Pacific Gas and Electric Company

Functional & Technical Application Design

Program

Project

Line of Business or Department

Client SDK Python Development Guide

Prepared by

Bharati Vanganuru

Date

05/22/2015

Version

V2.1

Version Type

Draft

Release Q4 2013
Document Instructions

- DO NOT leave Sections blank. All Sections MUST be completed. If a Section is Not Applicable, please enter N/A in the section with a brief explanation.
- Remove/erase all instructional or sample (italic) text
- DO NOT refer to other documents. If certain content is captured in other documents, please “Copy and Paste” OR embed a link to that document
- It is essential you contact ALL relevant stakeholders for your project to complete the content of this document (see project engagement below)
- For additional information on Project Engagement, IT Methodology and Compliance, templates, job aids, departmental links, and training please visit the IT Methodology SharePoint by typing “ITM” in your web browser

About this document

This document provides a complete Design of GasCAP Mobile Application.

Document Control

Change History

<table>
<thead>
<tr>
<th>Author/Contributor</th>
<th>Version</th>
<th>Date</th>
<th>Description of Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bharati Vanganuru</td>
<td>1.0</td>
<td>05/22/2015</td>
<td>Initial Draft</td>
</tr>
<tr>
<td>Bharati Vanganuru</td>
<td>2.0</td>
<td>2/6/2015</td>
<td>Code snippets are added</td>
</tr>
<tr>
<td>Bharati Vanganuru</td>
<td>2.1</td>
<td>4/6/2015</td>
<td>Table Content heading are modified as per the client sdk</td>
</tr>
</tbody>
</table>

Document Ownership and Responsibility

- These are suggested roles for review and approval
- Projects should reference the Deliverable Methodology Responsibility Matrix)

Document Owner
<table>
<thead>
<tr>
<th>Project Role &amp; Responsibility</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT Project Manager</td>
<td></td>
</tr>
</tbody>
</table>

**Document Approvers**

<table>
<thead>
<tr>
<th>Project Role &amp; Responsibility</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT Project Manager</td>
<td></td>
</tr>
<tr>
<td>Business Owner/Sponsor</td>
<td></td>
</tr>
<tr>
<td>Business Technology Leadership</td>
<td></td>
</tr>
<tr>
<td>Business Project Manager</td>
<td></td>
</tr>
</tbody>
</table>
**Document Contributors**

Project Manager to complete table of reviews and approvals shown below. Each technical department lead is to assign the appropriate individual to review and approve the design document. Document approvals shall be conducted in EDRS. This document has a two step review/approval process. 1) Reviews shall be conducted once the Specifications section is completed to confirm technical scope, requirements and technical specifications. 2) Final approval of this document will be made once the design elements have been completed and added to this master design document (either as embedded docs or via links to individual documents stored in the project’s share point).

<table>
<thead>
<tr>
<th>Project Role &amp; Responsibility</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT Project Manager</td>
<td></td>
</tr>
<tr>
<td>Business Analyst</td>
<td></td>
</tr>
<tr>
<td>Business Planner</td>
<td></td>
</tr>
<tr>
<td>Solution Architect</td>
<td></td>
</tr>
<tr>
<td>Infrastructure Architect</td>
<td></td>
</tr>
<tr>
<td>Technology Risk Advisor</td>
<td></td>
</tr>
<tr>
<td>Organizational Change Lead</td>
<td></td>
</tr>
<tr>
<td>Test Lead</td>
<td></td>
</tr>
<tr>
<td>My Fleet Design Lead</td>
<td></td>
</tr>
<tr>
<td>Documentum Key Contacts</td>
<td></td>
</tr>
<tr>
<td>DOT / RC Design Lead</td>
<td></td>
</tr>
<tr>
<td>SAP HCM Key Contacts</td>
<td></td>
</tr>
<tr>
<td>Integration (EI / I&amp;I) Design Lead</td>
<td></td>
</tr>
<tr>
<td>Performance Engineering</td>
<td></td>
</tr>
<tr>
<td>Development</td>
<td></td>
</tr>
<tr>
<td>Disaster Recovery</td>
<td></td>
</tr>
<tr>
<td>Infrastructure Operations</td>
<td></td>
</tr>
</tbody>
</table>
## Document Reviewers

<table>
<thead>
<tr>
<th>Project Role &amp; Responsibility</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Analyst</td>
<td></td>
</tr>
<tr>
<td>CTO Solution Architect</td>
<td></td>
</tr>
<tr>
<td>Infrastructure Architect</td>
<td></td>
</tr>
<tr>
<td>Technology Risk Advisor</td>
<td></td>
</tr>
<tr>
<td>CTO Portfolio Architect</td>
<td></td>
</tr>
<tr>
<td>Training Lead</td>
<td></td>
</tr>
</tbody>
</table>

## Required Reviews and Approvals

<table>
<thead>
<tr>
<th>Project Leads</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution Architect</td>
<td></td>
</tr>
<tr>
<td>Project Manager</td>
<td></td>
</tr>
<tr>
<td>Business Client Lead</td>
<td></td>
</tr>
<tr>
<td>Testing Lead</td>
<td></td>
</tr>
<tr>
<td>Technology Risk Advisor</td>
<td></td>
</tr>
<tr>
<td>Infrastructure Architect</td>
<td></td>
</tr>
<tr>
<td>CTO Portfolio Architect</td>
<td></td>
</tr>
<tr>
<td>Training Lead</td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

1.0 What is Python Scripting.................................................................................................................. 7
2.0 How to Install Python .......................................................................................................................... 7
3.0 Create OAuth2 SDK............................................................................................................................... 8
   3.1 Create setup.py file: .......................................................................................................................... 8
   3.2 Add Python modules: ........................................................................................................................ 9
   3.3 Custom Python modules (created for OAuth2): ................................................................................. 10
      3.3.1 OAuth2.py: ............................................................................................................................... 11
      3.3.2 ClientCredentials.py: ............................................................................................................. 12
      3.3.3 Api.py: .................................................................................................................................... 13
   3.4 Implementation Flow ......................................................................................................................... 15
      3.4.1 Redirect to Login (Data Custodian): ...................................................................................... 15
      3.4.2 Authorization code: .................................................................................................................. 16
   3.5 OAuth Access token Request: .......................................................................................................... 17
      3.5.1 get_access_token(): ................................................................................................................. 17
      3.5.2 get_refreshToken(): ............................................................................................................... 18
   3.6 OAuth Client Access Token Request: ............................................................................................... 19
      3.6.1 get_client_access_token(): ................................................................................................. 19
   3.7 API request using OAuth access token: ......................................................................................... 20
      3.7.1 sync_request(): ....................................................................................................................... 20
      3.7.2 async_request(): .................................................................................................................... 21
1.0 What is Python Scripting

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

2.0 How to Install Python

- Python is released under the MIS license, and bundles other liberally licensed OSS components.
- In order to install Python visit https://www.python.org and download and install for your machine.
- Current version: v0.3.4.3
- Set the environmental path for ..\python34\ and ..\python34\Scripts

Example: python -version: This will give you the version of Python installed in your system.
3.0 Create OAuth2 SDK

Create a project directory where you will start adding Python modules, it looks like below format.

```
pagesdk
  OAuth2
    ___init__.py
    __pycache__
    api.py
    ClientCredentials.py
    oauth2.py
    setup.py
```

3.1 Create setup.py file:

Inside the project directory create a new setup.py file. This will include the details about the project and also the dependencies which we will add to our project

```
setup(name="OAuth2",
      version= "1.0",
      description="library for OAuth2",
      author="Bharati",
      author_email="bharathi@sonata.com",
      packages = find_packages(),
      license = "MIT License"
    )
```

- Name: This provides the name of the application.
- Version: Version of your application.
- Description: Provide brief description about the application.
- Author: This provides the name of the author.
- Packages: Packages will be defined.
- Author_email: Provides email of the author
- License: Provides MIT license
3.2 Add Python modules:

We will be using some of the python predefined modules requests, json and base64. Once we add these modules will be updated with the names and version of their modules folders.

There are two ways to install python modules.

- `pip3.4 install requests`: This will install the requests module in the Lib/site_packages directory.
- `easy_install requests`: This will install the requests module in the Lib/site_packages directory

`pip` and `easy_install` will be available in the python/scripts

| pip3.4 install requests |

Below is the list of the modules being used:

**Requests**: Requests is an Apache2 Licensed HTTP library, written in Python. Most existing Python modules for sending HTTP requests are extremely verbose and cumbersome. We will use this module to make all the requests from our server to PG&E services and provide the response back to the client.

**Command to install Express**: `pip3.4 install requests`

For more details visit: https://docs.djangoproject.com/en/1.8/howto/windows

**Json**: This module is inbuilt in python. The json library can parse JSON from strings or files. The library parses JSON into a Python dictionary or list. It can also convert Python dictionaries or lists into JSON strings.

**Command to install Express**: `pip3.4 install json`

For more details visit: http://docs.python-guide.org/en/latest/scenarios/json

**Base64**: This module is inbuilt in python. It provides data encoding and decoding as specified in RFC 3548. This standard defines the Base16, Base32, and Base64 algorithms for encoding and decoding arbitrary binary strings into text.

For more details visit: https://docs.python.org/2/library/base64.html
3.3 Custom Python modules (created for OAuth2):
These are the modules created by us as per the requirements and to make the code structured by separating the different flows such as OAuth2 flow, Client Credentials flow and API request flow.

- How to install:
  
  Unzip OAuth2 (which is developed for PG&E).
  
  Python ./OAuth2/setup.py install
  
  This will generate OAuth2 and OAuth2.egg-info folders, these should be copy to ../Python34/Lib/site-packages or project directory.

The folder will be look like below:
3.3.1 OAuth2.py:

This file returns OAuth access token and refreshed OAuth access token. While creating a object for this class should be following steps:

- **How to include module**: from OAuth2 import OAuth2
- **How to create a OAuth2 Object**:
  
  ```
  client_credentials_hash =
  {
  "client_key": Client_key,
  "client_secret_key": client_secret_key
  }
  cert_params_hash =
  {
  "crt": ".\apitst_client.crt.pem",
  "key": ".\apitst_client.key.pem"
  }
  oauth = OAuth2(client_credentials_hash, cert_files_hash)
  ```

- **Methods**:
  1. **oauth.get_access_token()**: This is the POST request call. It returns access token with other elements.

```python
def get_access_token(self, url, code, redirect_uri):
    request_params = {
        "grant_type": "authorization_code",
        "code": code,
        "redirect_uri": redirect_uri
    }
    header_params = {
        "Authorization": self.base64code
    }
    request = requests.post(url, data = request_params,
                             headers = header_params, cert = self.cert)
    if request.status_code == "200":
        res = response.json()
        res.update({"status": response.status_code})
        return res
    response = {"status": request.status_code, "error": request.text}
    return response
```

**How to call**: `oauth.get_access_token(url, code, redirect_uri). refer 3.4.1`

  2. **oauth.get_refresh_token()**: This is the POST request call. It returns refreshed access token with other elements.
3.3.2 ClientCredentials.py:

This file returns client access token. While creating a object for this class should be following steps:

- **How to include module:**
  ```python
  from ClientCredentials import ClientCredentials
  ```

- **How to create a Client Credentials Object:**
  ```python
  client_credentials_hash =
  {
  "client_key": Client_key,
  "client_secret_key": client_secret_key
  }

  cert_params_hash =
  {
  "crt": "../apitst_client.crt.pem",
  "key": "../apitst_client.key.pem"
  }

  Client_credentials = ClientCredentials(client_credentials_hash, cert_files_hash)
  ```

How to call: `oauth.get_refresh_token(url)`, for more details refer [3.4.2](#)
Methods:

**get_client_access_token()**: This is the POST request call. It returns access token with other elements.

![Figure 3: Get Client Access Token](image)

def get_client_access_token(self, url):
    request_params = {'grant_type': 'client_credentials'}
    header_params = {'Authorization': self.base64code}
    response = requests.post(url, data=request_params,
                               headers=header_params, cert=self.cert)
    if str(response.status_code) == '200':
        res = response.json()
        res.update({'status': response.status_code})
        return res
    return {'status': response.status_code, 'error': response.text}

How to call: **Client_credentials.get_client_access_token(url)**, refer 3.5.1

---

### 3.3.3 Api.py:

This file contains Sync and Async request API request calls. While creating an object for this class should be following steps:

- **How to include module**: from api import Api
- **How to create a Api Object**:

```python
    cert_params_hash =

    {
        "crt": "../apitst_client.crt.pem",
        "key": "../apitst_client.key.pem"
    }

    api = Api (cert_files_hash)
```

- **Methods**:

  1. **sync_request()**: This is the GET request call. It returns XML data.
Figure 4: Sync Request

```python
def sync_request(self, url, subscription_id, usage_point, published_min, published_max, access_token):
    url = url + '/Subscription/' + subscription_id + '/UsagePoint/' + usage_point
    url += '?published-max=' + published_max + '&published-min=' + published_min
    header_params = {'Authorization': 'Bearer ' + access_token}
    request = requests.get(url, data = {}, headers = header_params, cert = self.cert)
    if request.status_code == 200:
        response = {'status': request.status_code, 'data': request.text}
        return response
    response = {'status': request.status_code, 'error': request.text}
    return response
```

How to call: `api.sync_request(url, subscription_id, usage_point, published_min, published_max, access_token)`, refer 3.6.1

2. `async_request()`: This is the GET request call. It returns XML data, refer 3.6.2

Figure 5: Async Request

```python
def async_request(self, url, subscription_id, published_min, published_max, access_token):
    url = url + '/Subscription/' + subscription_id
    url += '?published-max=' + published_max + '&published-min=' + published_min
    header_params = {'Authorization': 'Bearer ' + access_token}
    request = requests.get(url, data = {}, headers = header_params, cert = self.cert)
    if request.status_code == 200:
        response = {'status': request.status_code, 'data': request.text}
        return response
    response = {'status': request.status_code, 'error': request.text}
    return response
```

How to call: `api.async_request(url, subscription_id, published_min, published_max, access_token)`, refer 3.7.1
3.4 Implementation Flow

3.4.1 Redirect to Login (Data Custodian):

This section provides info on how to implement a redirect to login Domain. Where the user can log in and authorize itself.

- Code Snippet:

```python
def Goto_login(request):
    html = 'Hello PGE! Go to <a rel="https://sharemydataqa.pge.com/myAuthorization/?clientId=2858&verified=true">Authorization</a>

    return HttpResponse(html)
```

**Figure 6: Redirect to PG&E Login Page**

The above code snippet will call during the login process. This will make a request to the login URL by passing certain URL params i.e., clientId

- How to call this method:

  It is a simple `<a href>` call this can be call by the browser

  ```html
  www.localhost:3000/Goto_login
  ```

- Request Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clientId</td>
<td>Integer</td>
<td>Client Key</td>
</tr>
<tr>
<td>Verified</td>
<td>Boolean</td>
<td>True is the default value</td>
</tr>
</tbody>
</table>

- Response for the call:

  The request will initiate a 302 and the application is redirected to the callback which is get_auth_code(). Refer below section for more details.
3.4.2 Authorization code:

This section provides info on how to implement the logic to get the authorization code, which will be used to make the request for OAuth access token.

- Code Snippet:

  **Figure 7: Get Authorization Code**

  ```python
def get_auth_code(request):
    return html
  ```

  The above code snippet code returns the URL to which a redirection should be made in order to get authorization code. This method uses the below listed params:

- Request Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>String</td>
<td>Given URL</td>
</tr>
<tr>
<td>clientId</td>
<td>Integer</td>
<td>Client Key</td>
</tr>
<tr>
<td>scope</td>
<td>String</td>
<td>OAuthAuthorization &quot;code&quot;</td>
</tr>
<tr>
<td>redirect_uri</td>
<td>String</td>
<td>Client side redirect page URL</td>
</tr>
<tr>
<td>response_type</td>
<td>String</td>
<td>Code is the constant value</td>
</tr>
<tr>
<td>action</td>
<td>String</td>
<td>Grant is the Constant Value</td>
</tr>
</tbody>
</table>

- How to call this method:

  It is a simple `<a href>` call this can be call by the browser

  `www.localhost:3000/get_auth_code`

- Response Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>String</td>
<td>Authorization code. It used to get access token</td>
</tr>
</tbody>
</table>
3.5 OAuth Access token Request:

3.5.1 get_access_token():
This method fire posts a request to refresh OAuth access token.

- **Request Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>String</td>
<td>Given URL</td>
</tr>
<tr>
<td>code</td>
<td>String</td>
<td>OauthAuthorization “code”</td>
</tr>
<tr>
<td>redirect_uri</td>
<td>String</td>
<td>Client side redirect page URL</td>
</tr>
</tbody>
</table>

- **Sample data:**

  url = https://api.pge.com/datacustodian/oauth/v2/token
  
  code = '42ef216e-4321-487a-b21a-075a74b02694' (authorization code)
  
  redirect_uri = http://localhost:3000/OAuthCallback

- **API for call:**

  get_access_token( url, code, redirect_uri)

- **Response Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>token_type</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>scope</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>refresh_token</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>access_token</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>resourceURI</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>authorizationURI</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>expires_in</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>error</td>
<td>Text</td>
<td></td>
</tr>
</tbody>
</table>

- **Example data for Successful Response:**

  ```
  {'status': 200, 'token_type': 'Bearer', 'scope': '38475', 'refresh_token': 'fe53dc4e-cccc-4955-98f3-d1eab4c7d9ac', 'access_token': '389dfb40-130a-47b9-9552-5f75efcf190d', 'resourceURI': 'https://api.pge.com/GreenButtonConnect/espi/1_1/resource/Batch/Subscription/38475', 'authorizationURI': 'https://api.pge.com/GreenButtonConnect/epi/1_1/resource/Authorization/38475', 'expires_in': 3600}
  ```
• Example data for failure Response:

{"status": 400, 'error': "Invalid request"}

3.5.2 get_refreshToken(): This method fire posts a request to refresh OAuth access token. It will generate new OAuth access token for each call.

• Request Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>String</td>
<td>Provided URL</td>
</tr>
</tbody>
</table>

• Sample data:

url = https://api.pge.com/datacustodian/oauth/v2/token

• API for call:

get_refresh_token(url)

• Response Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>token_type</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>scope</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>refresh_token</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>access_token</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>resourceURI</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>authorizationURI</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>expires_in</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>error</td>
<td>Text</td>
<td></td>
</tr>
</tbody>
</table>

• Example data for Successful Response:

{"status": 200, 'token_type': 'Bearer', 'scope': '38475', 'refresh_token': '42ef216e-4321-487a-b21a-075a74b02694', 'access_token': '8303cfa5-a50f-476b-9518-3100cc5f2c66', 'resourceURI': 'https://api.pge.com/GreenButtonConnect/espi/1_1/resource/Batch/Subscription/38475', 'authorizationURI': 'https://api.pge.com/GreenButtonConnect/espi/1_1/resource/Authorization/38475', 'expires_in': 3600}

• Example data for failure Response:

{"status": 400, 'error': "Invalid request"}
3.6 OAuth Client Access Token Request:

3.6.1 `get_client_access_token()`: This method fires a post request to get the Client access token.

- **Request Parameters**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>String</td>
<td>Provided URL</td>
</tr>
</tbody>
</table>

- **Sample data**:

  ```
  url = https://api.pge.com/datacustodian/test/oauth/v2/token
  ```

- **API for call**:

  ```
  get_client_access_token(url)
  ```

- **Response Parameters**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>token_type</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>client_access_token</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>expires_in</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Scope</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>Text</td>
<td></td>
</tr>
</tbody>
</table>

- **Example data for Successful Response**:

  ```
  {status:200, 'token_type': 'Bearer', 'client_access_token': 'f29d2196-b644-4cdc-88b6-0700feea7265', 'expires_in': 3600, 'scope': '5+6+7'}
  ```

- **Example data for failure Response**:

  ```
  {'status': 400, 'error': "Invalid request"}
  ```
3.7 API request using OAuth access token:

3.7.1 sync_request(): This method fires a get request to get the XML data. This will be used for both synchronous and asynchronous data requests.

- Request Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>String</td>
<td>Provided url</td>
</tr>
<tr>
<td>subscription_id</td>
<td>String</td>
<td>get_access_token() Response data element “Scope”</td>
</tr>
<tr>
<td>UsagePoint</td>
<td>String</td>
<td>Constant data given by PG&amp;E</td>
</tr>
<tr>
<td>access_token</td>
<td>String</td>
<td>get_access_token() Response data element &quot;access_token&quot;</td>
</tr>
<tr>
<td>Published_min</td>
<td>String</td>
<td>Epoch time in seconds. Defines the upper limit of the data duration</td>
</tr>
<tr>
<td>Published_max</td>
<td>String</td>
<td>Epoch time in seconds. Defines the lower limit of the data duration</td>
</tr>
</tbody>
</table>

- Sample data:

```python
text = "https://api.pge.com/GreenButtonConnect/espi/1_1/resource/Batch/"
subscription_id = '38475'
UsagePoint = '8970920701'
Published_min = "1427886354"
published_max = "1428663954"
access_token = '8303cfa5-a50f-476b-9518-3100cc5f2c66'
```

- API for call:

```python
api_sync_request(url, subscription_id, usage_point, published_min,
published_max, access_token)
```

- Response data:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Integer</td>
</tr>
<tr>
<td>data</td>
<td>Text</td>
</tr>
<tr>
<td>error</td>
<td>Text</td>
</tr>
</tbody>
</table>
- **Example data for successful Response:**
  ```json
  {'status': 200, 'data': XMLdata}
  ```

- **Example data for failure Response:**
  ```json
  {'status': 400, 'error': 'Invalid request'}
  ```

### 3.7.2 asyncio_request()

This method fires a get request to get the XML data.

- **Request Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>String</td>
<td>Given url</td>
</tr>
<tr>
<td>subscription_id</td>
<td>String</td>
<td>get_access_token() Response data element &quot;Scope&quot;</td>
</tr>
<tr>
<td>access_token</td>
<td>String</td>
<td>get_access_token() Response data element &quot;access_token&quot;</td>
</tr>
<tr>
<td>Published_min</td>
<td>String</td>
<td>Epoch time in seconds. Defines the upper limit of the data duration</td>
</tr>
<tr>
<td>Published_max</td>
<td>String</td>
<td>Epoch time in seconds. Defines the lower limit of the data duration</td>
</tr>
</tbody>
</table>

**Example data:**
```python
url = "https://api.pge.com/GreenButtonConnect/espi/1_1/resource/Batch/
subscription_id = '38475'
published_min = "1427886354"
published_max = "1428663954"
access_token = '8303cfa5-a50f-476b-9518-3100cc5f2c66'
```

- **Sample data:**

```python
subscription_id = '38475'
published_min = "1427886354"
published_max = "1428663954"
access_token = '8303cfa5-a50f-476b-9518-3100cc5f2c66'
```

- **API for call:**

```python
api_sync_request(url, subscription_id, usage_point, published_min,
published_max, access_token)
```
• **Response data:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Integer</td>
</tr>
<tr>
<td>data</td>
<td>Text</td>
</tr>
<tr>
<td>error</td>
<td>Text</td>
</tr>
</tbody>
</table>

• **Example data for successful Response:**

```
{‘status’: 202, ‘data: XMLdata}
```

• **Example data for failure Response:**

```
{‘status’: 400, ‘error’ : “Invalid request”}
```