc\_info, endpoint))

endpoint = 'policy/accesspolicies/005056A4-5126-0ed3-0000-042949673459/accessrules'

print (get\_names\_and\_ids(fmc\_info, endpoint))

1. Run the main\_7\_2\_v1.py program and verify that the output provides json for just name and id. Example output for just the policies is show below:

[{'name': 'Child2Policy', 'id': '005056A4-5126-0ed3-0000-051539612071'}, {'name': 'ChildPolicy', 'id': '005056A4-5126-0ed3-0000-025769803780'}, {'name': 'ParentPolicy', 'id': '005056A4-5126-0ed3-0000-042949673459'}]

### Simplifying the Output

The output is still json and is not pretty. We can print it in an easier to read format by using a for loop.

Perform these steps:

1. Copy main\_7\_2\_v1.py as main\_7\_2\_v2.py.
2. In main\_7\_2\_v2.py, replace the endpoint assignments and print statements with the following code:

endpoint = 'policy/accesspolicies'

item\_list = get\_names\_and\_ids(fmc\_info, endpoint)

for item in item\_list:

print(f'{item["name"]} {item["id"]}')

endpoint = 'policy/accesspolicies/005056A4-5126-0ed3-0000-042949673459/accessrules'

item\_list = get\_names\_and\_ids(fmc\_info, endpoint)

for item in item\_list:

print(f'{item["name"]} {item["id"]}')

1. Run main\_7\_2\_v2.py and verify that the output provides an easy to read listing of names and IDs of policies as shown below.

RCLENDEN-M-D6UH:Development rclenden$ python3 Mod7/Mod7\_2/main\_7\_2\_v2.py

GET successful. Response data -->

Child2Policy 005056A4-5126-0ed3-0000-051539612071

ChildPolicy 005056A4-5126-0ed3-0000-025769803780

ParentPolicy 005056A4-5126-0ed3-0000-042949673459

RCLENDEN-M-D6UH:Development rclenden$

## Working with Properties

Perform these steps:

1. Create a Mod7\_3 folder inside the Mod7 folder.
2. Copy the files as indicated in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| From Folder | File | To Folder | File |
| Mod7\_2 | get\_7\_2\_v1.py | Mod7\_3 | get\_7\_3\_v1.py |
| Mod7\_2 | main\_7\_2\_v2.py | Mod7\_3 | main\_7\_3\_v1.py |

Table 1: Files for Module 7.2

### Creating the Properties File

Perform these steps:

1. Create a new file in Mod7\_3 named config\_7\_3\_v1.properties.
2. Copy and paste these contents into the new file (but use the information for your FMC):

[FmcInfo]

fmc.server=https://10.81.127.36

fmc.username=api

fmc.password=superpass

1. Modify the contents of main\_7\_3\_v1.py as follows to read in the configuration file.

from get\_7\_3\_v0 import get\_names\_and\_ids

import configparser

config = configparser.RawConfigParser()

config.read('config.properties')

print(config.get('FmcInfo', 'fmc.server'))

# fmc\_info = {

# 'server': 'https://10.81.127.36',

# 'username': 'api',

# 'password': 'superpass'

# }

# endpoint = 'policy/accesspolicies'

# print (get.get\_object(fmc\_info, endpoint))

# endpoint = 'policy/accesspolicies/005056A4-5126-0ed3-0000-042949673459/accessrules'

# print (get.get\_object(fmc\_info, endpoint))

1. Run the main\_7\_3\_v1.py program to verify that the properties are read successfully. This test should print only the server.

### Using the Properties File

We are now ready to use the new properties file. Perform the following steps:

1. Copy main\_7\_3\_v1.py as main\_7\_3\_v2.py.
2. Modify the main\_7\_3\_v2.py program as follows:

from get\_7\_3\_v0 import get\_names\_and\_ids

import configparser

config = configparser.RawConfigParser()

config.read('config\_7\_3\_v1.properties')

config.get('FmcInfo', 'fmc.server')

fmc\_info = {

'server': config.get('FmcInfo', 'fmc.server'),

'username': config.get('FmcInfo', 'fmc.username'),

'password': config.get('FmcInfo', 'fmc.password')

}

endpoint = 'policy/accesspolicies'

item\_list = get\_names\_and\_ids(fmc\_info, endpoint)

for item in item\_list:

print(f'{item["name"]} {item["id"]}')

endpoint = 'policy/accesspolicies/005056A4-5126-0ed3-0000-042949673459/accessrules'

item\_list = get\_names\_and\_ids(fmc\_info, endpoint)

for item in item\_list:

print(f'{item["name"]} {item["id"]}')

1. Run main\_7\_3\_v2.py and verify that it executes just the same as before.

### Separating the Properties Reading and Parsing

Too much of our main program is now cluttered dealing with these properties! We can put these details into another module.

Perform the following steps.

1. Create a new file in the Mod7\_3 folder and name it settings\_7\_3\_v1.py.
2. Copy and paste these contents into the new file:

import configparser

config = configparser.RawConfigParser()

config.read('config\_1.properties')

def get\_fmc\_info():

config.get('FmcInfo', 'fmc.server')

fmc\_info = {

'server': config.get('FmcInfo', 'fmc.server'),

'username': config.get('FmcInfo', 'fmc.username'),

'password': config.get('FmcInfo', 'fmc.password')

}

return fmc\_info

1. Copy the main\_7\_3\_v2.py file as main\_7\_3\_v3.py.
2. Modify the new main file as follows:

from get\_7\_3\_v0 import get\_names\_and\_ids

from settings\_7\_3\_v1 import get\_fmc\_info

import configparser

fmc\_info = get\_fmc\_info()

endpoint = 'policy/accesspolicies'

item\_list = get\_names\_and\_ids(fmc\_info, endpoint)

for item in item\_list:

print(f'{item["name"]} {item["id"]}')

endpoint = 'policy/accesspolicies/005056A4-5126-0ed3-0000-042949673459/accessrules'

item\_list = get\_names\_and\_ids(fmc\_info, endpoint)

for item in item\_list:

print(f'{item["name"]} {item["id"]}')

1. Execute the main\_7\_3\_v3.py and verify it works as before.

### Creating a function for printing items

We have duplicate code for printing. Let’s create a function to eliminate the duplicate code.

Perform the following steps:

1. Copy main\_7\_3\_v3.py as main\_7\_3\_v4.py
2. Modify main\_7\_3\_v4.py as follows and execute it, verifying it works as before.

from get\_7\_4\_v0 import get\_names\_and\_ids

from settings\_7\_4\_v0 import get\_fmc\_info

import configparser

fmc\_info = get\_fmc\_info()

def print\_item\_list(item\_list):

for item in item\_list:

print(f'{item["name"]} {item["id"]}')

endpoint = 'policy/accesspolicies'

print\_item\_list(get\_names\_and\_ids(fmc\_info, endpoint))

endpoint = 'policy/accesspolicies/005056A4-5126-0ed3-0000-042949673459/accessrules'

print\_item\_list(get\_names\_and\_ids(fmc\_info, endpoint))

### Creating a utility module

Let’s separate this function into a utility module. Perform the following steps:

1. Create a new file called utils\_7\_3\_v1.py with these contents:

#A file for useful functions

def print\_item\_list(item\_list):

for item in item\_list:

print(f'{item["name"]} {item["id"]}')

1. Copy main\_7\_3\_v4.py to main\_7\_4\_v5.py.
2. Modify main\_7\_4\_v5.py to use the function in the utils module, as shown below.

from get\_7\_4\_v0 import get\_names\_and\_ids

from settings\_7\_4\_v0 import get\_fmc\_info

from utils\_7\_4\_1 import print\_item\_list

import configparser

fmc\_info = get\_fmc\_info()

endpoint = 'policy/accesspolicies'

print\_item\_list(get\_names\_and\_ids(fmc\_info, endpoint))

endpoint = 'policy/accesspolicies/005056A4-5126-0ed3-0000-042949673459/accessrules'

print\_item\_list(get\_names\_and\_ids(fmc\_info, endpoint))

1. Verify that the new main program works as before.

### Creating a constants file

Let’s separate this function into a utility module. Perform the following steps:

1. Create a new file called constants\_7\_3\_v1.py.
2. Copy these contents into the file:

#A file for useful constants

#Endpoints

ACCESS\_POLICIES = 'policy/accesspolicies'

1. Copy main\_7\_3\_v5.py to main\_7\_3\_v6.py.
2. Modify main\_7\_3\_v6.py as follows:

from get\_7\_5\_v0 import get\_names\_and\_ids

from settings\_7\_5\_v0 import get\_fmc\_info

from utils\_7\_5\_v0 import print\_item\_list

import constants\_7\_5\_v1 as constants

fmc\_info = get\_fmc\_info()

print\_item\_list(get\_names\_and\_ids(fmc\_info, constants.ACCESS\_POLICIES))

## Module 7.4 Has Been Eliminated

## Module 7.5 Has Been Eliminated

## Logging

Often we insert print statements to assist with debugging, but logging is far more efficient as our program grows larger.

Perform these steps:

1. Copy the Mod7\_3 folder as Mod7\_5, and update the file names to reflect that the module is now 7\_5.
2. Modify the main\_7\_6\_v1.py to add logging as shown below:

from get\_7\_5\_v0 import get\_names\_and\_ids

from settings\_7\_5\_v0 import get\_fmc\_info

from utils\_7\_5\_v0 import print\_item\_list

import constants\_7\_5\_v1 as constants

import logging

logging.basicConfig(filename='application.log',level=logging.INFO)

def main():

logging.basicConfig(filename='myapp.log', level=logging.INFO)

logging.info('Starting the application.')

logging.info('Retrieving the FMC info.')

fmc\_info = get\_fmc\_info()

logging.info('Retrieving and printing the access policies.')

print\_item\_list(get\_names\_and\_ids(fmc\_info, constants.ACCESS\_POLICIES))

logging.info('Finished execution.')

if \_\_name\_\_ == '\_\_main\_\_':

main()

1. Add logging statements to get\_7\_6v1.py, and convert any print statements to logging statements. See the solution for assistance.
2. Execute the main\_7\_6\_v1.py program.
3. Verify that the application.log file contains the log statements that you expect.

# Module 8: GETting and POSTing Objects

For Module 8 and future modules, you will create a folder for the module, such as Mod8, Mod9, etc. And for each subsection in Module 8, you will create a folder such as Mod8\_1, Mod8\_2, etc., and copy the files from the previous section as appropriate.

## GETting Objects

1. Add additional endpoints to the constants file as shown below.

#Endpoints

ACCESS\_POLICIES = 'policy/accesspolicies'

OBJECT\_HOSTS = 'object/hosts'

OBJECT\_NETWORKS = 'object/networks'

OBJECT\_NETWORK\_ADDRESSES = 'object/networkaddresses'

OBJECT\_NETWORK\_GROUPS = 'object/networkgroups'

1. Test the OBJECT\_NETWORKS constant by adding it to the main program:

logging.info('Retrieving and printing the network objects.')

print\_item\_list(get\_names\_and\_ids(fmc\_info, constants.OBJECT\_NETWORKS))

1. Run the main program and verify that you see the names and IDs of the networks.

## POSTing an Object

1. In API Explorer, Legacy Explorer, get the sample code for object/networks POST.
2. Create a new file, post\_8\_2\_v1.py and paste the sample code in it.
3. Make the needed changes to turn off SSL verification.
4. Remove all comments.
5. Make everything below the import a post(fmc\_info, endpoint, post\_data) function.
6. Copy these lines from get and use them to replace the lines above r = None.

server = fmc\_info['server']

username = fmc\_info['username']

password = fmc\_info['password']

1. Import logging, and replace print statements with log statements.
2. The print that displays the return data can be a logging.debug. Other print statements should become logging.info or logging.error.
3. Remove the line that tries to assign the post\_data.
4. Add a line to assign json\_resp
5. Now for the the post\_data.
   1. Get the example JSON data from the OpenAPI POST of object/networks, and copy/paste it into the main\_8\_2\_v1.py file.
6. All you need is the following, so eliminate everything else from the JSON:
   1. Name: TestNetwork1
   2. Type: Network
   3. Value: 10.1.2.3/24
7. The result should be similar to the following:

def main():

logging.basicConfig(filename='myapp.log', level=logging.INFO)

logging.info('Starting the application.')

logging.info('Retrieving the FMC info.')

fmc\_info = get\_fmc\_info()

post\_data = {

"name": "TestNetwork1",

"type": "Network",

"value": "10.1.2.3/24"

}

logging.info('Posting the network object.')

print(post(fmc\_info, constants.ACCESS\_POLICIES))

logging.info('Finished execution.')

1. Execute the program to POST the network, and verify it has been POSTed, both in the FMC and in the log.

## POSTing Objects and Groups in Bulk

### POSTing Network Objects from a CSV File

1. In this section you will modify main\_8\_3\_v1.py.
2. Create a CSV with three fields:
   1. Name
   2. Type
   3. Value
3. Create objects in the CSV for these types. Make sure that the names of your objects begin with your initials, so that they are identified with you.
   1. Network
   2. Range
   3. Host
4. Write code to read in the file. Each line is a list of strings.
5. Create a dictionary to link type and endpoint.
6. Move the authentication code to the utils module, since it is only necessary to authenticate once at the beginning, and we can reuse that token for all subsequent calls.
7. Write code to iterate through the rows of the CSV file and POST each object.

### POSTing Network Groups in Bulk

1. In this section you will modify main\_8\_3\_v2.py.
2. Create a new CSV with the fields needed to support network groups.
3. Copy the code for reading a net object csv, and paste it below that code so you can modify it to read the network group csv.
4. Temporarily, comment out the network post code to focus on the network group post code.
5. For groups, instead of a value, the group data consists of literals and objects:

literals = row[2]

objects = row[3]

1. For literals, you will need to create a list of literals where each item in the list is a dictionary:

{

“type”: “Host”,

“value”: “10.2.3.4”

}

1. For group members that are objects, each object will be identified by a type and an id. We need a dictionary of objects and groups, where the keys are the names of the objects/groups, and the values are dictionaries that have type and id. Create this dictionary by calling the get\_names\_and\_ids() method (which we have to expand to also return the type).
2. Move the code that retrieves the token out of the get\_object() function into the utils module, so that the main program can get the token once and make it part of fmc\_info, and all future calls can use that same token.
3. Write code in get\_names\_and\_ids() to handle paging. (Visit API Explorer to see what paging looks like.) To see paging in operation, GET networkaddresses and specify a limit of 5:

"paging": {

"offset": 0,

"limit": 5,

"count": 13,

"next": [

"https://10.81.127.194/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/networkaddresses?offset=5&limit=5",

"https://10.81.127.194/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/networkaddresses?offset=10&limit=3"

],

"pages": 3

}

When there are no more pages, then there is no key of “next” in the return dictionary:

"paging": {

"offset": 10,

"limit": 3,

"count": 13,

"prev": [

"https://10.81.127.194/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/networkaddresses?offset=7&limit=3",

"https://10.81.127.194/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/networkaddresses?offset=4&limit=3",

"https://10.81.127.194/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/networkaddresses?offset=1&limit=3",

"https://10.81.127.194/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/networkaddresses?offset=0&limit=1"

],

"pages": 5

}

Thus, you can employ a loop that stops when “next” does not appear in the output.

1. Note that in paging, the API provides the entire URL, and we’d like to use this. But our current get\_object method only takes an “endpoint” argument, which is just the end of the path. Write code to allow the get\_object method to optionally take an entire url.
2. At this point, in the main program, you may need to modify the code that creates a dictionary with both the network addresses and groups.

## Arguments and Abstraction

Although the code in our main program works, it may be over 100 lines since it has all the details of how to POST network objects and groups. It can also GET and POST objects. In this module you will abstract much of the main code into the post module. But we will begin by using argparse to specify operations via the command line.

### argparse for GET

Consider that our program can do many things now. How can we specify the single operation we want to do at a given moment? One option is to use command line arguments to specify the operation and any needed parameters for the operation we want to perform.

Using command line arguments, we can specify whether we want to get or post objects. If we want to get them, we can specify the types of object we want to get. If we want to post them, we can specify a file that contains the objects to be posted.

The python.org web site has a good tutorial on how to use argparse to incorporate command line arguments into our program. The tutorial is at: <https://docs.python.org/3.3/howto/argparse.html#id1>

To post objects, we will use a command line argument of --post, and to get them we will use a command line argument of --get.

Perform these steps:

1. We want to specify natural object types when getting objects, such as --get Networks. Create a dictionary in the constants file that maps the natural object types to the endpoints (also in constants).
2. In the main file (main\_8\_4\_v1.py), write the code to accept arguments for get and post.
3. For now, we will just test the GET functionality. In the main program, write an if statement that tests for --get at the command line, and calls the print function on the get\_names\_and\_ids function to print the specified object types.

### Argparse for POST

In this section, we will work with main\_8\_4\_v2.py and post\_8\_4\_v1.py.

Perform these steps:

1. Write a post\_from\_file() function in the post module, and move the post code from main into that module.
2. Create functions in the post module:
   1. post\_network\_objects()
   2. post\_network\_groups()
   3. post\_protocolport\_objects()
   4. post\_port\_groups()

Our code needs to determine what kind of file it is working with. It turns out that the third header of each file is unique to that type of file.

Perform these steps:

1. Create a dictionary that maps the third header in the CSV file to the appropriate function.
2. In the post\_from\_file() function, write code that reads the file, and based on the third header, calls the appropriate function to POST the objects.

### Posting from net group file

In this section, we will improve on post, in post\_8\_4\_v2.py.

Perform these steps:

1. Move the post network group code into the post\_network\_groups() function.
2. Remove any unnecessary code from the function (i.e., it is no longer necessary to read in the file in this function).
3. Update imports as needed.

### Abstracting argparse and post

In this section, we will improve the code by introducing more abstraction. Perform these steps.

1. In the main program (now, main\_8\_4\_v3.py), create a function for get\_arguments() that contains the code for getting the arguments.
2. In the post module (now, post\_8\_4\_v3.py), use functions to simplify the code for posting the network groups.
3. Remove unnecessary comments from all code modules.

# Module 9: POSTing AC Rules

In this module, we will POST AC rules. As with other POSTs, we will put the rules into a CSV file.

## POSTing Simple AC Rules

In this section, you will work on post\_9\_1\_v1.py. Perform these steps:

1. Create a CSV file for access rules.
2. Create a function in the post module to post rules. For now, hardcode the id of a specific access policy into the code. In the next section, we will explore how to get this id from the name.
3. Create an entry in the THIRD\_HEADER\_MAP dictionary to steer execution to the post rules function if the third header is description.
4. Test the code to ensure that you can successfully POST the rule(s) into the policy.

## POSTing AC Rules with AC Policy Lookup

In the last section, we hard-coded the ID of the AC policy into the endpoint that the post code used. In this section, you will glean the name of the policy from the CSV and perform a lookup for the ID.

Before we start iterating through the rows of the file, we want to create a list of access policies. Each access policy should be a dictionary of name and ID, so we can look up a policy by name and get its ID.

In this section, you will work on post\_9\_1\_v2.py. Perform these steps:

1. Get the names and ID’s of the access policies and store them in a list.
2. Write a for loop inside the row iterator that iterates through the list of access policies looking for the name of the policy that was in the file. When it finds it, the loop terminates and the policy id is preserved and is made part of the endpoint that is sent to the post function.

# Module 12: Converting Rules from Text

We will explore reading rules from a text file, creating rule objects from them, and POSTing the rules.

# Module 13: AC Rule Advanced Operations

We will explore POSTing rules in bulk, into specific sections and categories, and before and after another rule.

# Module 14: Deployment

We will deploy changes to the vFTD firewall.

# Appendix A: Acronym Listing

|  |  |
| --- | --- |
| Term | Definition |
| API | Application Programming Interface |
| FMC | Firepower Management Center |
| REST | Representational State Transfer |
|  |  |
|  |  |

# Appendix B: Answers to Exercises

## Module 1 Answers/Results

|  |  |
| --- | --- |
| Section | Answers/Results |
| 1.2 | You should be able to successfully log into the FMC GUI. |
| 1.3 | You should find that the FMC REST API is already enabled (it is enabled by default). |
| 1.4 | You should be able to log in successfully to the FMC API Explorer and access the Legacy Explorer. |
| 1.5 | You should be able to access the referenced links. |
| 1.6 | Verify that you get a 200 response code and can identify the names of the Access Control policies in the JSON response. |

Table 1: Module 1 answers

## Module 2 Answer

|  |
| --- |
| JSON Exercise Answer |
| {  "items": [  {  "name": "Allow DNS",  "log":false,  },  {  "name": "Allow SSH",  "log":true,  }  ]  } |

Table 1: JSON exercise answer

## Module 3 Answers

|  |
| --- |
| Successful Reponse for Requesting a Particular Policy |
| {  "metadata": {  "inherit": true,  "parentPolicy": {  "type": "AccessPolicy",  "name": "ParentPolicy",  "id": "005056A4-5126-0ed3-0000-042949673459"  },  "domain": {  "name": "Global",  "id": "e276abec-e0f2-11e3-8169-6d9ed49b625f",  "type": "Domain"  }  },  "type": "AccessPolicy",  "links": {  "self": "https://10.81.127.36/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/policy/accesspolicies/005056A4-5126-0ed3-0000-025769803780"  },  "rules": {  "refType": "list",  "links": {  "self": "https://10.81.127.36/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/policy/accesspolicies/005056A4-5126-0ed3-0000-025769803780/accessrules"  },  "type": "AccessRule"  },  "defaultAction": {  "type": "AccessPolicyDefaultAction",  "id": "005056A4-5126-0ed3-0000-000268434432"  },  "name": "ChildPolicy",  "description": " ",  "id": "005056A4-5126-0ed3-0000-025769803780"  } |

|  |
| --- |
| Successful Reponse for Requesting a Policy’s Access Rules |
| {  "links": {  "self": "https://10.81.127.36/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/policy/accesspolicies/005056A4-5126-0ed3-0000-025769803780/accessrules?offset=0&limit=5"  },  "items": [  {  "links": {  "self": "https://10.81.127.36/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/policy/accesspolicies/005056A4-5126-0ed3-0000-025769803780/accessrules/005056A4-5126-0ed3-0000-000268435457"  },  "id": "005056A4-5126-0ed3-0000-000268435457",  "type": "AccessRule",  "name": "ImportantRule1"  },  {  "links": {  "self": "https://10.81.127.36/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/policy/accesspolicies/005056A4-5126-0ed3-0000-025769803780/accessrules/005056A4-5126-0ed3-0000-000268436480"  },  "id": "005056A4-5126-0ed3-0000-000268436480",  "type": "AccessRule",  "name": "Allow All"  },  {  "links": {  "self": "https://10.81.127.36/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/policy/accesspolicies/005056A4-5126-0ed3-0000-025769803780/accessrules/005056A4-5126-0ed3-0000-000268434433"  },  "id": "005056A4-5126-0ed3-0000-000268434433",  "type": "AccessRule",  "name": "Test"  },  {  "links": {  "self": "https://10.81.127.36/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/policy/accesspolicies/005056A4-5126-0ed3-0000-025769803780/accessrules/005056A4-5126-0ed3-0000-000268437504"  },  "id": "005056A4-5126-0ed3-0000-000268437504",  "type": "AccessRule",  "name": "Another Rule"  },  {  "links": {  "self": "https://10.81.127.36/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/policy/accesspolicies/005056A4-5126-0ed3-0000-025769803780/accessrules/005056A4-5126-0ed3-0000-000268435458"  },  "id": "005056A4-5126-0ed3-0000-000268435458",  "type": "AccessRule",  "name": "LowImportanceRule"  }  ],  "paging": {  "offset": 0,  "limit": 5,  "count": 5,  "pages": 1  }  } |

## Module 4 Answers

A screenshot of a cell phone

Description automatically generated

The URL to request access policies:

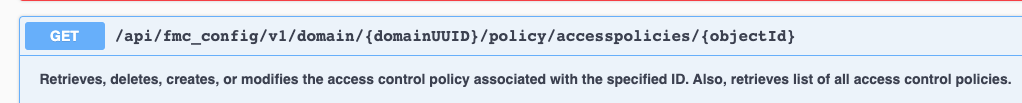
**https://10.81.127.36/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/policy/accesspolicies**

Example of a successful response to a request for Access Policies.

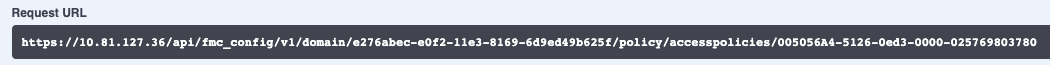
A screenshot of a social media post

Description automatically generated

The section for requesting a particular AC Policy:



Example Request URL



Example request for the rules in an ac policy

**https://10.81.127.36/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/policy/accesspolicies/005056A4-5126-0ed3-0000-025769803780/accessrules**

# Appendix C: VS Code Shortcuts

|  |  |
| --- | --- |
| Shortcut | Operation |
| Cmd-/ | Comment out selected lines |
| Ctrl-~ | Open a Terminal |
| Cmd-Shift-K | Delete selected lines |
| Alt-Shift-up arrow | Copy a line |
| Alt-Shift-down arrow | Copy a line |
| Alt-up arrow | Move a line up |
| Alt-down arrow | Move a line down |
| F2 | Rename |

Trademarks and Disclaimers

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THIRD PARTY SUPPLIERS ARE PROVIDED “AS IS” WITH ALL FAULTS. CISCO AND THIRD PARTY SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Cisco and the Cisco Logo are trademarks of Cisco Systems, Inc. and/or its affiliates in the U.S. and other countries. A listing of Cisco's trademarks can be found at www.cisco.com/go/trademarks. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

©2017 Cisco Systems, Inc. All rights reserved.

End of Document