

An **API (Application Programming Interface)** is any interface that allows you to get data in a format that is designed for another application to use.

The CUMTD API gives us data about the CUMTD bus system (where are stops located, what routes to buses take) and about the current state of the CUMTD buses.

...CUMTD documentation: <https://developer.cumtd.com/>

The first part of working with an API is to find out how the APIs will communicate data with you. There are two major types of APIs:

- **RESTful (Representational State Transfer):** For an API to be considered a RESTful API, it must have the following properties:
 1. **Client-Server Architecture:** API requests from a client must be directed to a server (a single endpoint, like a URL)
 2. **Stateless:** Every request must be entirely self-contained. The server must not “save” any information about one request and have it impact the next request.
 3. **Catchable:** Data must have a valid-until or expiration timestamp, or must be defined as part of the protocol.

- **Publish/Subscribe:** For an API to be considered a Publish/Subscribe API, a client must connect to a server as a subscriber and receive updates as they are published by the server.

Example: An API to return the current temperature at a given location:

| RESTful | Publish/Subscribe |
|--|---|
| Client makes a request to get the current temperature and the current temperature is returned. | Client subscribes to the temperature “Pub-Sub” server to get temperature updates. |
| If the client wants to know if the temperature has changed, it must make another API request. | When the temperature changes, all subscribers receive a broadcast of the new temperature. |

The CUMTD is a **RESTful** API, so we will be making a request every time we want to get updated information. We will not get notified if the information changes until we make another request.

CUMTD Request Format

To send a CUMTD request, their documentation tells us:

The RESTful endpoints use the HTTP GET verb with query string parameters.

Get a CUMTD Key

Every request to CUMTD requires a person-specific API key. This “API key” is a standard practice in RESTful APIs to allow users to make only a set number of API requests per time period.

...CUMTD allows upwards of hundreds of requests an hour per user, so the rate limit is not something we need to worry about for our small little program.

Get your API Key:

Head to <https://developer.cumtd.com/> and scroll down to the “**Get an API Key**” section. With a name and e-mail, they’ll send you the key.

Once you have your key, go to your **CS 205 workbook** directory, go into the **static** directory (**NOT** inside of any experience/project), then into the **keys** directory, and open **cumtd.txt**. Paste the CUMTD API key from your e-mail into **cumtd.txt**.

API Method: GetDeparturesByStop

Get a list of real-time departures for a specific stop_id.

URL: <https://developer.cumtd.com/api/v2.2/json/GetDeparturesByStop>

Parameters:

| Key | Value |
|-----------------|--|
| key | Your API key, required |
| stop_id | CUMTD identifier for a stop, required (IU is Illini Union, IT is Illinois Terminal; see GFTS data) |
| route_id | Semi-colon separated list of CUMTD route identifies, optional |
| pt | Preview time in minutes between 0 and 60, defaults to 30 |
| count | Maximum number of departures to return, optional |

Puzzle #1:

Make an HTML GET request by hand for getting all of the buses that will be stopping at the Illini Union in the next 30 minutes. Test that URL by entering it into a web browser.

Making API Requests in Python

Making a request by hand is good, but having a computer do it for you is even better! For this, there are three things we need to accomplish:

1. Load our CUMTD API key from `../static/keys/cumtd.txt`
2. Construct the HTTP GET request
3. Read the HTTP response by CUMTD's API server

Part 1: Loading the CUMTD API key

You can load this file the same way you have loaded all the other files to date in Python. The path to this file is the tricky bit:

```
1 # Import the libraries we will use in this file
2 import urllib.parse
...
5
6 # Open and read the file storing the CUMTD key
7 myKey = open("../static/keys/cumtd.txt",
               "r").read().strip()
```

(These first 7 lines are provided for you in `compute.py`, the rest you'll type to help with Python.)

Part 2: Construction the HTTP GET request

Since HTTP GET is a widely used standard, the Python language contains functions to help us make our HTTP GET query string:

```
9 # Construct a Python dictionary for HTTP GET
10 parameters = {
11     "key": myKey,
12     "stop_id": "IU"
13 }
14
15 # Construct the full URL
16 url = "https://developer.cumtd.com/api/v2.2/json/" +
       "GetDeparturesByStop?" +
       urllib.parse.urlencode( parameters )
```

Part 3: Read the response

Using the same library, we are able to make the HTTP GET request and get back the response into Python as a String:

```
18 # Read the response JSON into `data`
19 with urllib.request.urlopen(url) as response:
20     data_str = response.read().decode("utf-8")
21     data = json.loads(data_str)
```

Puzzle #2:

Run `compute.py` to make an API request to CUMTD. Use `print` to print out the response to see what CUMTD gives us!

Building a Timetable Application

The timetable screens shown at the CUMTD bus stops shows the upcoming buses and the amount of time they're away from the current stop:

| | |
|--------------------|----------------|
| 5E Green: | 2 mins |
| 12W Teal: | 5 mins |
| 22S Illini: | 6 mins |
| 13N Silver: | 10 mins |

In order to build this, we need to accomplish only three things:

1. Loop through all the upcoming departures
2. Add relevant information about each departure into a new list

Part 1: Loop through the departures

Look at the data you received – what data do you want to loop through?

How do we access that data from the dictionary `data`?

Part 2: Construct the results list

As your looping through the data, you should construct a new list of dictionaries. Specifically, your output should follow the following format:

```
[
  { "route": "5E Green", "expected": 3 },
  { "route": "12W Teal", "expected": 5 },
  { "route": "22S Illini", "expected": 6 },
  { "route": "13N Silver", "expected": 10 },
]
```

Puzzle #3 + Midweek Homework:

Complete the `compute.py` and save the new list as `timetable.json`. Modify `viz.js` to display the information on the HTML page.