Cisco CMX REST API スタートアップ ガイド

Cisco CMX モビリティ サービスの概要

CMX モビリティ サービスを使用することで、開発者は Wi-Fi とデバイス MAC アドレスによってデバイスのロケーションを利用できるようになります。クーポンやプロモーションなどのプッシュ通知をデバイスに提供することで、カスタマー エクスペリエンスの向上が期待できます。さらに、様々なクライアント ロケーション ベースのサービス ソリューションを開発し、エンド デバイス ユーザや施設内のオペレータが利用できる数多くのアプリケーションを提供することもできます。

このソリューションは、Wi-Fi フィールドで動作するモバイル デバイスを検出、接続、およびエンゲージするためのモバイル ソフトウェア ソリューションからなる画期的なスイートです。各モバイル ソフトウェア サービスが連携して、クライアントとそのエンド ユーザが現実の様々な状況下で便利に利用できるように設定できる、包括的なソリューションを形成します。

CMX モビリティ サービス Restful API により、これらを様々な言語、プラットフォーム、フレームワークにわたって使用できます。API を介して、Wi-Fi ネットワークから収集されたリアルタイムのインテリジェンスを活用したアプリケーション ソリューションを開発することで、ロケーション、温度、ユーザのアベイラビリティ、モバイル デバイスのアセットなどのリアルタイムのコンテキスト情報を通じて、ユーザやユーザのデバイスとのより効果的な対話を実現できます。

このガイドでは、CMX モビリティ サービス Restful API について説明します。Cisco CMX モビリティ アプリケーションの詳細については、http://www.cisco.com/c/en/us/td/docs/wireless/mse/8-0/CMX-ConnectandEngage-Mobile-SDK/guide/Cisco-CMX-ConnectEngage-Mobile-SDK-Config-Guide.html を参照してください。

（注） CMX モビリティ サービスのすべての機能およびガイドに関する情報は、Cisco DevNet サイトで閲覧できます。

REST API のサポート

Cisco Connect Mobile Experiences（Cisco CMX）が提供する簡素化された Representational State Transfer（REST）API は、様々な方法で使用できます。RESTful API には次の利点があります。

- ステートレス REST アーキテクチャに基づいています。
- 情報は十分に理解されている HTTP/HTTPS プロトコル経由で配信されます。
- 簡素化された設計で、リソース URI を簡単に理解することができます。通常、リソース URI の内容は一目瞭然です。
- 標準化されたライブラリにより、ユーザが要求するフォーマットで自動的に応答を送信してコンテンツをネゴシエーションできます。
- データ モデルの簡素化、古い拡張性のない API の廃止、階層依存性の排除といった機能向上に取り組むことができるようにします。

REST API では次の HTTP メソッドを使用します。

- GET：リソースの表現を取得する HTTP メソッド。
- PUT：URI でエンティティを保管する HTTP メソッド。このメソッドは、新しいエンティティの作成や既存のエンティティの更新に役立ちます。
- POST：URI にあるリソースを要求し、指定されたエンティティで何らかのアクションを実行する HTTP メソッド。
- DELETE：リソースの削除を要求する HTTP メソッド。

Cisco CMX では、REST API に固有の更新済みドキュメントを提供しています。これらの資料に、REST API クエリーの名前、メソッド、構造が説明されています。REST API ドキュメントについての詳細は、http://mse-ip-address/apidocs/を参照してください。

REST API ドキュメントは、特定の機能に基づく 5 つの論理カテゴリに分類されています。API の分類は次のとおりです。

- コンフィギュレーション API：Cisco CMX をプログラムによって設定するための API。
- ロケーション API：Cisco CMX からクライアントのロケーション データをプログラムによって取得するための API。
- 分析 API：Cisco CMX から分析データをプログラムによって取得するための API。
- 接続 API：ユーザ セッション情報を見つけるための API。
- プレゼンス API：ビジターのプレゼンス データを見つけるための API。

REST API との対話

Cisco MSE にクエリーを送信するには、ユーザ名とパスワードが必要です。これらのクレデンシャルは、Base64 でエンコードされた文字列を作成するために必要となります。認証に成功すると、要求が HTTP メソッド（GET、PUT、POST、DELETE）によって送信されます。

REST API のワークフローは次のとおりです。

1. 要求を開始する前に、クライアント側から Authorization ヘッダーを使用して認証が送信されます。
2. ユーザ名とパスワードのクレデンシャルを結合して、文字列「username:password」を作成します。
3. 作成された文字列リテラルを Base64 でエンコードします。
4. エンコードされた文字列の前に、認証方式（スペースおよび「Basic」という文字列）を追加します。

たとえば、ユーザ エージェントがユーザ名として「MyUsername」、パスワードとして「MyPassword」を使用している場合、次のヘッダーが作成されます。

- Authorization: Basic QxhZGluOnNlc2FtIG9wZW4=
- 認証が宛先のルート URL（例：https://<mseip>/api/contextaware）に送信されます。

以下に、Base64 でエンコードされた認証文字列を作成する Python 2.7.x の例を示します。

```
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)
```

REST API と対話するには、次の 3 つの方法があります。

- [Try It] メソッド
- REST クライアント プラグイン（Google Chrome または Mozilla Firefox ブラウザでサポートされています）
- プログラミング言語の使用

[Try It] メソッドの使用

Cisco CMX に同梱されている REST API ドキュメントには、ブラウザからアクセスできます。Cisco CMX REST API ドキュメントのページは、サーバ上でホストされています。REST API のほとんどには、[Try It] 機能があります。

ステップ 1 Cisco CMX サーバの IP アドレスまたは DNS 名に続けて `/apidocs` を入力します。たとえば、*http://yourcmxIP/apidocs* と入力します。
apidocs の Cisco CMX ホーム ページが表示されます。

ステップ 2 表示されている利用可能ないずれかのサービスの名前に対応する [View Details] をクリックします。
それぞれのメソッドが、大まかな API カテゴリに分類されています。

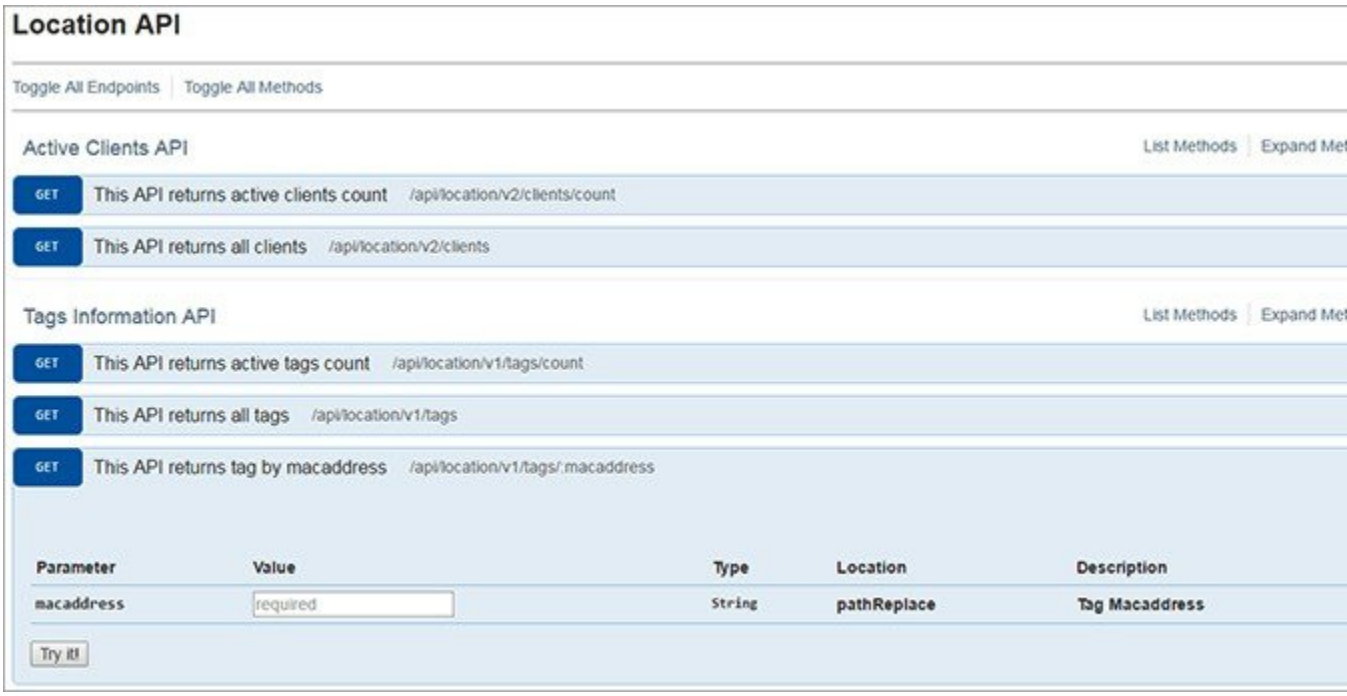
ステップ 3 ユーザ名とパスワードを入力して、API にアクセスします。

図 *1*：*Cisco CMX* クレデンシャル

管理者は、REST API にアクセスするためのユーザ名とパスワードを設定できます。

ステップ 4 API メソッドを選択すると、そのカテゴリのメソッド一覧が表示されます。

図 *2* : ***API*** メソッド



ステップ 5 個々のメソッドをクリックして展開するか、メソッドの右側にある [Expand Methods] ボタンをクリックして展開し、選択したメソッドに分類されるすべてのメソッドを表示します。

（注） API メソッドごとに URL が割り当てられています。コード内では、この URL を使用して対応する API にアクセスできます。この URL は、POSTMAN や Advanced REST Client などのサードパーティ製 REST クライアントで使用することもできます。

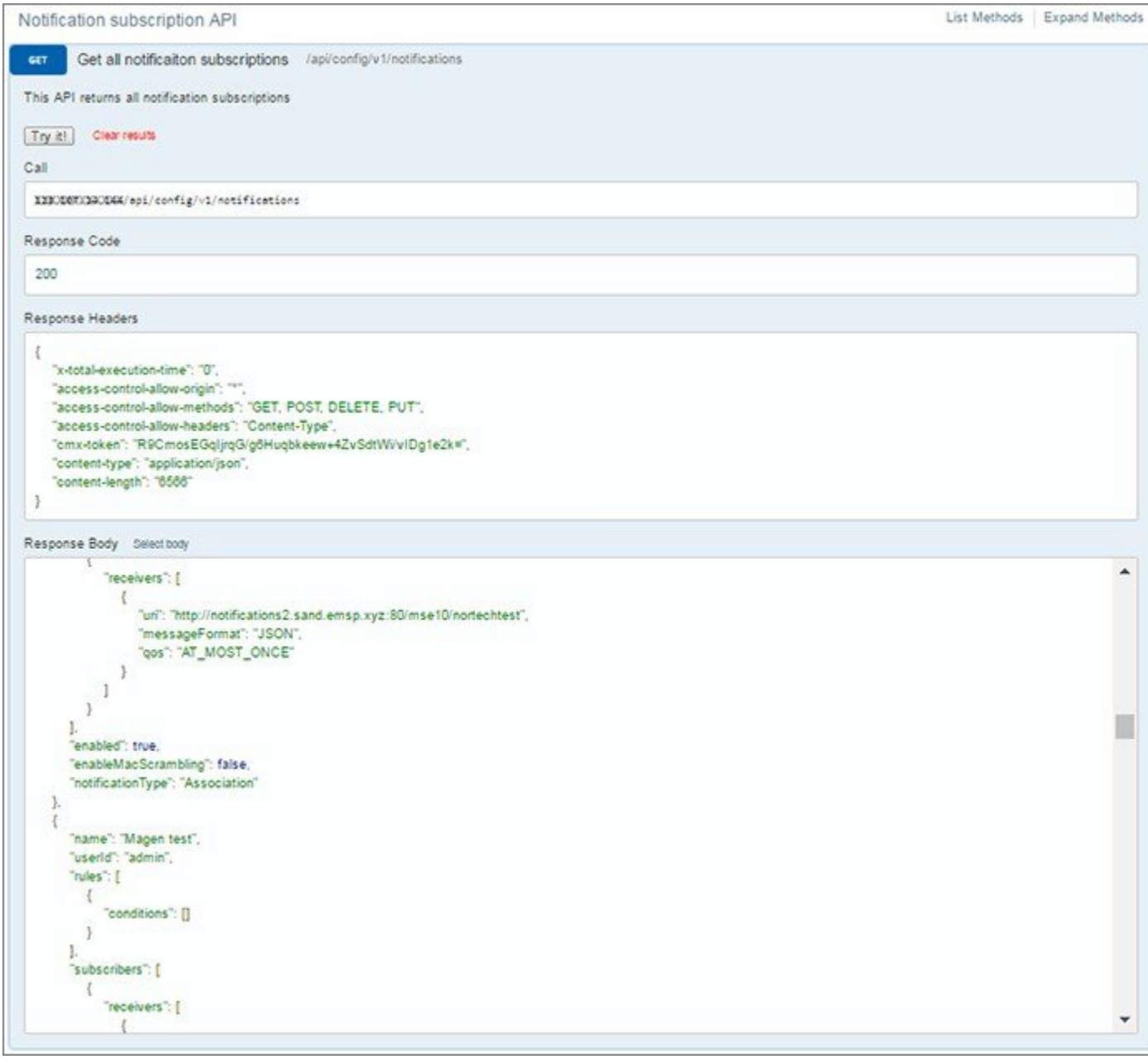
ステップ 6 選択した API に対応する [Try It] をクリックします。

図 *3* : ***[Try It]*** 機能

API の詳細が表示されます。コール詳細、応答コード、応答ヘッダーおよび応答本文を表示できます。

ステップ 7 出力を確認します。

図 *4* : ***[Try It]*** 出力



出力には、次のパラメータが表示されます。

- [Call]：クエリーの構造を表示します。構文をテストするには、クエリーをウェブ ブラウザにコピーアンド ペーストします。
- [Response Code]：クエリーに関する情報を提供します。クエリーが成功したかどうかが示されます。
- [Response Headers]：要求された情報を表示します。
- [Response Body]：要求された情報を表示します。このパラメータは、PUT 形式を理解するのに役立ちます。

ヒント
- [Response Body] を使用して、PUT 形式を理解します。
- コールの詳細

サードパーティ製 REST クライアントの使用

Cisco CMX に対する API コールおよびクエリーの構造をテストするには、最新バージョンの Web ブラウザ（Google Chrome、Mozilla Firefox）対応の REST クライアント プラグインを使用してください。たとえば、Advanced REST Client、Postman、REST Client などのプラグインを使用します。使用する REST クライアントは自由に選択できます。

次の例では、Chrome の Advanced REST Client プラグインを使用しています。

（注） REST クライアントの使用方法の詳細は、選択した REST クライアントのプラグインおよびアプリケーションに関する手順を参照してください。

例：通知定義（PUT）

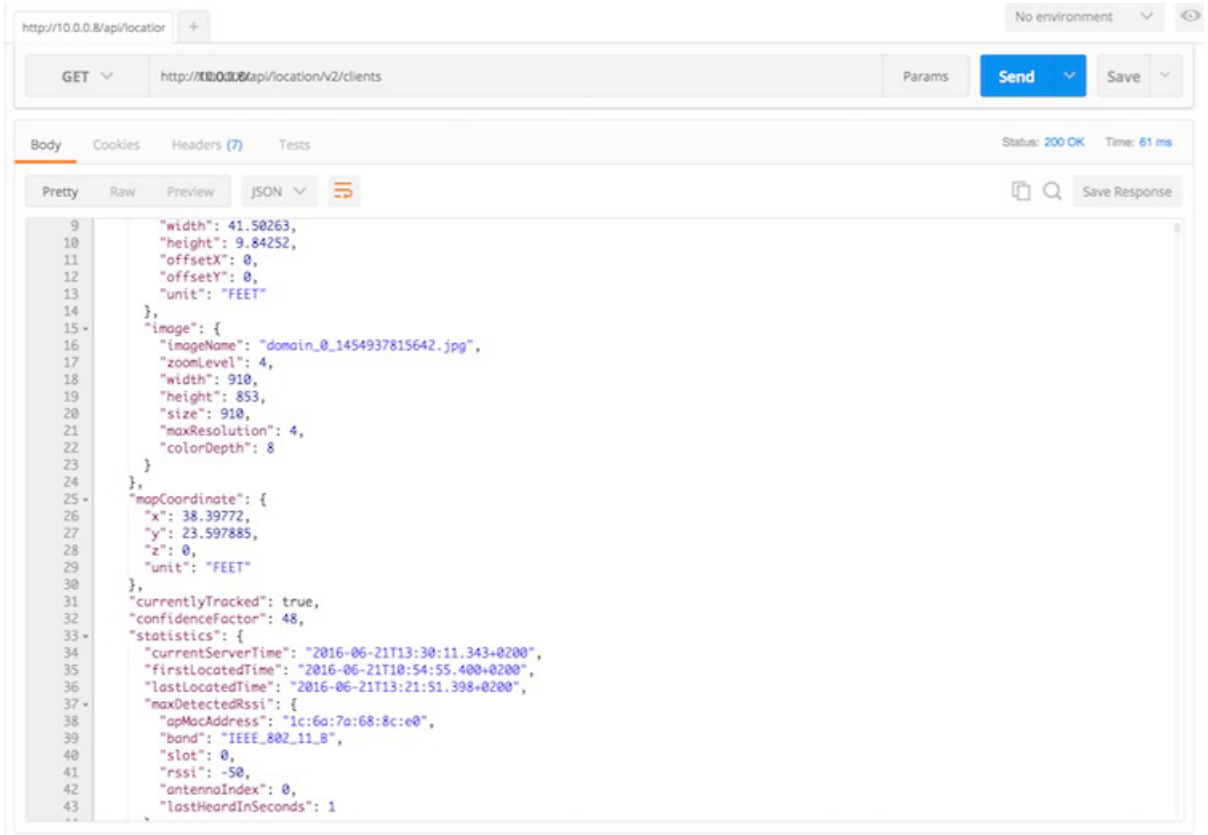
図 *5*：通知定義（*PUT*）

例：通知定義（GET）

図 *6*：通知定義（*GET*）

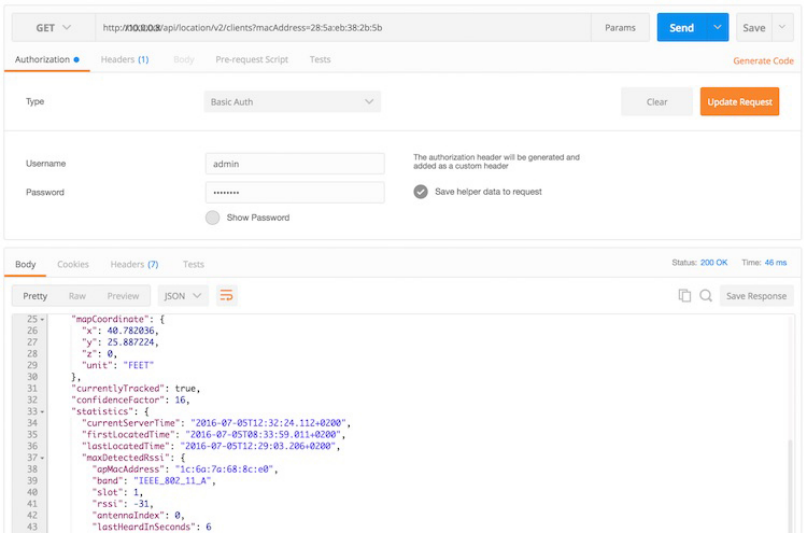
例：すべてのクライアント（GET）

図 *7*：すべてのクライアント（*GET*）



例：MAC アドレス別のクライアント履歴（GET）

図 *8*：*MAC* アドレス別のクライアント履歴（*GET*）



プログラミング言語の使用

API コールの構造をテストするには、Python プログラミング言語を使用します。Python は、データセンター環境でのリソースの設定および操作に広範に使用されている単純なプログラミング言語です。Python 2.7. と Python 3.4.x の両方が使用されます。どちらの Python バージョンが使用されているかは、コード内のコメントで示されます。

（注） 以下の例は単純なものなので、フォーマット設定やその他の機能は実装されていません。

ヒント CMX REST API にアクセスするアプリケーションの作成について最良のサポートを得るには、Cisco Devnet に参加してください。詳細については、https://developer.cisco.com/site/devnet/home/index.gspを参照してください。

例：クライアント ロケーションの一覧（GET）

図 *9*：クライアント ロケーションの一覧（*GET*）

```
#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)
```

354663


```
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]
```


354664

```
-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
   }
```


354665


```
    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"
    }
```


354666

```
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
```


354667

例：単一のクライアント ロケーション情報（GET）

図 *10*：単一のクライアント ロケーション情報（*GET*）

```
#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)
```

354668

```
{
  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
        }
```


354669

ヒント　Cisco CMX REST API にアクセスするアプリケーションを作成する際に最良のサポートを得るには、Cisco Devnet に参加することをお勧めします。詳細については、https://developer.cisco.com/site/devnet/home/index.gspを参照してください。

例：ロケーションのセットアップ（POST）

204 を返します。コンテンツはありません。

例：通知サブスクリプション（GET）

Cisco MSE では、すべてのクライアントのすべてのアクティビティについて、リアルタイム ストリームを宛先に送信できます。これらの通知はロケーション更新イベントのスーパーセットです。通知機能を有効にするには、PUT API および GET API を使用します。

図 *11*：通知サブスクリプション（*GET*）

```
#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)
```

354661

例：通知サブスクリプション（PUT）

204 を返します。コンテンツはありません。

関連資料

『*Cisco Connected Mobile Experiences REST API Guide, Release 10.2*』に、Cisco CMX ソリューションのすべての REST API が記載されています。詳細については、https://www.cisco.com/c/en/us/td/docs/wireless/mse/10-2/api/b_cmx_102_api_reference.htmlを参照してください。