Table of Contents

LAB 01: Invoke Webex REST APIs from the interactive documentation

https://developer.cisco.com/learning/lab/collab-spark-doc-tour-itp/step/1

Tour Webex interactive documentation

Note: Webex Teams is now known as Webex.

In this lab we will navigate through the Webex for Developers portal and experiment the Webex REST API.

Objectives

- Navigate through the Webex for Developers portal
- Test Webex REST APIs from the interactive documentation

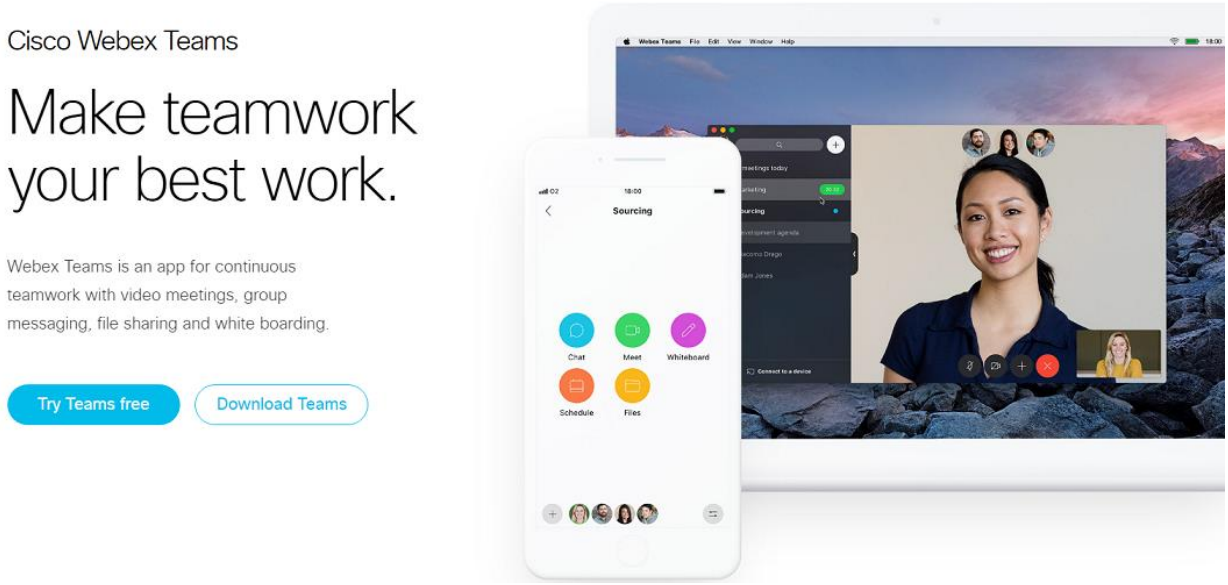
Prerequisites

To successfully complete this lab, you will need a Webex account.

- Navigate to Webex Sign up page and create an account by entering your email address.

Step 1. What is Webex?

Webex is an app for continuous teamwork with video meetings, group messaging, file sharing and white boarding.



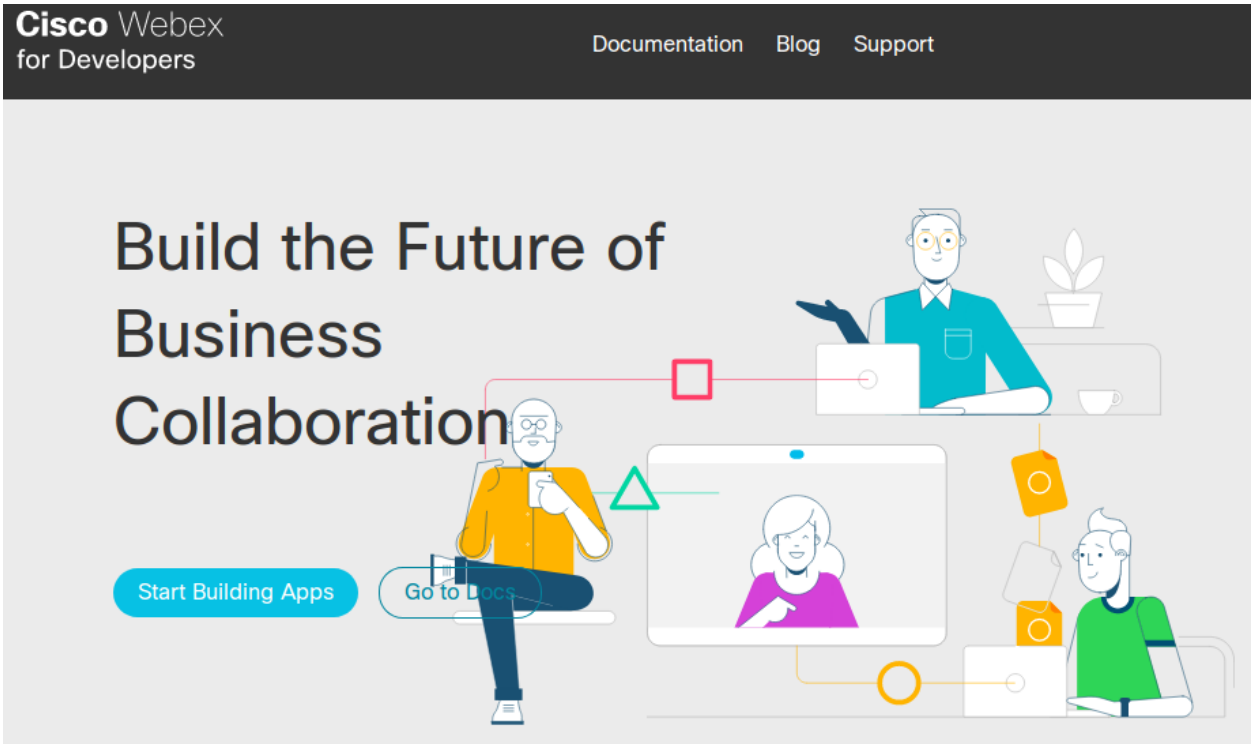
Besides being a communications tool, it also allows developers to easily integrate solutions with Webex via the REST API. The APIs may be used to do many things such as adding a Webex messaging features to an application user interface, or automate sending messages to Webex meetings based upon business system or real-world events.

Application developers integrating with Webex must register their applications via the Webex for Developers portal - defining the application name, permissions, and OAuth2 redirect URL (more on this later). During registration the system generates a Client ID and Client Secret pair, which are later used by the application to access the Webex OAuth2 authentication service.

In the next step we will learn what Webex has to offer to developers.

Step 2. Webex for Developers

In this step we will help you navigate through the Webex for Developers page, and find exactly what you are looking for. Now let's delve into it.



1. Open your web browser and navigate to https://developer.webex.com/. If you have already created a Webex Account click on the **Log In** button and authenticate. If you do not have an account click on the **Sign Up** button and create one
2. After successful authentication, click on the **Go to Docs** button in the middle of the screen, then on **Getting Started** from the left navigation
3. Now lets examine the page. On the top of the page we see several tabs. The most important ones are the:

    - **Documentation** - documentation to successfully develop an application
    - **Blog** - page where people blog about their achievements in Webex
    - **Support** - information on to get help

By clicking on your avatar in the upper right corner, you can access the **My Webex Apps** menu entry, and create applications for Webex

4. Next we will examine the left panel of the Documentation's page. It consist of three sections. They are:
    - **Overview** - high level intros and discussion around the general platform, integrations, bots, etc.
    - **REST API** - reference docs and guides/tutorial for using the Webex messaging APIs
    - **SDKS** - tools, packages, libraries and widgets enabling you to embed Webex collaboration capabilities (including voice and video) into your own apps

We encourage you to read the following pages as they provide information necessary for you to start developing applications for Webex:

- Getting Started
- Integrations (OAuth)
- and Bots pages

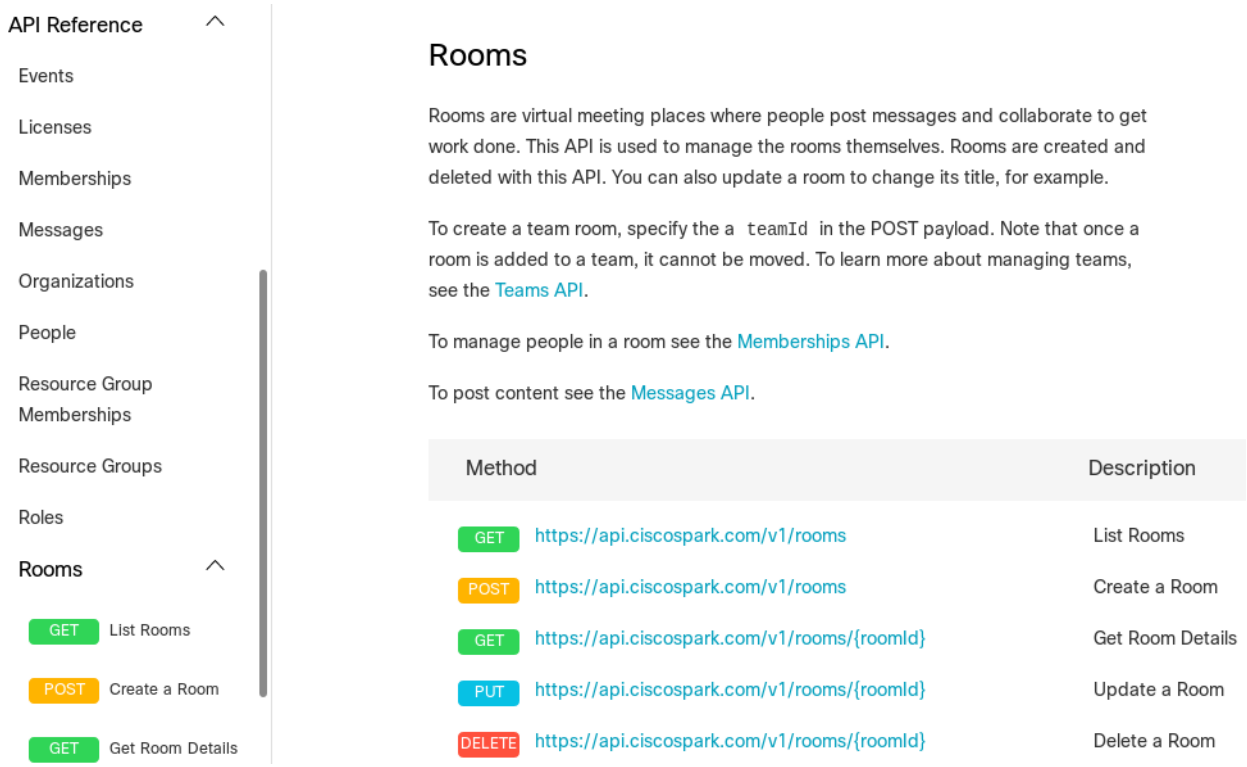
In the next section we will introduce you to the messaging API reference section and will show how to make REST APIs requests right from your web browser.

Webex API reference documentation

In this section we will explain how to read the Webex REST APIs and will show you how to make a call using the Web UI.

We will pick one API call and explain its moving pieces. Everything we show and explain applies to the other APIs.

1. Lets pick an API from the provided list under **REST API | API Reference**. For this lab, we will choose the Rooms API.



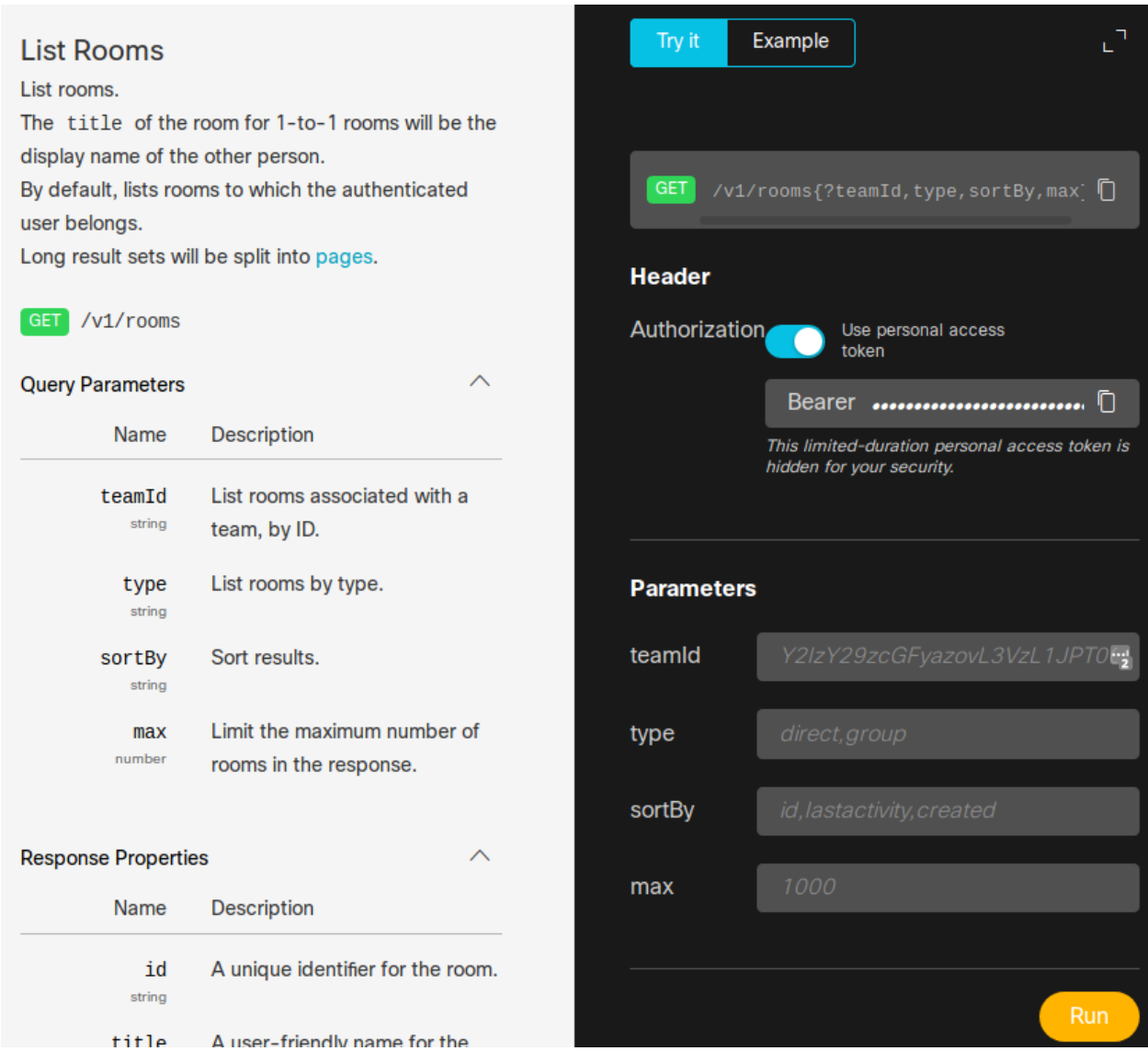
2. As you can see, the main page for the API provides a brief description of what is possisble to achieve with this API. Below the description are the API methods. Each methods does different type of job. Lets briefly go over them:

    - GET - retrieves an information
    - POST - posts information to the server
    - PUT - updates existing information

- DELETE - deletes the information

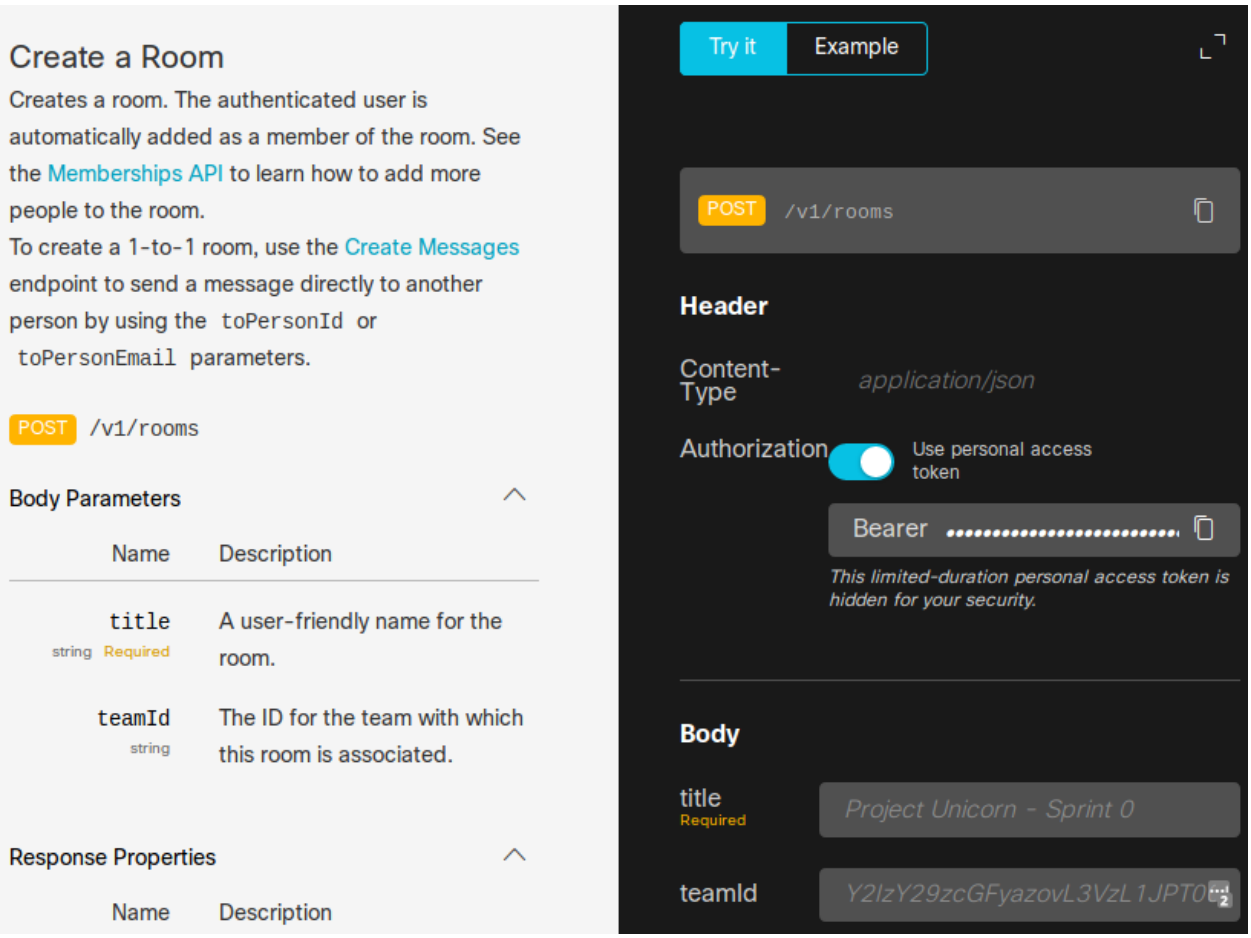
3. Now, lets examine one of the methods. Lets click the GET https://webexapis.com/v1/rooms method. As you can see it provides detailed information of what can be done with this method. Let break them down and explain them:

    - **GET** - API method which needs to be used to get a result from the server
    - https://webexapis.com/v1/rooms - URL against which the call is made
    - **Query Parameters** - optional information that can be passed along while making the call. Think of the query parameter as a filter
    - **Response Properties** - descriptions of each of the fields returned in the API response
    - **Response Codes** - after each call server returns a code which act like a status update. Each code has its own meaning



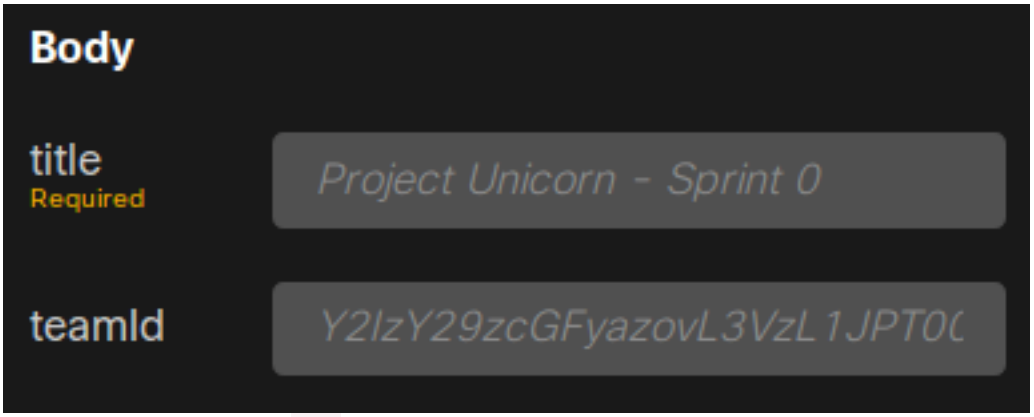
Now you know how to read the API docs and use them in your applications. But that is not all that the Webex for Developers page has to offer. It is also possible to make an API call directly

from the website. Is that not great? You can easily craft a call and see the result immediately. For this demonstration click on the create a room from the API ref on the left, we will use [POST https://webexapis.com/v1/rooms](https://webexapis.com/v1/rooms) API method.



1. In the **Create a Room** page, make sure the **Try It** mode is enabled: 

2. The page will change its look and will offer new fields were you can input values. For this particular method, we have one field which is required; that is the title under the **Body Parameters** section.



3. Now lets populate the title parameter with a value. The provided value will be used as a room's name when it is created
4. Now you can make the request by hitting the **Run** button
5. Immediately after, you will see a response from the Webex cloud platform in the lower/right. It provides us with very useful information like the response code (200 OK / success) and detailed information about the created room in JSON format. From the output we get very useful data like:

    - id - unique identifier assigned to the room.
    - title - name of the room
    - type - group type. This value can be either direct (1-to-1) or group
    - creatorId - ID which belongs to the creator of this room


```
6. {
7.  "id": "Y2lzY29zcGFyazovL3VzL1JPT00vMDljNGVhMjAtZTQyZC0xMWU2LTk1YmUtYTMyMmQ1ODQ2OWVk",
8.  "title": "DevNet Event",
9.  "type": "group",
10. "isLocked": false,
11. "lastActivity": "2017-09-28T01:07:57.018Z",
12. "creatorId": "Y2lzY29zcGFyazovL3VzL1BFT1BMRS8zNDhhOWY4Zi01NDc5LTQxODMtODY1YS1hNmU4MTkzMDUxZGU",
13. "created": "2017-09-28T01:07:56.994Z"
14. }
```


15. Now, open your Webex client. You should see that your newly created space is present and ready for you to post a message into it.

Challenge (Optional)

We have an interesting challenge for you: using nothing but the Webex for Developers interactive docs, post a message in the space you have just created.

Hint

- Using the GET method for the Rooms API, obtain the room ID assigned to the room you created
- Using the POST method for the Messages API, post a message to the room.

  Note: roomId and text parameters have to have a value for you to successfully make the request

Congratulations! You have successfully completed this lab!

LAB 02: Calling REST APIs from Python

Note: Webex Teams is now known as Webex.

In this lab we will show you how to make REST API calls to GET (read) and POST (create) Webex rooms and messages using Python programming.

Objective

- Review REST API components and the Webex interactive docs
- Build GET/POST calls using Python to automate room and message creation

Prerequisites

Python - To run the code samples, you need to have Python 3 installed on your PC.

See **How to Set up Your Own Computer** above for installation instructions.

Requests library - These code samples use the Python Requests Library to simplify making REST API calls. We'll install this library into the Python virtual environment used to test the code.

Webex account - If you do not have an account yet, go to Webex for Developers page, click on the **Sign Up** button and create a free account.

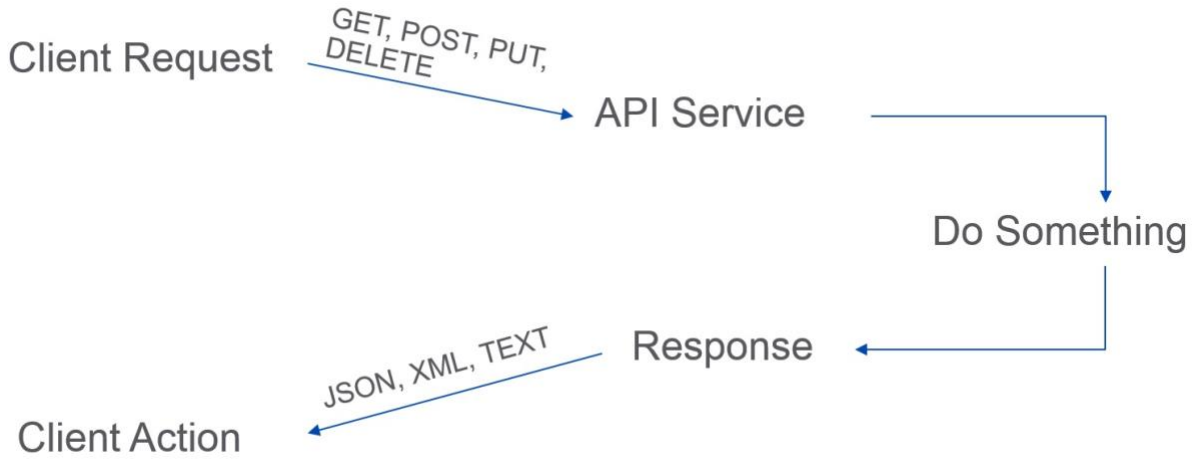
What is a REST API?

You may skip to the next page if you are familiar with REST APIs.
REST (Representational State Transfer) is an architecture style for designing networked applications. A REST web service is a web service that is as easy to call as making an HTTP request.

An API (Application Programming Interface) is a way for two pieces of software to talk to each other.

If we combine above two definitions then we will get a very simple explanation of what a REST API is:

An API utilizing REST architecture to retrieve or manipulate data on the server.
The image depicts how REST APIs works.



In the next few steps, we will show you how to create REST API calls using the Python scripting language.

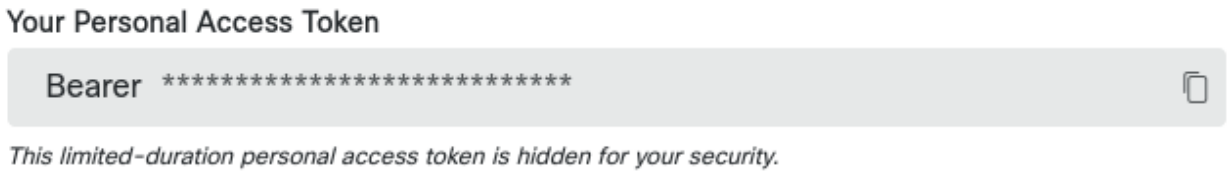
How to write a REST API GET request in Python

In this step we will explain how to write a Python script to make REST API calls. As was stated in the previous step, we will be using Webex REST APIs. So, without further ado, lets start learning!

You will need a Webex Teawms **Personal Access Token** in order to explore the Webex REST API:

1. Browse to to the Webex for Developers website
2. Log in
3. At the top of the page, select **Documentation**, then on the left under **REST API** click **Getting Started**.

   Scroll down to the **Accounts and Authentication** section, where you should see your "Personal Access Token":

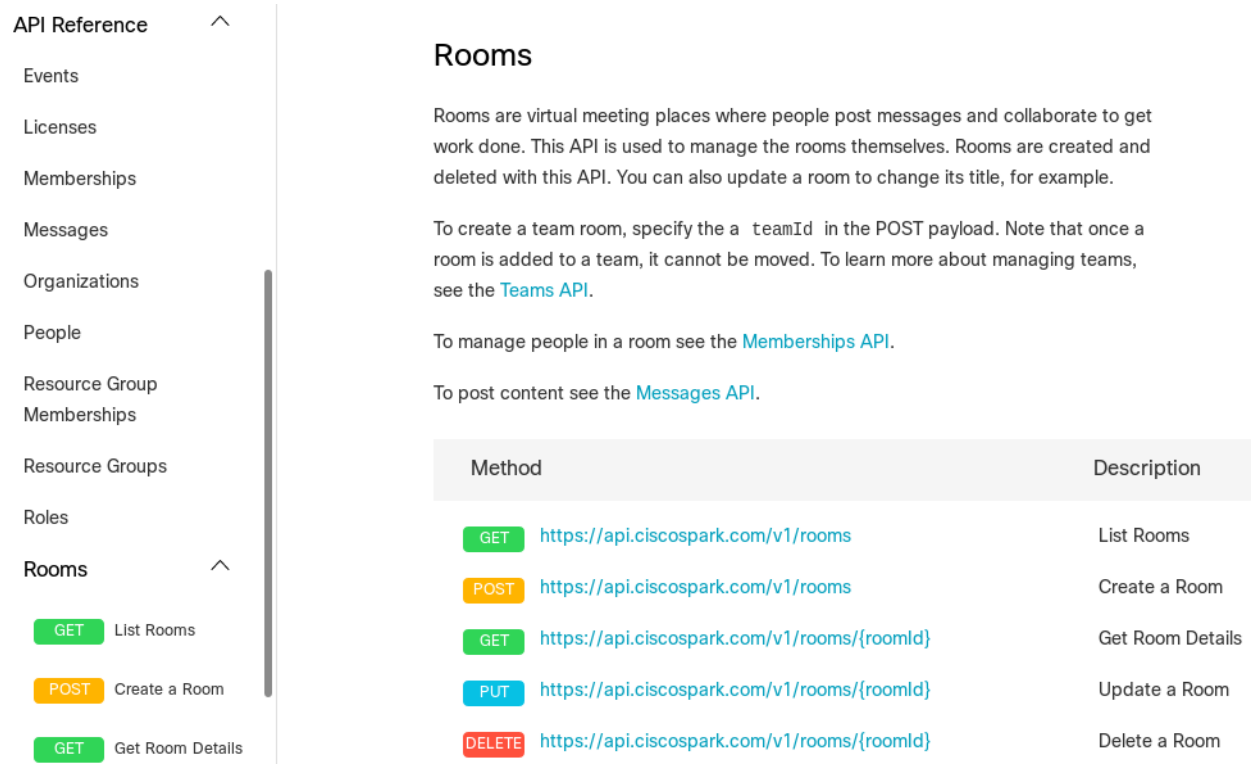
   Note: this personal token should be used only for experimentation and testing - do not use it in any production applications!
4. Click the **copy** icon to copy the token to your clipboard

Listing Webex rooms

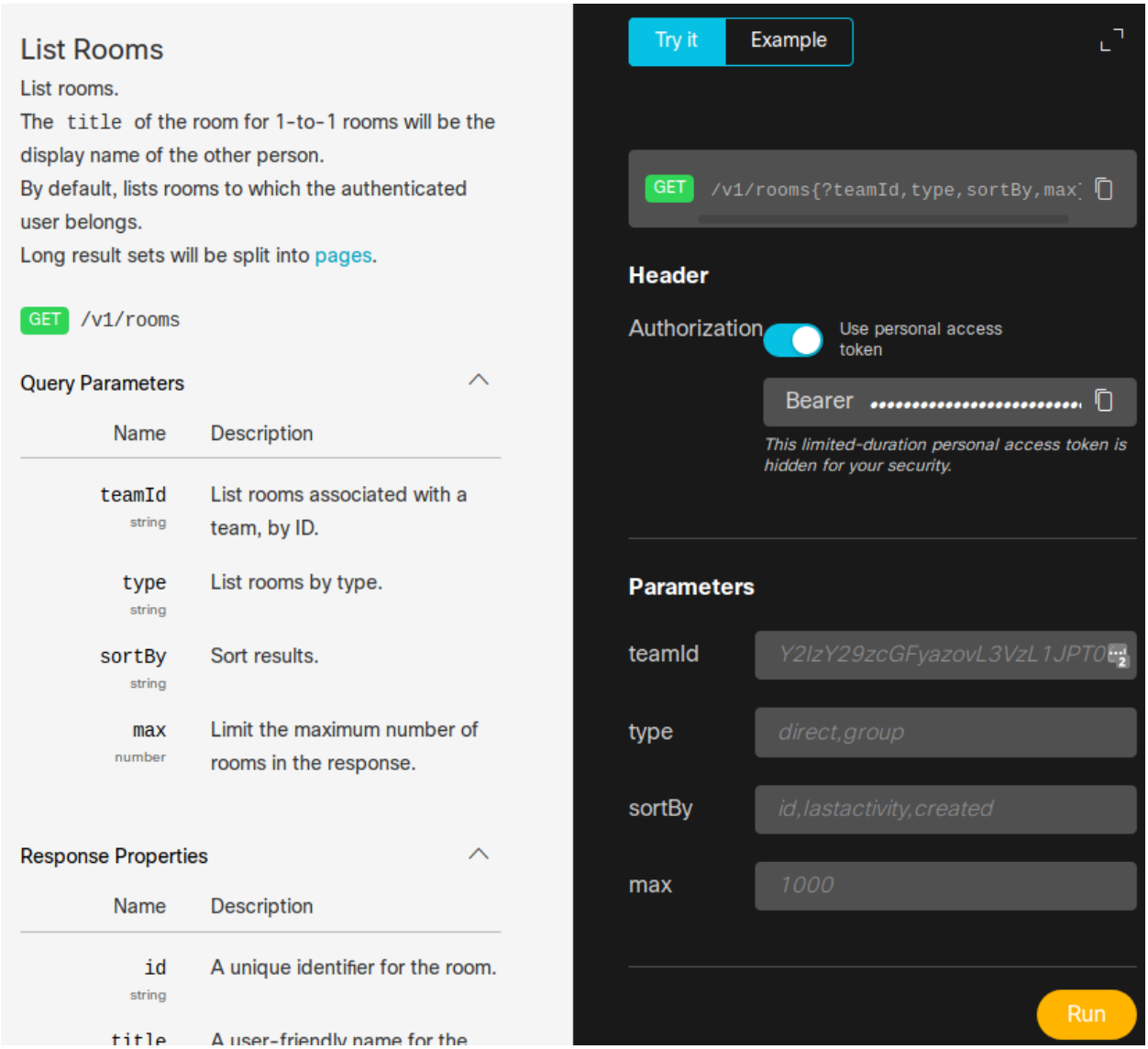
Now lets look into one of the REST API resources supported by Webex. For this step we will work with the /rooms resource and demonstrate how to use the **GET** and **POST** methods in a Python script.

The **GET** REST method is used to retrieve a listing or a single instance of a resource. Let's use the interactive docs to get a listing of your Webex rooms:

1. To access the Rooms API docs, browse to https://developer.webex.com then click on the **Documentation** link at the top.
2. Under **REST API** expand the **API Reference** section, then click on **Rooms**:



3. Click on the **GET** method link
4. Make sure the **Try it** toggle is enabled for interactive mode:



5. Leave the settings at their default and click on the **Run** button to execute the request.

   Examine the results.

Writing Python code

Note that the interactive doc mode provides all of the REST API details we'll need to code a Python version of this operation:

- **Method** - GET
- **URL** - https://webexapis.com/v1/rooms
- **Headers**
  - Content-type: application/json
  - Authorization: Bearer ACCESS_TOKEN
- **Parameters**

- teamId
- max
- type
- sortBy

First a few steps to set up our Python environment:

1. From a terminal, navigate to a suitable working directory and create/activate a Python virtual environment:

```
# For Windows use: py -3
python3 -m venv env
source env/bin/activate
```

5. Next install the Python requests package:

```
pip install requests
```

Next, let's write some code:

1. From your favorite text editor/IDE create a new source file
2. The first bit of code we need imports the Python requests helper library that will make coding to REST easier.

   Type in:

   ```
   import requests
   ```

3. Next, we will define a couple of global variables. These variables will hold the URL of the /rooms API, and your developer access token - enter these lines in:

```
apiUrl = 'https://webexapis.com/v1/rooms'
access_token = 'your_access_token'
```

   Be sure to replace your_access_token with the personal access token you copied earlier.
6. Now we will create a Python "dictionary" type variable which will hold the neessary HTTP headers for the API request.

   A dictionary is an associative array (also as a hash). Any key of the dictionary is associated (or mapped) to a value. The values of a dictionary can be any Python data type. So dictionaries are unordered key-value pairs.
   This variable will hold the information in key: value format and will be separated by commas if more than one entry exits. All of this will be enclosed in curly braces {}.

In our case Content-Type is a key and application/json is its value. Same applies to the Authorization and Bearer ACCESS_TOKEN portion:

```
httpHeaders = { 'Content-Type': 'application/json', 'Authorization': 'Bearer ' + access_tok
en }
```

7. We can add HTTP query parameters via another dictionary. These will be used to filter the items that the request will act on.

   For this **GET** request we would like to receive room details sorted by 'lastactivity', and with no more than 2 results returned:

```
queryParams = { 'sortBy': 'lastactivity', 'max': '2' }
```

Note: query parameters in a GET method act as a filters to the query and are optional.

Now, we got to the interesting part of the code, which is performing the API request itself.

As we said in the beginning, we will be using the requests library. This library contains Python methods such as get(), post(), put() and delete() which correspond to REST methods. Each method accepts certain arguments to successfully execute the request:

1. Here we provide the necessary arguments for the get() method, which we assembed before hand via the two dictionary variables above. The output of the request is directed to a response variable:

```
2. response = requests.get( url = apiUrl, headers = httpHeaders, params = queryParams )
```

   Lets break it down to pieces:

   - response - a variable which will hold the result of the call.
   - requests.get() - from the requests library use the get() method.
   - url = apiUrl - url the apiUrl variable which holds the Webex API location.
   - headers = httpHeaders - headers argument will use the httpHeaders variable contents (see above).
   - params = queryParams - params argument will use the queryParams variable contents.

3. Now we will use print() commands to output the returned HTTP status code and any response body text to the console:

```
4. print( response.status_code )
5. print( response.text )
```

Awesome! The code is ready. The final version should look something like this:

```
import requests

apiUrl = 'https://webexapis.com/v1/rooms'
access_token = 'your_access_token'

httpHeaders = { 'Content-Type': 'application/json', 'Authorization': 'Bearer ' + access_token }
queryParams = { 'sortBy': 'lastactivity', 'max': '2' }

response = requests.get( url = apiUrl, headers = httpHeaders, params = queryParams )

print( response.status_code )
print( response.text )
```

Running the script

1. Save your file as a Python script with name: list_rooms.py
2. To run the saved code, from your terminal run the Python executable passing the source file name as the argument:

```
3. python list_rooms.py
```

Great! In this step we showed you how to make a GET request with Python to list rooms. In the next step we will show you how to make a POST request to create a message.

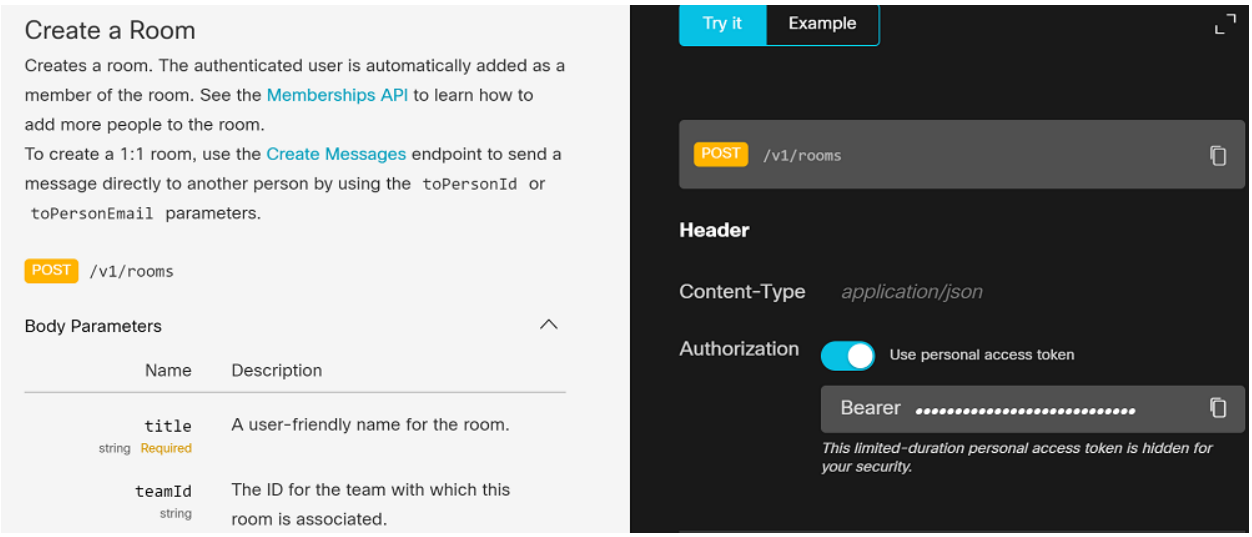
How to write a REST API POST request in Python

In the previous step, we explained how to code a Python script that makes a REST API call using the GET method. In this step we will do the same for the POST method, which has a few differences.

As in the previous step, we are going to use the Webex API reference as a guide to creating our Python script:

1. Browse to the **API Reference** / **Messages** / POST Create a Message page.

   This page shows information on how to create POST a message in a Webex room:



Here you can see all the REST elements we will need:

- **Method** - POST
- **URL** - https://webexapis.com/v1/messages
- **Headers**
  - Content-Type: application/json
  - Authorization: Bearer ACCESS_TOKEN
- **Request Parameters**
  - roomId
  - toPersonId
  - toPersonEmail
  - text
  - markdown
  - files

Lets start writing our code:

1. The import statement, global variables, and headers information will be the same as in the previous step:

```
import requests

apiUrl = 'https://webexapis.com/v1/messages'
access_token = 'your_access_token'

httpHeaders = { 'Content-Type': 'application/json', 'Authorization': 'Bearer ' + access_token }
```

Be sure to replace your_access_token with your personal access token

8. Now we have to define our request body - this will provide the details of the message we wish to create.

   The variable will have a dictionary data-type with key:value pairs separated by commas. We will use two key parameters toPersonEmail and text, i.e. the recipient's email address and the message text:

```
body = { 'toPersonEmail': 'tofrench@webex.bot', 'text': 'Hello' }
```

9. We will use the requests library's post() method.

   The post() method has an argument named json which takes a Python object, coverts it to JSON format and uses it for the request body:

```
response = requests.post( url = apiUrl, json = body, headers = httpHeaders )
```

10. Now we can print the response's HTTP status code, and the response text:

```
11. print( response.status_code )
12. print( response.text )
```

If we combine everything together, then our code will look like this:

```
import requests

apiUrl = 'https://webexapis.com/v1/messages'
access_token = 'your_access_token'
httpHeaders = { 'Content-type': 'application/json', 'Authorization': 'Bearer ' + access_token }

body = { 'toPersonEmail': 'tofrench@webex.bot', 'text': 'Hello' }

response = requests.post( url = apiUrl, json = body, headers = httpHeaders )

print( response.status_code )
print( response.text )
```

Great our code is ready for a test run:

1. Save the file as `post_message.py`
2. From the terminal, run the code by using the Python comamnd:

```
3. python post_message.py
```

After the script finishes its job, open your your Webex client - you should see your message has been sent to the intended recipient (an English-to-French translation bot.)

Challenge time: Let's Markdown to business

If you're up for a challenge, let's style the Webex message with a little bit of "Markdown" formatting syntax.

According to Wikipedia:

Markdown is a lightweight markup language with plain text formatting syntax designed so that it can be converted to HTML and many other formats using a tool by the same name.
So basically, markdown allows us to easily style text. Luckily for us, Webex speaks markdown, so we will use it to get our message across, with some emphasis!

Note: to understand all the markdown styling options available, see the Formatting Messages section of the Webex API docs.
Let's give it a try by sending a few more messages:

1. Below your initial code from above, add a Python list type variable. You can use the following as an example - note it contains three separate markdown-formatted messages:

```
messages = [
    '**Warning!!!**',
    '_Warning!!!_',
    '[Danger, Will Robinson!!!](https://en.wikipedia.org/wiki/Lost_in_Space#Catchphrases)'
    ]
```

7. Next include a `for` loop that iterates through your list and sends a message each time it cycles through:

```
for message in messages:

    body = { 'toPersonEmail': 'gifbot@webex.bot', 'markdown': message }
    response = requests.post( url = apiUrl, json = body, headers = httpHeaders )
```

```
12.
13.     print( response.status_code )
14.     print( response.text )
```

15. Save/run your new code and check the results in your Webex client!

An example of what your new code might look can be seen below:

```
import requests

apiUrl = 'https://webexapis.com/v1/messages'
access_token = 'your_access_token'
httpHeaders = { 'Content-type': 'application/json', 'Authorization': 'Bearer ' + access_token }

body = { 'toPersonEmail': 'gifbot@webex.bot', 'text': 'Hello' }

response = requests.post( url = apiUrl, json = body, headers = httpHeaders )

print( response.status_code )
print( response.text )

messages = [
    '**Warning!!!**',
    '_Warning!!!_',
    '[Danger, Will Robinson!!!](https://en.wikipedia.org/wiki/Lost_in_Space#Catchphrases)'
    ]

for message in messages:

    body = { 'toPersonEmail': 'gifbot@webex.bot', 'markdown': message }
    response = requests.post( url = apiUrl, json = body, headers = httpHeaders )

    print( response.status_code )
```

```
print( response.text )
```

Congratulations! You have successfully completed this lab!