# Python Demo

Mark O'Neil

{% assign sluggedName = page.name | replace: '.md' %}

modified: {{ page.last_modified_at | date: '%b-%d-%y' }}

## Demo using Python

The rest demo script demonstrates authenticating a REST application, management and use of the authorization token, and creating, updating, discovering, and deleting supported Learn objects.

### Prerequisites

- You must register a developer account and application in the Developer Portal
- You must register your application in Learn
- You must also configure the script as outlined in the README for the project

This webapp allows you to:

- Authenticate
- Create, Read, and Update a Data Source
- Create, Read, and Update a Term
- Create, Read, and Update a Course
- Create, Read, and Update a User
- Create, Read, and Update a Membership
- Delete created objects in reverse order of create - membership, user, course, term, datasource.

All generated output is sent to the terminal console.

**This is not meant to be a Python tutorial. It will not teach you to write code in Python. It will, however, give a Developer familiar with Python the knowledge necessary to build a REST-based Web Services integration.**

### Assumptions

This help topic assumes the Developer:

- Is familiar with Python
- Has obtained a copy of thesource code and run as noted in the project README.md file.
- Has a REST-enabled Learn instance.

### Code Walkthrough

To build an integration with the Learn REST Web Services, regardless of the programming language of choice, can really be summed up in two steps:

1. Use the Application Key and Secret to obtain an OAuth 2.0 access token, as described in the Basic Authentication document.
2. Call the appropriate REST endpoint with the appropriate data to perform the appropriate action.

### Authorization and Authentication

The REST Services rely on OAuth 2.0 Bearer Tokens for authentication. A request is made to the token endpoint with a Basic Authorization header containing the base64-encoded key:secret string as its key. The token service returns a JSON object containing the Access Token, the Token Type, and the number of seconds until the token expires. The token is set to expire after one hour, and subsequent calls to retrieve the token will return the same token with an updated expiry time until such time that the token has expired. There is no refresh token and currently no revoke token method.

The Python code handles authentication and token management in the auth class in auth.py:

```
        def __init__(self, URL):
            self.SECRET = "insert_your_application_key_here"
            self.KEY = "insert_your_application_secret_here"
            self.CREDENTIALS = 'client_credentials'
            self.PAYLOAD = {
                'grant_type':'client_credentials'
            }
            self.TOKEN = None
            self.target_url = URL
    ...
        def setToken(self):
            oauth_path = '/learn/api/public/v1/oauth2/token'
            OAUTH_URL = 'https://' + self.target_url + oauth_path
            if self.TOKEN is None:
                session = requests.session()
                session.mount('https://', Tls1Adapter()) # remove for production
            # Authenticate
                r = session.post(OAUTH_URL, data=self.PAYLOAD, auth=(self.KEY, self.SECRET),
↪   verify=False)
                print("[auth:setToken()] STATUS CODE: " + str(r.status_code) )
                print("[auth:setToken()] RESPONSE: " + r.text)
                if r.status_code == 200:
                    parsed_json = json.loads(r.text)
                    self.TOKEN = parsed_json['access_token']
    ...
```

The JSON response is parsed to retrieve the Token which is stored and retrieved for subsequent calls by calling `auth.getToken()`. Note that the retrieving the stored token invokes a sanity check on token expiration, requesting a fresh token if nearly expired - see `auth.getToken()` and `auth.isExpired()`.

### Calling Services

The individual service calls are handled by Python in service specific class files:

- datasource.py
- term.py
- course.py
- user.py
- membership.py

A sixth file `constants.py` contains a set of constants used by the application.

Each of these classes creates the JSON body when appropriate and then calls the appropriate HTTP Request, and return the JSON response as a String to be displayed in the terminal console window.

In the demo restdemo.py, each of the above objects are called using POST to create, PATCH to update, GET to read, and DELETE to delete the target object. Note that memberships use PUT to create a membership.

End points are generally defined as `/learn/api/public/v1/<objecttype>/<objectId>`. Object ID can be either the pk1, like `_1_1`, or as the batch_uid. This value should be prepended by externalId:, like `externalId:test101`.

For example, to retrieve a course by the pk1 `_1_1`, you would call **GET /learn/api/public/v1/courses/_1_1**. To retrieve by the batch_uid `test101`, you would call **GET /learn/api/public/v1/courses/externalId:test101.**

Create is sent to Learn as a HTTP POST message with a JSON body that defines the object. The endpoint should omit the objectId, as this will be generated on creation.

Read is sent to Learn as a HTTP GET message with an empty body. The endpoint should include the objectId being retrieved.

Update is sent to Learn as a HTTP PATCH message with a JSON body that defines the object. The endpoint should include the objectId being updated.

Delete is sent to Learn as a HTTP DELETE message with empty body. The endpoint should include the objectId being deleted.

The following shows the key components of the Python code for making REST requests against each of the supported objects. In each case detailed messages are printed to the terminal screen for you to follow the process.

### Datasources

Datasources are handled in `datasource.py`.

#### Variables and Create Payload

```
    self.DATASOURCES_PATH = '/learn/api/public/v1/dataSources' #create(POST)/get(GET)
    self.DATASOURCE_PATH = '/learn/api/public/v1/dataSources/externalId:'
    authStr = 'Bearer ' + token
   ...
    self.PAYLOAD = {
        "externalId":DSKEXTERNALID,
        "description":"Data Source used for REST demo"
    }
```

#### Create

```
    r = session.post("https://" + self.target_url + self.DATASOURCES_PATH,
↪   data=json.dumps(self.PAYLOAD), headers={'Authorization':authStr,
↪   'Content-Type':'application/json'}, verify=False)
```

#### Read

```
    r = session.get("https://" + self.target_url + self.DATASOURCE_PATH+DSKEXTERNALID,
↪   headers={'Authorization':authStr, 'Content-Type':'application/json'}, verify=False)
    ...
    print("[DataSource:getDataSource()] RESPONSE: " + r.text)
```

#### Update

```
    r = session.patch("https://" + self.target_url + self.DATASOURCE_PATH+DSKEXTERNALID,
↪   data=json.dumps(self.PAYLOAD), headers={'Authorization':authStr,
↪   'Content-Type':'application/json'}, verify=False)
    ...
    print("[DataSource:updateDataSource()] RESPONSE: " + r.text)
```

#### Delete

```
    r = session.delete("https://" + self.target_url + self.DATASOURCE_PATH+DSKEXTERNALID,
↪   headers={'Authorization':authStr, 'Content-Type':'application/json'}, verify=False)
```

### Terms

Terms are handled in `term.py`. This Class contains the methods for creating, reading, updating and deleting Term objects.

### Variables and Create Payload

```
    self.terms_Path = '/learn/api/public/v1/terms' #create(POST)/get(GET)
    self.term_Path = '/learn/api/public/v1/terms/externalId:'
    self.termExternalId = TERMEXTERNALID #'BBDN-PYTHON-REST-DEMO-TERM'
    authStr = 'Bearer ' + token
    self.PAYLOAD = {
         "externalId":self.termExternalId,
         "dataSourceId": dsk, #self.dskExternalId, Supported soon.
         "name":"REST Demo Term",
         "description": "Term used for REST demo",
         "availability": {
              "duration":"Continuous"
          }
    }
```

### Create

```
    r = session.post("https://" + self.target_url + self.terms_Path,
↪   data=json.dumps(self.PAYLOAD), headers={'Authorization':authStr,
↪   'Content-Type':'application/json'}, verify=False)
```

### Read

```
    r = session.get("https://" + self.target_url + self.term_Path+self.termExternalId,
↪   headers={'Authorization':authStr},  verify=False)
    ...
    print("[Term:updateTerm()] RESPONSE: " + r.text)
    Update
    self.PAYLOAD = {
           "externalId":self.termExternalId,
           "dataSourceId": dsk, #self.dskExternalId, #Supported soon
           "name":"REST Python Demo Term",
           "description": "Term used for REST Python demo",
           "availability": {
                  "duration":"continuous"
           }
    }
    ...
    r = session.patch("https://" + self.target_url + self.term_Path+self.termExternalId,
↪   data=json.dumps(self.PAYLOAD), headers={'Authorization':authStr,
↪   'Content-Type':'application/json'}, verify=False)
```

### Delete

```
    r = session.delete("https://" + self.target_url + self.term_Path+self.termExternalId,
↪   headers={'Authorization':authStr}, verify=False)
```

### Courses

Courses are handled in `course.py`. This class contains the methods for creating, reading, updating and deleting course objects .

### Variables and Create Payload

```
    self.courses_Path = '/learn/api/public/v1/courses' #create(POST)/get(GET)
    self.course_Path = '/learn/api/public/v1/courses/externalId:'
    self.termId = None
    self.PAYLOAD = {
           "externalId":constants.COURSEEXTERNALID,
           "dataSourceId": dsk, #self.dskExternalId, Supported soon.
           "courseId":constants.COURSEEXTERNALID,
           "name":"Course used for REST demo",
           "description":"Course used for REST demo",
           "allowGuests":"true",
           "readOnly": "false",
           "availability": {
                  "duration":"continuous"
           }
    }
```

**Create**

```
    r = session.post("https://" + self.target_url + self.courses_Path,
↪   data=json.dumps(self.PAYLOAD), headers={'Authorization':authStr,
↪   'Content-Type':'application/json'}, verify=False)
```

**Read**

```
    r = session.get("https://" + self.target_url + self.course_Path+constants.COURSEEXTERNALID,
↪   headers={'Authorization':authStr},  verify=False)
    ...
    print("[Course:getCourse()] RESPONSE: " + r.text)
```

**Update**

```
    self.PAYLOAD = {
           "externalId":constants.COURSEEXTERNALID,
           "dataSourceId": dsk, #self.dskExternalId, Supported soon.
           "courseId":constants.COURSEEXTERNALID,
           "name":"Course used for REST Python demo",
           "description":"Course used for REST Python demo",
           "allowGuests":"true",
           "readOnly": "false",
           "availability": {
                  "available":"Yes",
                  "duration":"continuous"
           }
    }
    r = session.patch("https://" + self.target_url +
↪   self.course_Path+constants.COURSEEXTERNALID, data=json.dumps(self.PAYLOAD),
↪   headers={'Authorization':authStr, 'Content-Type':'application/json'}, verify=False)
```

**Delete**

```
    r = session.delete("https://" + self.target_url +
↪   self.course_Path+constants.COURSEEXTERNALID, headers={'Authorization':authStr},
↪   verify=False)
```

### Users

Users are handled in `user.py`. This class contains the methods for creating, reading, updating and deleting user objects .

**Variables and Create Payload**

```
    self.users_Path = '/learn/api/public/v1/users' #create(POST)/get(GET)
    self.user_Path = '/learn/api/public/v1/users/externalId:'
    self.PAYLOAD = {
           "externalId":USEREXTERNALID,
           "dataSourceId": dsk, #self.dskExternalId, Supported soon.
           "userName":"python_demo",
           "password":"python61",
           "availability": {
                  "available": "Yes"
           },
           "name": {
           "given": "Python",
           "family": "Demo",
           },
           "contact": {
           "email": "no.one@ereh.won",
            }
    }
```

**Create**

```
    r = session.post("https://" + self.target_url + self.users_Path,
↪   data=json.dumps(self.PAYLOAD), headers={'Authorization':authStr,
↪   'Content-Type':'application/json'}, verify=False)
```

**Read**

```
    r = session.get("https://" + self.target_url + self.user_Path+USEREXTERNALID,
↪   headers={'Authorization':authStr},  verify=False)
    ...
    print("[User:getUser()] RESPONSE: " + r.text)
```

**Update**

```
    self.PAYLOAD = {
                "externalId": USEREXTERNALID,
                "dataSourceId": dsk, #self.dskExternalId, Supported soon.
                "userName":"python_demo",
                "password": "python16",
                "availability": {
                    "available": "Yes"
                },
                "name": {
                    "given": "Python",
                    "family": "BbDN",
                    "middle": "Demo",
                },
                "contact": {
                    "email": "no.one@ereh.won",
                }
            }
    ...
    r = session.patch("https://" + self.target_url + self.user_Path+USEREXTERNALID,
↪   data=json.dumps(self.PAYLOAD), headers={'Authorization':authStr,
↪   'Content-Type':'application/json'}, verify=False)
```

**Delete**

```
    r = session.delete("https://" + self.target_url + self.user_Path+USEREXTERNALID,
↪   headers={'Authorization':authStr}, verify=False)
```

### Memberships

Memberships (enrollment/staff) are handled in `membership.py`. This class contains the methods for creating, reading, updating and deleting membership objects .

### Variables and Create Payload

```
    self.PAYLOAD = {
           "dataSourceId":dsk, #self.dskExternalId, supported soon.
           "availability": {
                  "available":"Yes"
           },
           "courseRoleId":"Instructor"
    }
```

### Create

```
    r = session.put("https://" + self.target_url + membership_Path,
↪   data=json.dumps(self.PAYLOAD), headers={'Authorization':authStr,
↪   'Content-Type':'application/json'}, verify=False)
```

### Read

```
    r = session.get("https://" + self.target_url + memberships_Path,
↪   headers={'Authorization':authStr}, verify=False)
    ...
    print("[Membership:getMemberships()] RESPONSE: " + r.text)
```

### Update

```
    self.PAYLOAD = {
           "dataSourceId":dsk,
           "availability": {
                  "available":"No"
           },
           "courseRoleId":"Student"
    }
    ...
    r = session.patch("https://" + self.target_url + membership_Path,
↪   data=json.dumps(self.PAYLOAD), headers={'Authorization':authStr,
↪   'Content-Type':'application/json'}, verify=False)
```

### Delete

```
    r = session.delete("https://" + self.target_url + membership_Path,
↪   headers={'Authorization':authStr}, verify=False)
```

### Conclusion

All of the code snippets included in this document are included in a sample REST Demo Python command line application available on GitHub. There is a README.html included that talks more specifically running the code. Feel free to review the code and run it against a test or development Learn instance to see how it works.