



**Pacific Gas and
Electric Company®**

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

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

Document Ownership and Responsibility

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

Document Owner



Project Role & Responsibility	Name
IT Project Manager	

Document Approvers

Project Role & Responsibility	Name
IT Project Manager	
Business Owner/Sponsor	
Business Technology Leadership	
Business Project Manager	



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).

Project Role & Responsibility	Name
IT Project Manager	
Business Analyst	
Business Planner	
Solution Architect	
Infrastructure Architect	
Technology Risk Advisor	
Organizational Change Lead	
Test Lead	
My Fleet Design Lead	
Documentum Key Contacts	
DOT / RC Design Lead	
SAP HCM Key Contacts	
Integration (EI / I&I) Design Lead	
Performance Engineering	
Development	
Disaster Recovery	
Infrastructure Operations	



Document Reviewers

Project Role & Responsibility	Name
Business Analyst	
CTO Solution Architect	
Infrastructure Architect	
Technology Risk Advisor	
CTO Portfolio Architect	
Training Lead	

Required Reviews and Approvals

Project Leads	
Solution Architect	
Project Manager	
Business Client Lead	
Testing Lead	
Technology Risk Advisor	
Infrastructure Architect	
CTO Portfolio Architect	
Training Lead	



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	Authroization 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

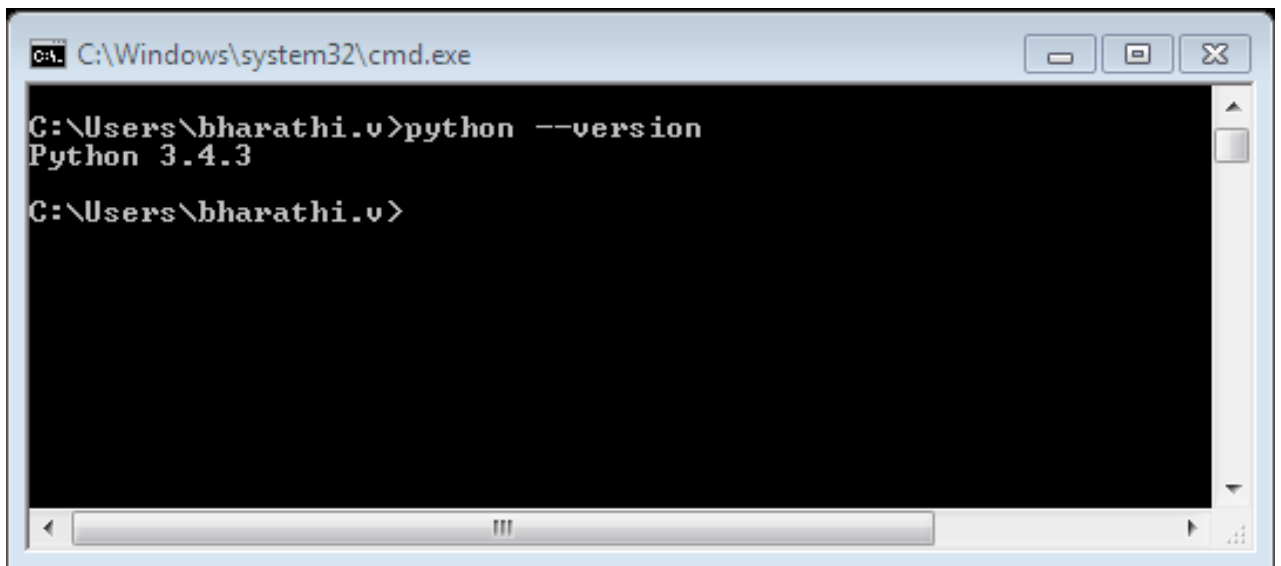
1.0 What is Python Scripting

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.



```

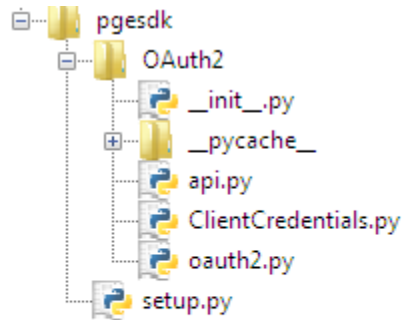
C:\Windows\system32\cmd.exe

C:\Users\bharathi.v>python --version
Python 3.4.3

C:\Users\bharathi.v>
  
```

3.0 Create OAuth2 SDK

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



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*.

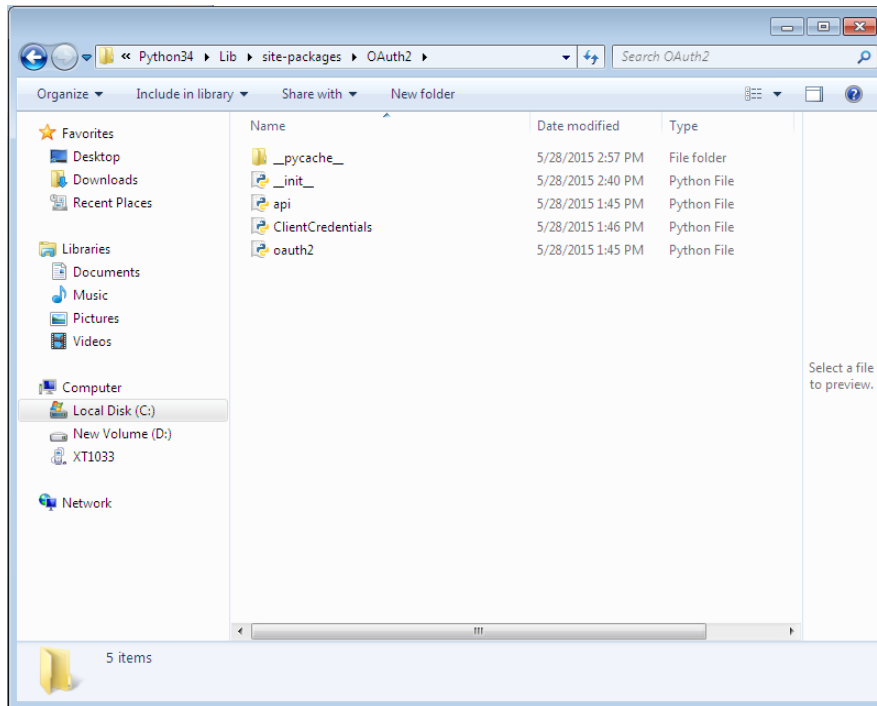
- **How to install:**

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 =  
  
{  
  
    "cert" : "../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.

Figure 1: Get Access Token API

```
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.



Figure 2: Get Refresh token

```
def get_refresh_token(self, url):
    request_params = {"grant_type": "refresh_token",
                     "refresh_token": refresh_token}
    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_refresh_token(url)`, for more details [refer 3.4.2](#)

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 :**
from ClientCredentials import ClientCredentials
- **How to create a Client Credentials Object:**

```
client_credentials_hash =
{
"client_key" : Client_key,
"client_secret_key": client_secret_key
}
```

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

```
Client_credentials = ClientCredentials (client_credentials_hash,
cert_files_hash)
```



- **Methods:**

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

Figure 3: Get Client Access Token

```
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 a object for this class should be following steps:

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

```
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

```
def sync_request(self, url, subscription_id, usage_point, published_min, published_max, access_token):
    url = url + "/Subscription/" + subscription_id + "/UsagePoint/" + usage_point
    url = 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

```
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:**

```
def Goto_login(request):  
    html = 'Hello PGE! Go to <a ref="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

www.localhost:3000/Goto_login

- **Request Parameters:**

Name	Data Type	Description
clientId	Integer	Client Key
Verified	Boolean	True is the default value

- **Reponse 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 Authroization 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

```
def get_auth_code(request):  
    html = '<a href=https://apiqa.pge.com/datacustodian/oauth/v2/authorize?client_id=  
        323835384B6579&redirect_uri=http://localhost:3000/OAuthCallback&scope=9951  
        &response_type=code&action=Grant">Get Auth Code</a>'  
    return html
```

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

- **Request Parameters:**

Name	Data Type	Description
url	String	Given URL
clientId	Integer	Client Key
scope	String	OAuthAuthurization "code"
redirect_uri	String	Client side redirect page URL
response_type	String	Code is the constant value
action	String	Grant is the Constant Value

- **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:**

Name	Data Type	Description
Code	String	Authorization code. It used to get access token



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:**

Name	Data Type	Description
url	String	Given URL
code	String	OAuthAuthorization "code"
Redirect_uri	String	Client side redirect page URL

- **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:**

Name	Data Type
status	Integer
token_type	String
scope	String
refresh_token	String
access_token	String
resourceURI	String
authorizationURI	String
expires_in	Integer
error	Text

- **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:**

Name	Data Type	Description
url	String	Provided URL

- **Sample data:**

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

- **API for call:**

```
get_refresh_token(url)
```

- **Response Parameters:**

Name	Data Type
status	Integer
token_type	String
scope	String
refresh_token	String
access_token	String
resourceURI	String
authorizationURI	String
expires_in	String
error	Text

- **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 fire a post a request to get the Client access token.

- **Request Parameters:**

Name	Data Type	Description
url	String	Provided URL

- **Sample data:**

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

- **API for call :**

`get_client_access_token (url)`

- **Response Parameters:**

Name	Data Type
Status	Integer
token_type	String
client_access_token	String
expires_in	String
Scope	String
Error	Text

- **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:

Name	Data Type	Description
url	String	Provided url
subscription_id	String	get_access_token() Response data element "Scope"
UsagePoint	String	Constant data given by PG&E
access_token	String	get_access_token() Response data element "access_token"
Published_min Example:	String	Epoch time in seconds. Defines the upper limit of the data duration
published_max Example:	String	Epoch time in seconds. Defines the lower limit of the data duration

- Sample data:

url =" 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:

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

- Response data:

Name	Data Type
status	Integer
data	Text
error	Text



- **Example data for successful Response :**

{'status': 200, 'data : XMLdata}

- **Example data for failure Response :**

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

3.7.2 `async_request()`: This method fires a get request to get the XML data.

- **Request Parameters:**

Name	Data Type	Description
url	String	Given url
subscription_id	String	get_access_token() Response data element "Scope"
access_token	String	get_access_token() Response data element "access_token"
Published_min	String	Epoch time in seconds. Defines the upper limit of the data duration
Published_max	String	Epoch time in seconds. Defines the lower limit of the data duration

Example data:

url = "https://api.pge.com/GreenButtonConnect/responses/1/resource/Batch/

- **Sample data:**

subscription_id = '38475

published_min = "1427886354"

published_max = "1428663954"

access_token = '8303cfa5-a50f-476b-9518-3100cc5f2c66'

- **API for call:**

api_sync_request(url, subscription_id, usage_point, published_min,

published_max, access_token)



- **Response data:**

Name	Data Type
status	Integer
data	Text
error	Text

- **Example data for successful Response :**

{'status': 202, 'data: XMLdata}

- **Example data for failure Response :**

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