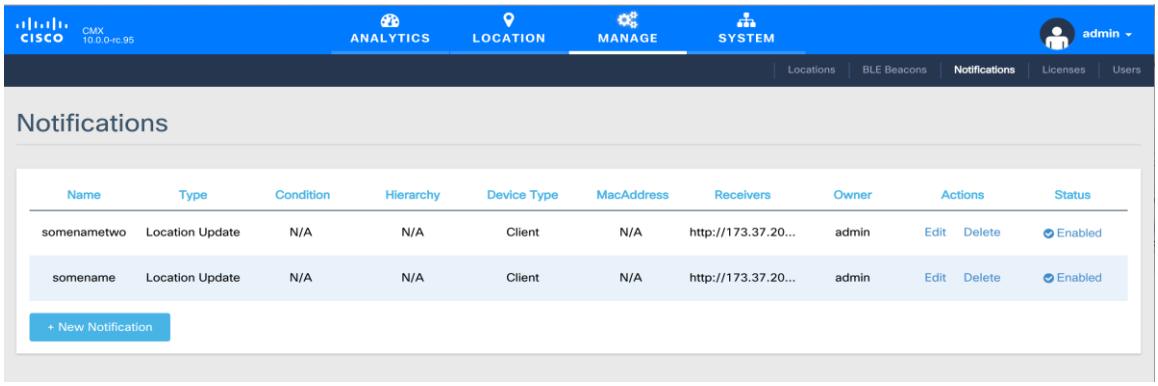
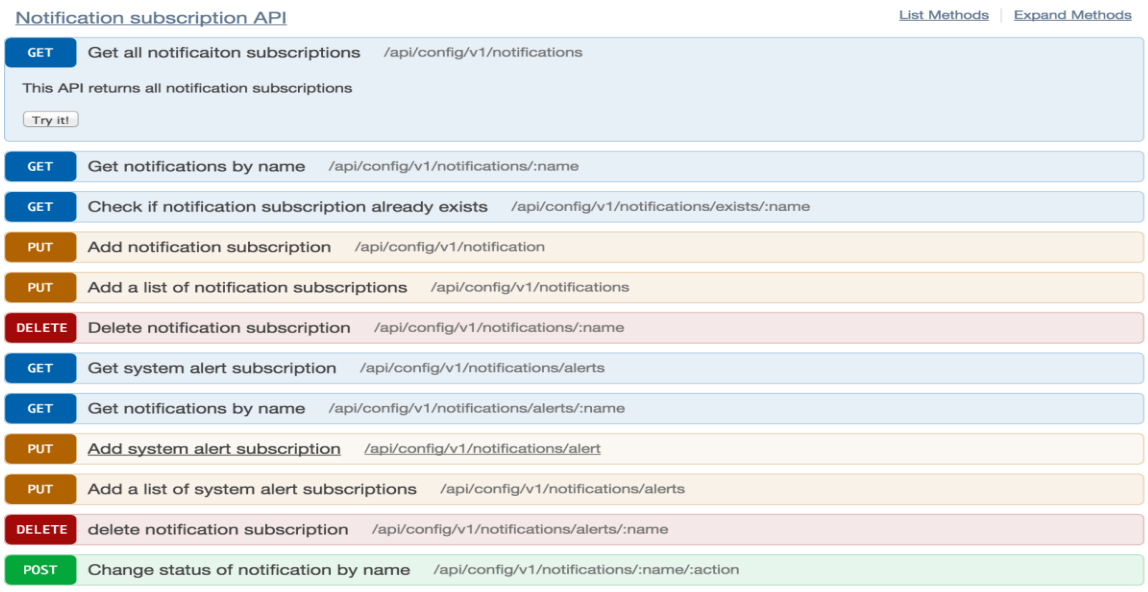
Cisco Mobility Services Engine Partner Developer Guide




```
import urllib2

mse_user = 'username'                ##MSE Username
mse_pass = 'password'                ##MSE Password

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(
    None, 'https://ip-address-of-mse', mse_user, mse_pass     ##IP Address or resolvable hostname
)
auth_handler = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_handler)
urllib2.install_opener(opener)
```



CMX Mobility Services

Using CMX Mobility Services, developer may leverage device location via WiFi and device MAC address. Customer experiences may be improved by providing coupons, promotions, and other push notifications to devices. Additionally various client location based service solutions may be developed providing many useful applications for use by the end device user as well as the venue operator.

CMX Mobility Services fall into two main categories:

- Cisco Mobile Application Server with CMX Mobility Application SDK
- CMX Mobility Services Restful API

The Cisco Mobile Application Server with SDK solution is an exciting suite of mobile software solutions that detect, connect, and engage with mobile devices operating in a WiFi field. The individual mobile software offerings work together to create a total solution that you can configure to the benefit of your clients and their end users in a very wide variety of real world situations.

Additionally the CMX Mobility Application SDK is used to create mobile device apps with location tracking and navigation for end users carrying WiFi enabled mobile phones. Using the CMX Mobility Application SDK's set of libraries, you can create push notifications to invite users to join networks, receive offers or get special information whenever the user is in range of designated Points of Interest. The CMX Mobile Application Server acts as the middleware between the Mobility Services Engine and the mobile device application that you built with the CMX Mobility Application SDK. Your mobile app will be able to show the user’s location, display maps, point to associated points of interest, and make route suggestions for the user. The real-world use cases for the mobile features provided by this platform are endless.

The CMX Mobility Services Restful API allows you to use them across languages, platforms, and frameworks. Using the APIs, you can develop application solutions that will use real-time intelligence gathered from your Wi-Fi network to enable people and their devices to interact more effectively through real-time contextual information such as location, temperature, availability of a user, or mobile device asset.

This guide will focus on the CMX Mobility Services Restful API. Information regarding the Cisco Mobile Application Server with CMX Mobility Application SDK may be found here ---

http://www.cisco.com/c/en/us/td/docs/wireless/mse/8-0/CMX-Connect-and-Engage-Mobile-SDK/guide/Cisco-CMX-ConnectEngage-Mobile-SDK-Config-Guide.html

Information for all CMX Mobility Services features, and guides may be viewed at the Cisco DevNet Site.

Table of Contents

Restful API Support

Cisco Mobility Services Engine REST API. The goal is to provide a simplified API for use in various ways. The advantages of RESTful APIs:

- Based on the stateless REST architecture.
- The information is delivered over the well-understood HTTP/HTTPS protocol.
- Simplified design and easy to understand Resource URIs. These are usually self-explanatory.
- Standardized libraries allow content negotiation capabilities by automatically sending the response in the user requested format.
- Gives us a chance to simplify our data model, deprecate the older non scalable APIs, remove hierarchy dependency and do other enhancements.

The documentation is divided into three logical categories based on the specific functionality. The categories are as follows –

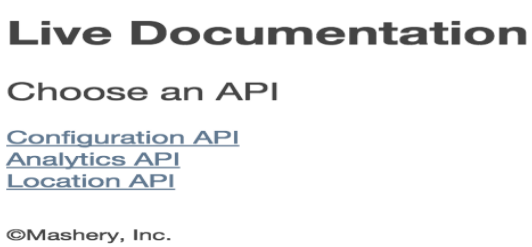
- Configuration API
- Analytics API
- Location API

The REST API uses the following http methods –

- GET
- PUT
- POST
- DELETE

The Cisco Mobility Services Engine provides updated documents specific to the REST API. The documentation provide the name, methods, and structure of the REST API query. The documents may be found at the following location on the server:

http://mse-ip-address/apidocs/



The Configuration API may be used to configure the Cisco MSE programmatically. The following categories are available –

- Operations For Managing Heterachy
- History Alerts
- Licenses
- Map Resources
- Users
- Cache
- MSE Configuration
- Alerts
- Mail Service
- Notification Subscriptions

The Analytics API may be used to retrieve Analytics data programmatically from the Cisco MSE. The following categories are available –

- Provide A Summary of Analytics Metrics
- Total Dwell time
- Analytics Notification Alerts
- Device Count
- Overview
- Dwell time

The Location API may be used to retrieve client location data programmatically from the Cisco MSE. The following categories are available-

- Tags Information
- Rogue Information
- Beacon Management
- Client Information
- Northbound Notification Types and Attributes
- Clients Information
- Rogue Clients Information

Interacting with the REST API

Sending queries to the MSE requires a username and password needed to create a Base64 encoded string.

Requests are sent, after authentication, using an http method (GET, PUT, POST, DELETE).

- Authentication can be sent from the client side before the request, using the Authorization header.
- Username and password are combined into a string "username:password".
- The resulting string literal is then encoded using Base64
- The authorization method, a space and the string "Basic" is then put before the encoded string.

For example, if the user agent uses 'MyUsername' as the username and 'MyPassword as the password then the header is formed as follows:

- Authorization: Basic QxhZGluOnNlc2FtIG9wZW4=
- Authentication is sent to the root URI to reach (e.g. https://<mseip>/api/contextaware)

TIP: Python 2.7.x Example to create Base64 encoded authentication string:

```
import urllib2

mse_user = 'username'               ##MSE Username
mse_pass = 'password'              ##MSE Password

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(
    None, 'https://ip-address-of-mse', mse_user, mse_pass     ##IP Address or resolvable hostname
)
auth_handler = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_handler)
urllib2.install_opener(opener)
```

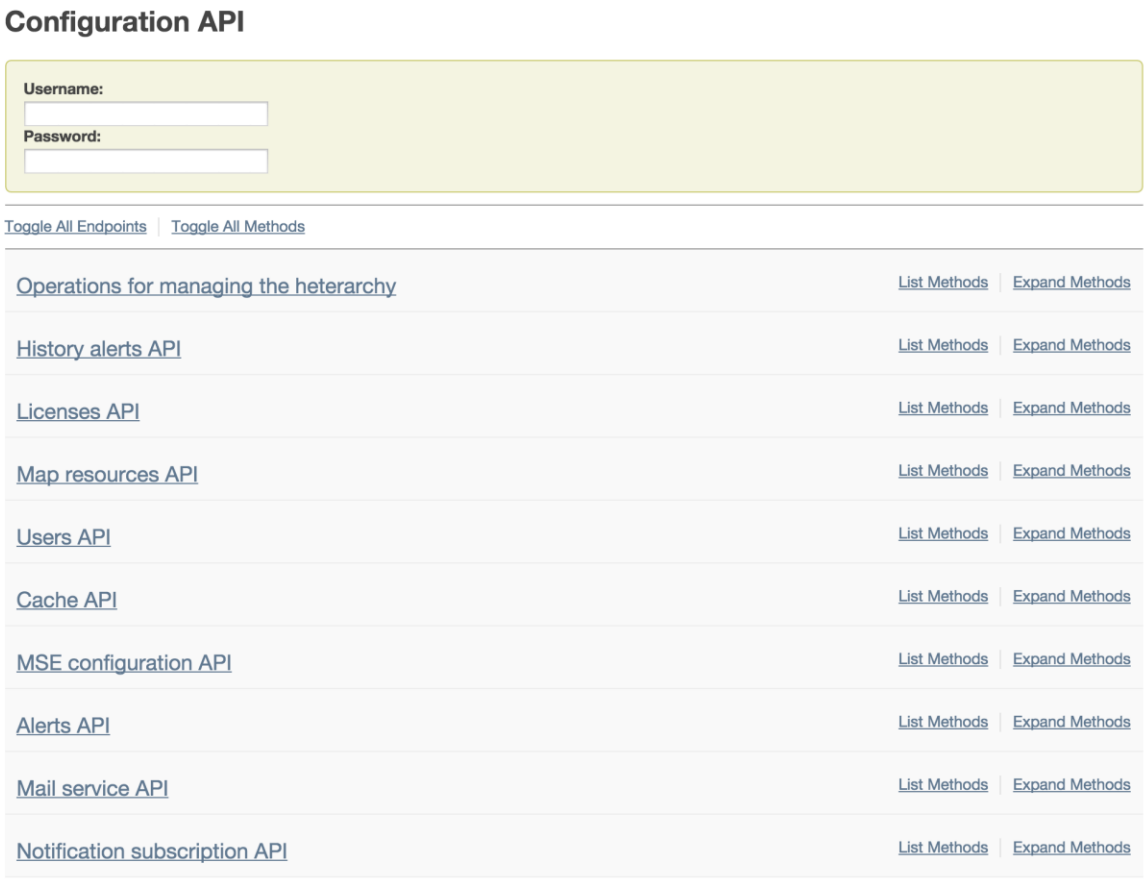
There are three ways to interact with the Cisco MSE REST API. Those include the following:

- "Try It" method.
- Use of REST Client plugin Chrome or Firefox browser.
- Programmatically.

"Try It"

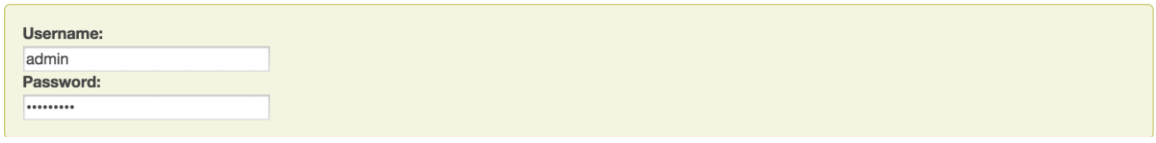
Via the documents link listed previously:

Step 1. Select the category of interest (Configuration, Analytics, or Location)

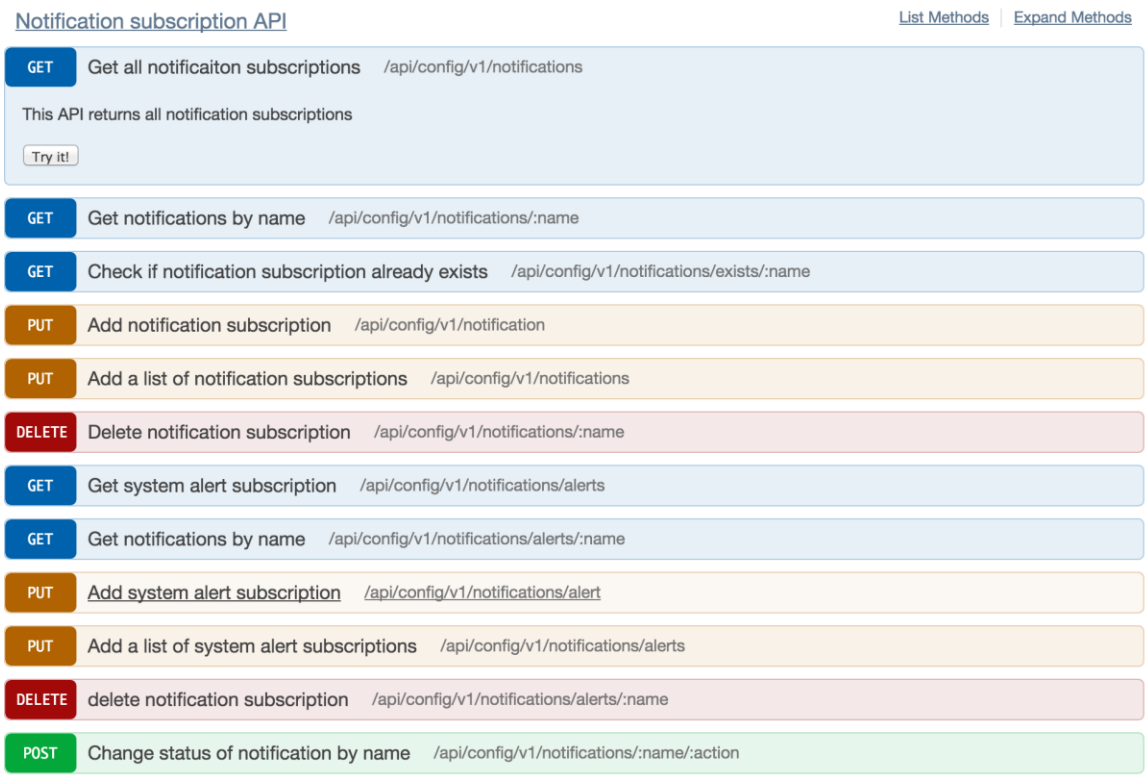


Step 2. Enter the Username/Password to access the Cisco MSE.

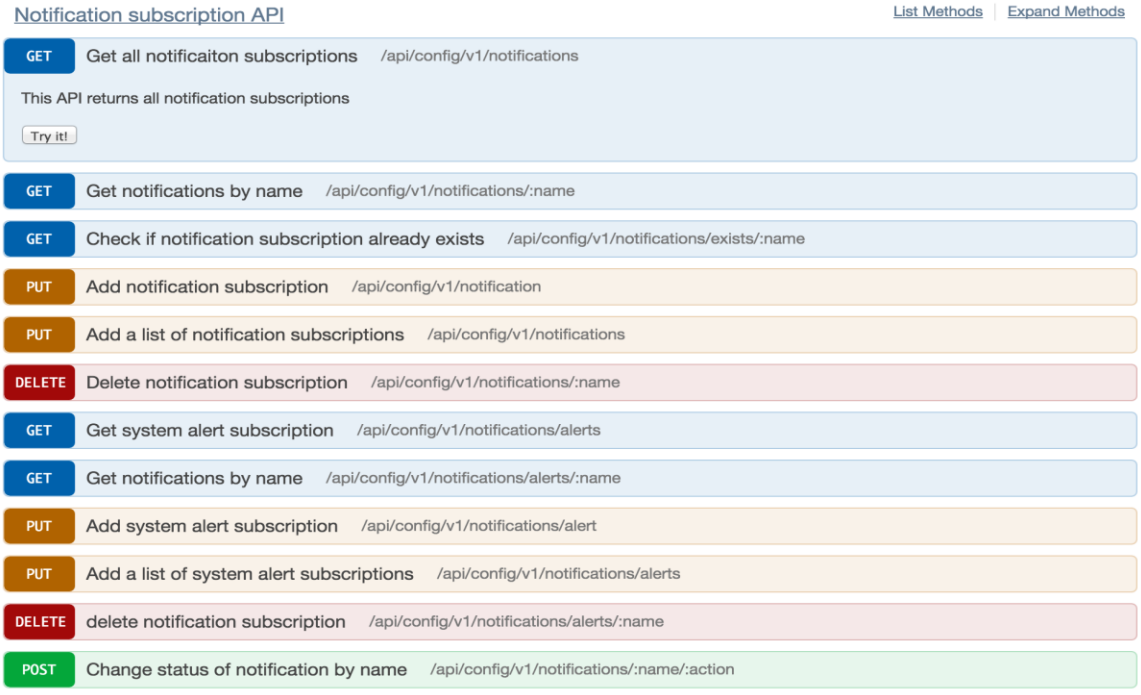
TIP: The username and password are configured by the system administrator via the Cisco MSE UI.



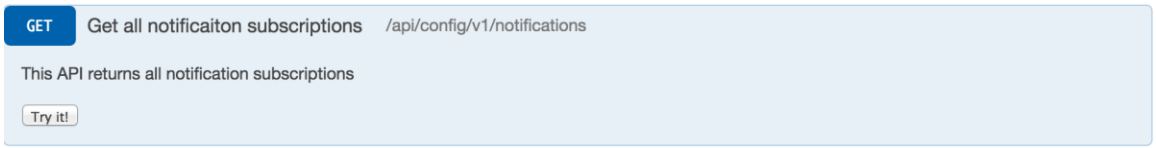
Step 3. Choose a subcategory (in this case Notifications).



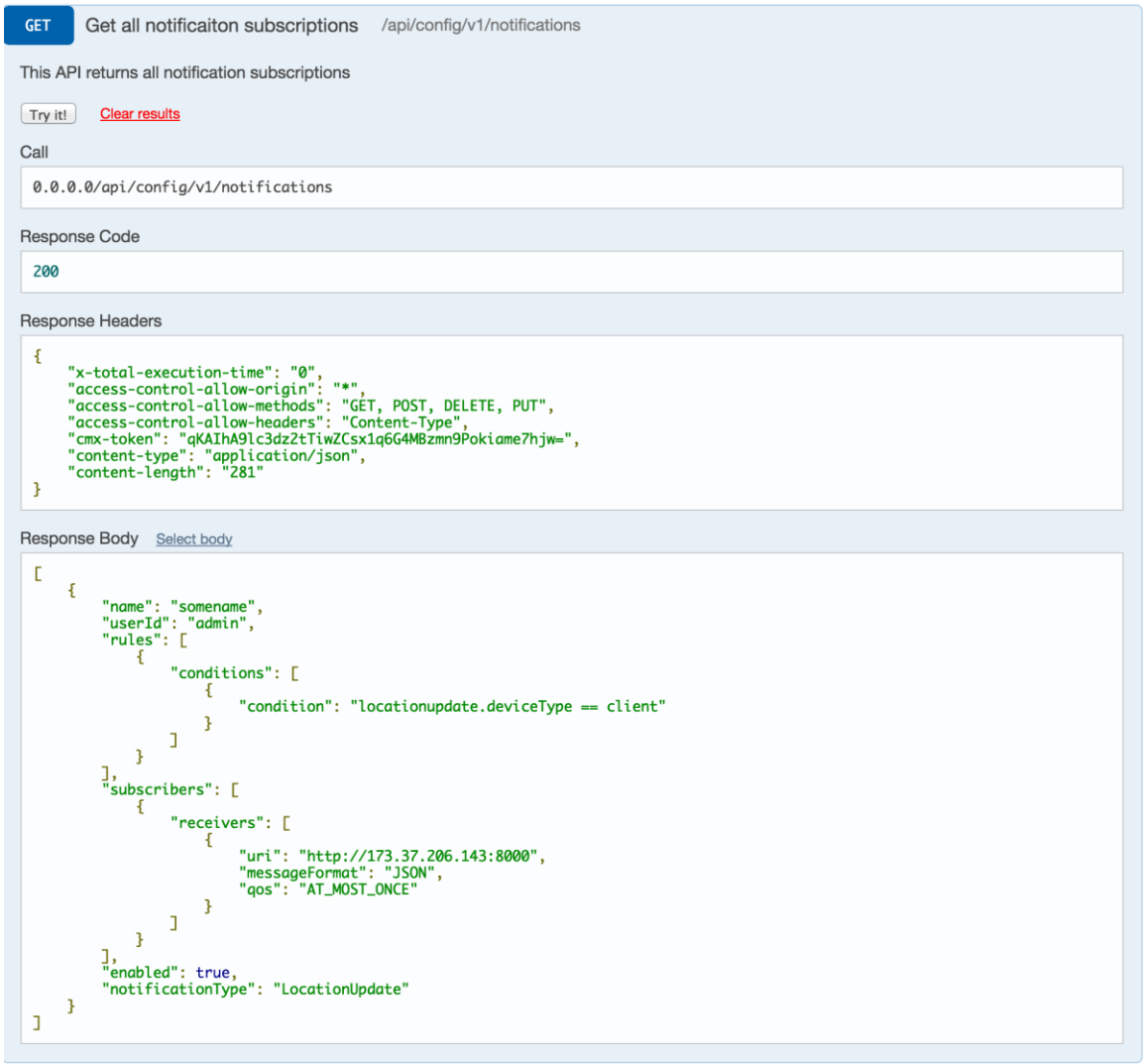
Step 4. Select the Method of interest (in this case GET all notification subscriptions).



Step 5. Select "Try It"



Step 6. Examine the results.



Note that the following information is made available:

- Call
- Response Code
- Response Headers
- Response Body

TIP: The Response Body may be used to understand the form of the PUT for the same item.

The Call shows the structure of the query, in this case, an may be copied and pasted into a web browser for a quick way to test syntax.
The response code provides information about the query such as if it was successful. This may provide hints as to why a particular query was not successful.

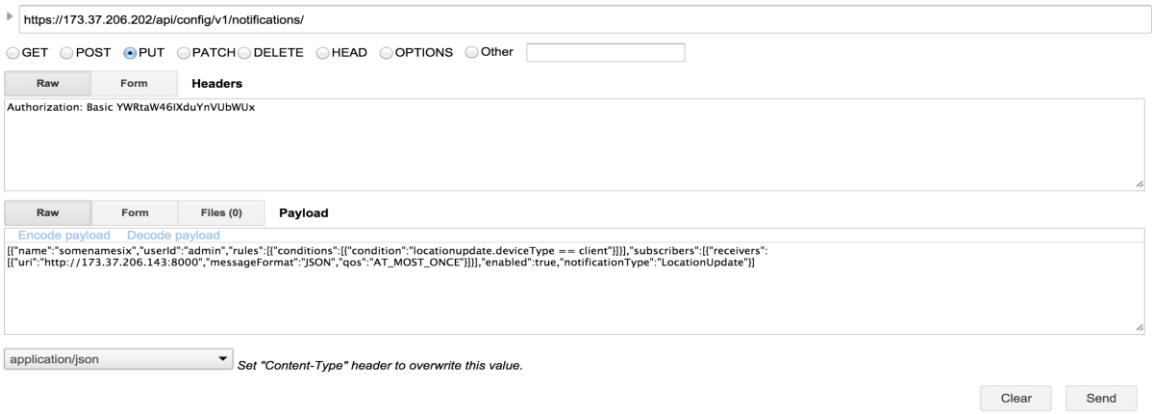
The response Header and Body show the return of information which was requested.

REST Client (Browser Plugin)

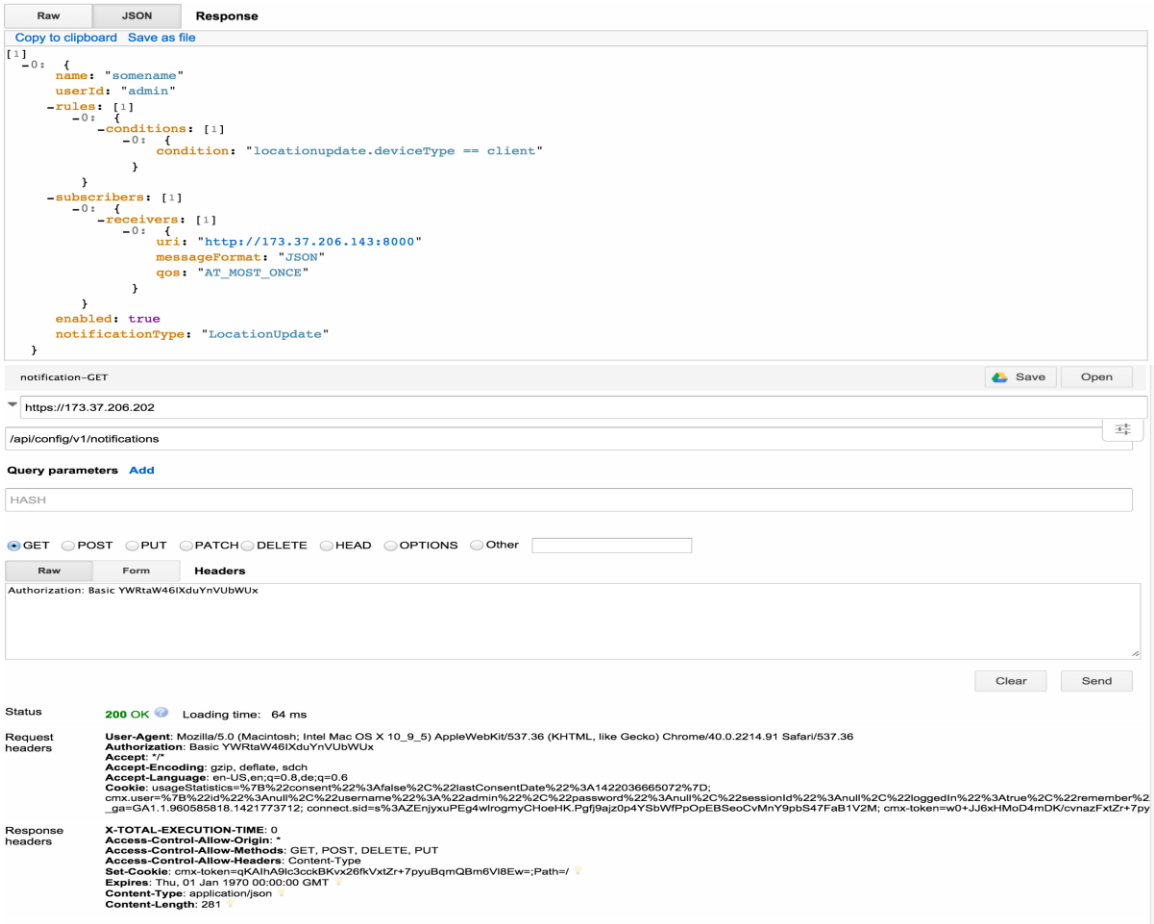
Another way we can test the structure of our queries and other API calls to Cisco MSE is through the use of REST Client plugins for the latest versions of Chrome and Firefox. Some examples are “Advanced REST Client”, Postman, and RESTClient. The particular REST Client used is a personal preference, for these examples we are using advanced REST Client plugin for Chrome.

Note: Describing how to use these REST Clients is not in the scope of this document. Please see instructions for the REST client of choice describing use for the perspective plugin/application.

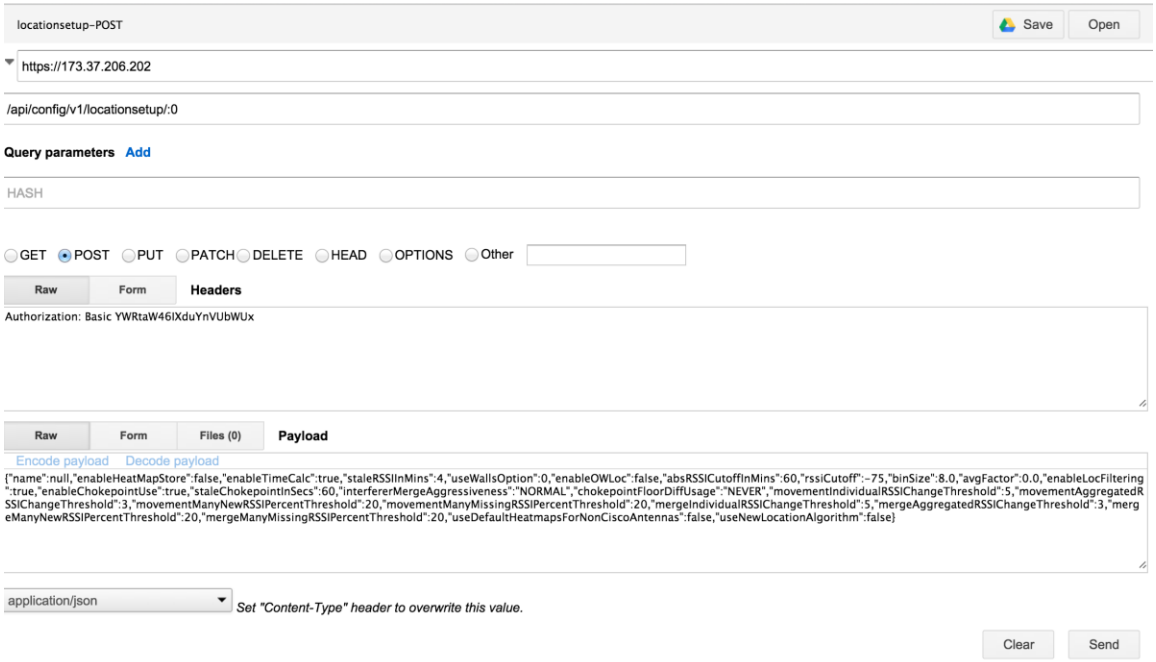
Example of Notification Definition (PUT):



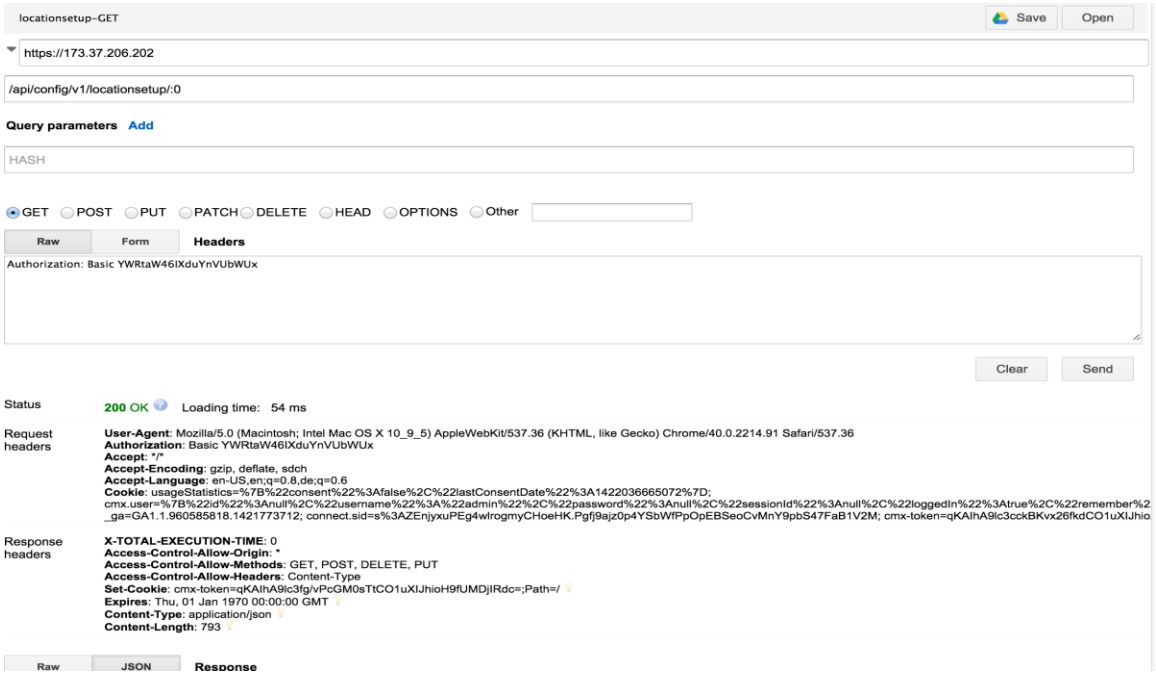
Example of Notification Definition (GET)



Example of Location Setup (POST)



Example of Location Setup (GET)



Programmatically

For this section the example will be using the Python programming language. Python was chosen for the examples because of it's simplicity and availability. Additionally we see Python use extensively for configuration and interaction of resource in the data center environment. Both Python 2.7. and Python 3.4.x are used, comments in code specify which version.

Note: These examples are as simplistic as possible, formatting and other features are not implemented.

Example of Location Setup (GET)


```
{
    name: null
    enableHeatMapStore: false
    enableTimeCalc: true
    staleRSSIInMins: 4
    useWallsOption: 0
    enableOWLoc: false
    absRSSICutoffInMins: 60
    rssiCutoff: -75
    binSize: 8
    avgFactor: 0
    enableLocFiltering: true
    enableChokepointUse: true
    staleChokepointInSecs: 60
    interfererMergeAggressiveness: "NORMAL"
    chokepointFloorDiffUsage: "NEVER"
    movementIndividualRSSIChangeThreshold: 5
    movementAggregatedRSSIChangeThreshold: 3
    movementManyNewRSSIPercentThreshold: 20
    movementManyMissingRSSIPercentThreshold: 20
    mergeIndividualRSSIChangeThreshold: 5
    mergeAggregatedRSSIChangeThreshold: 3
    mergeManyNewRSSIPercentThreshold: 20
    mergeManyMissingRSSIPercentThreshold: 20
    useDefaultHeatmapsForNonCiscoAntennas: false
    useNewLocationAlgorithm: false
    maxAllowableDiplacementForBeacon: 20
}
```


Example of Location Setup (POST)

Returns 204 No Content

Example of Notification Subscription (GET)

Cisco MSE can send a real-time stream of all the activity for all clients to a destination. These notifications are a superset of the location update events. This feature can be enabled using the API (PUT and GET).

```python
#using python 3.4.x

from http.client import HTTPSConnection
from base64 import b64encode

#Create the  https connection
c = HTTPSConnection("173.37.206.202")

#encode as Base64
#decode to ascii (python 3 stores as byte string, we don't want that as we need to pass ascii value for auth)

#userAndPass = b64encode(b"admin:!wnbuTme1").decode("ascii")
usernamepasssword = b64encode(b"admin:!wnbuTme1").decode("ascii")
headers = { 'Authorization' : 'Basic %s' %  usernamepasssword}

#connect and ask for resource
c.request('GET', '/api/config/v1/locationsetup/:0', headers=headers)

#reponse
res = c.getresponse()

data = res.read()

#print the results
print (data)
```

```
 {
 name: "somenametwo"
 userId: "admin"
-rules: [1]
   -0:  {
      -conditions: [1]
         -0:  {
              condition: "locationupdate.deviceType == client"
          }
    }
-subscribers: [1]
   -0:  {
      -receivers: [1]
         -0:  {
              uri: "http://173.37.206.143:8000"
              messageFormat: "JSON"
              qos: "AT_MOST_ONCE"
          }
    }
 enabled: true
 notificationType: "LocationUpdate"

 {
 name: "somename"
 userId: "admin"
-rules: [1]
   -0:  {
      -conditions: [1]
         -0:  {
              condition: "locationupdate.deviceType == client"
          }
    }
-subscribers: [1]
   -0:  {
      -receivers: [1]
         -0:  {
              uri: "http://173.37.206.143:8000"
              messageFormat: "JSON"
              qos: "AT_MOST_ONCE"
          }
    }
 enabled: true
 notificationType: "LocationUpdate"
```

Example of Notification Subscription (PUT)

Returns 204 No Content

Example of Client Location List (GET)

```python
#using python 3.4.x

from http.client import HTTPSConnection
from base64 import b64encode

#Create the  https connection
c = HTTPSConnection("173.37.206.202")

#encode as Base64
#decode to ascii (python 3 stores as byte string, we don't want that as we need to pass ascii value for auth)

#userAndPass = b64encode(b"admin:!wnbuTme1").decode("ascii")
usernamepasssword = b64encode(b"admin:!wnbuTme1").decode("ascii")
headers = { 'Authorization' : 'Basic %s' %  usernamepasssword}

#connect and ask for resource
c.request('GET', '/api/location/v1/clients/', headers=headers)

#reponse
res = c.getresponse()

data = res.read()

#print the results
print (data)
```

```
3:  {
 macAddress: "28:b2:bd:33:71:e2"
-mapInfo: {
    mapHierarchyString: "Richardson_TX_75082>Cisco_Building_5>2250_East_PGBT_First_Floor>Lab_Coverage_Area"
    floorRefId: "-5970138651793817391"
   -floorDimension: {
        length: 259
        width: 419
        height: 10
        offsetX: 0
        offsetY: 0
        unit: "FEET"
    }
   -image: {
        imageName: "domain_0_1410184557303.png"
        sourceFile: null
        zoomLevel: 0
        width: 0
        height: 0
        size: 0
        maxResolution: 0
        colorDepth: 0
    }
    tagList: [0]
 }
-mapCoordinate: {
    x: 168.54506
    y: 40.6289
    unit: "FEET"
 }
 currentlyTracked: true
 confidenceFactor: 144
-statistics: {
    currentServerTime: "2015-01-27T17:41:57.105+0000"
    firstLocatedTime: "2015-01-27T17:14:50.903+0000"
    lastLocatedTime: "2015-01-27T17:41:51.654+0000"
   -rssiList: [4]
-rssiList: [4]
   -0:  {
        apMacAddress: "20:3a:07:07:6d:b0"
        band: "IEEE_802_11_B"
        slot: 0
        rssi: -72
        antennaIndex: 1
        lastHeardInSeconds: 1
    }
   -1:  {
        apMacAddress: "2c:3f:38:58:51:90"
        band: "IEEE_802_11_B"
        slot: 0
        rssi: -73
        antennaIndex: 0
        lastHeardInSeconds: 1
    }
   -2:  {
        apMacAddress: "20:3a:07:07:6d:b0"
        band: "IEEE_802_11_B"
        slot: 0
        rssi: -76
        antennaIndex: 0
        lastHeardInSeconds: 1
    }
   -3:  {
        apMacAddress: "2c:3f:38:58:51:90"
        band: "IEEE_802_11_B"
        slot: 0
        rssi: -81
        antennaIndex: 1
        lastHeardInSeconds: 1
    }
```

```
        maxDetectedRssi: null
    }
    historyLogReason: null
    geoCoordinate: null
    networkStatus: "ACTIVE"
    changedOn: 1422380511654
    ipAddress: null
    userName: ""
    ssId: ""
    sourceTimestamp: null
    band: "UNKNOWN"
    apMacAddress: ""
    dot11Status: "UNKNOWN"
    manufacturer: "Intel"
  -areaGlobalIdList: [8]
        0:  21
        1:  23
        2:  3
        3:  2
        4:  1
        5:  24
        6:  25
        7:  18
    detectingControllers: "173.37.206.31"
    bytesSent: 0
    bytesReceived: 0
    guestUser: false

.84:  {
    macAddress: "00:ee:bd:a1:9f:ab"
  -mapInfo: {
        mapHierarchyString: "Richardson_TX_7508
        floorRefId: "-5970138651793817391"
      -floorDimension: {
            length: 259
            width: 419
            height: 10
            offsetX: 0
            offsetY: 0
            unit: "FEET"
        }


84:  {
   macAddress: "00:ee:bd:a1:9f:ab"
  -mapInfo: {
      mapHierarchyString: "Richardson_TX_75082>Cisco_Building_5>2250_East_PGBT_First_Floor>Lab_Coverage_Area"
      floorRefId: "-5970138651793817391"
    -floorDimension: {
        length: 259
        width: 419
        height: 10
        offsetX: 0
        offsetY: 0
        unit: "FEET"
      }
    -image: {
        imageName: "domain_0_1410184557303.png"
        sourceFile: null
*
*
*
```


Example of Single Client Location Information (GET)

```python
#using python 3.4.x

from http.client import HTTPSConnection
from base64 import b64encode

#Create the  https connection
c = HTTPSConnection("173.37.206.202")

#encode as Base64
#decode to ascii (python 3 stores as byte string, we don't want that as we need to pass ascii value for auth)

#userAndPass = b64encode(b"admin:!wnbuTme1").decode("ascii")
usernamepasssword = b64encode(b"admin:!wnbuTme1").decode("ascii")
headers = { 'Authorization' : 'Basic %s' %  usernamepasssword}

#connect and ask for resource
c.request('GET', '/api/location/v1/clients/28:b2:bd:33:71:e2', headers=headers)

#reponse
res = c.getresponse()

data = res.read()

#print the results
print (data)
```

```
{
  macAddress: "28:b2:bd:33:71:e2"
 -mapInfo: {
     mapHierarchyString: "Richardson_TX_75082>Cisco_Building_5>2250_East_PGBT_First_Floor>Lab_Coverage_Area"
     floorRefId: "-5970138651793817391"
    -floorDimension: {
        length: 259
        width: 419
        height: 10
        offsetX: 0
        offsetY: 0
        unit: "FEET"
     }
    -image: {
        imageName: "domain_0_1410184557303.png"
        sourceFile: null
        zoomLevel: 0
        width: 0
        height: 0
        size: 0
        maxResolution: 0
        colorDepth: 0
     }
     tagList: [0]
  }
 -mapCoordinate: {
     x: 168.33615
     y: 45.074123
     unit: "FEET"
  }
  currentlyTracked: true
  confidenceFactor: 112
 -statistics: {
     currentServerTime: "2015-01-27T17:52:53.982+0000"
     firstLocatedTime: "2015-01-27T17:14:50.903+0000"
     lastLocatedTime: "2015-01-27T17:51:51.636+0000"
    -rssiList: [4]
       -0:  {
            apMacAddress: "20:3a:07:07:6d:b0"
            band: "IEEE_802_11_B"
            slot: 0
            rssi: -66
            antennaIndex: 0
            lastHeardInSeconds: 61
        }
```

```
      -1:  {
             apMacAddress: "20:3a:07:07:6d:b0"
             band: "IEEE_802_11_B"
             slot: 0
             rssi: -71
             antennaIndex: 1
             lastHeardInSeconds: 61
           }
      -2:  {
             apMacAddress: "2c:3f:38:58:51:90"
             band: "IEEE_802_11_B"
             slot: 0
             rssi: -72
             antennaIndex: 0
             lastHeardInSeconds: 1
           }
      -3:  {
             apMacAddress: "2c:3f:38:58:51:90"
             band: "IEEE_802_11_B"
             slot: 0
             rssi: -75
             antennaIndex: 1
             lastHeardInSeconds: 1
           }
    maxDetectedRssi: null
  }
  historyLogReason: null
  geoCoordinate: null
  networkStatus: "ACTIVE"
  changedOn: 1422381111636
  ipAddress: null
  userName: ""
  ssId: ""
  sourceTimestamp: null
  band: "UNKNOWN"
  apMacAddress: ""
  dot11Status: "UNKNOWN"
  manufacturer: "Intel"
- areaGlobalIdList: [8]
    0:  21
    1:  23
    2:  3
    3:  2
    4:  1
    5:  24
    6:  25
    7:  18
  detectingControllers: "173.37.206.31"
  bytesSent: 0
  bytesReceived: 0
  guestUser: false
```


TIP: Joining the Cisco Devnet will give you the best support for creating applications that access the MSE REST API. Please check out https://developer.cisco.com/site/devnet/home/index.gsp