Pacific Gas and Electric Company

Functional & Technical Application Design

Program

Project

Line of Business or Department

SDK Development Node JS Development Guide

Prepared by	Siddhant Wadhera
Date	06/03/2015
Version	*V2.0*
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
Siddhant Wadhera	1.0	05/15/2015	Initial Draft
Siddhant Wadhera	2.0	06/03/2015	Changed as per Client Feedback

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 sharepoint).

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 Engeneering	
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 Node JS 7
2.0 How to Install Node JS 7
3.0 Node Modules used 8
4.0 Custom node modules created: 10
4.1 Config: 10
4.2 OAuth2.0 11
4.3 authCode: 11
4.4 client: 11
4.5 api 11
5.0 Implementation Guide: 12
5.1 Redirect to Login (Data Custodian): 12
5.2 Authorization code : 13
5.3 OAuth access Token Request: 14
5.4 Refresh OAuth access Token: 17
5.5 OAuth Client Access Token Request: 19
5.6 API request using OAuth access token: 21

1.0 What is Node JS

Node.js is an open source, cross-platform runtime environment for server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows etc.

Node.js provides an event-driven architecture and a non-blocking I/O API that optimizes an application's throughput and scalability. These technologies are commonly used for real-time web applications.

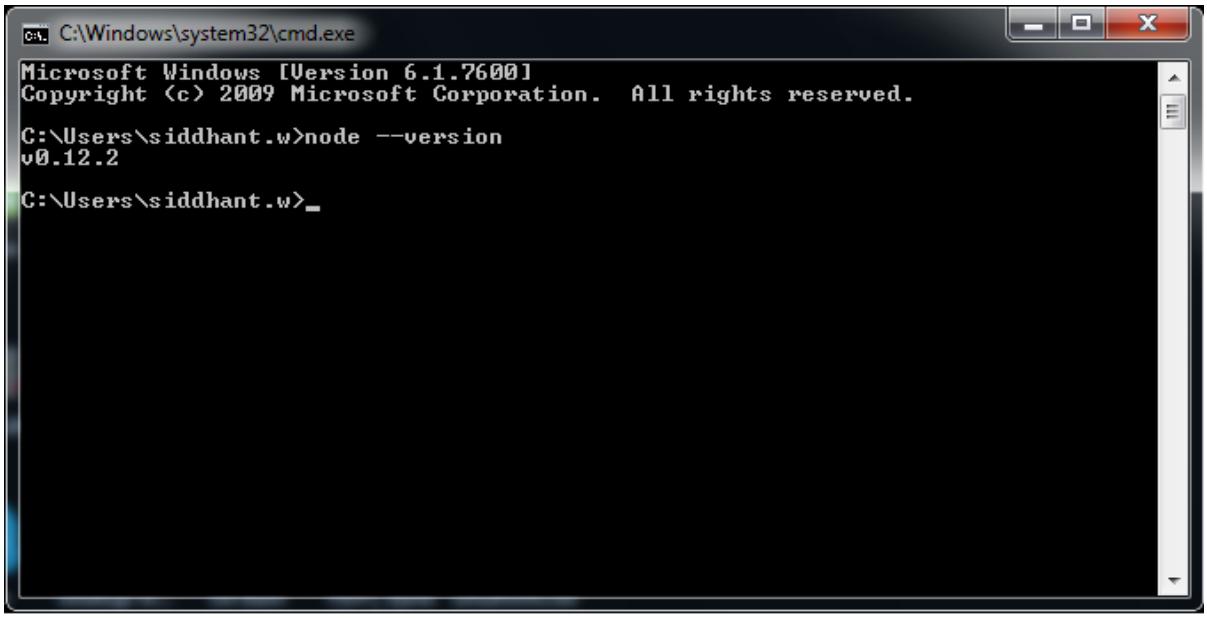
Node.js uses the Google V8 JavaScript engine to execute code, and a large percentage of the basic modules are written in JavaScript. Node.js contains a built-in library to allow applications to act as a Web server without software such as Apache HTTP Server or IIS.

Node.js is gaining adoption as a server-side platform.

2.0 How to Install Node JS

- Node.js is released under the MIT license, and bundles other liberally licensed OSS components.
- In order to install Node JS visit https://nodejs.org/download/ and download the Node.js source code or a pre-built installer for your platform
- **Current version: v0.12.2**
- Run the .exe file downloaded and follow the installation instructions
- Once the installation process completed. Open a Terminal and check whether node is installed properly by running any of node commands

Ex: node –version : This will give you the version of node installed in your system.



3.0 Node Modules used

We will be using some of the node modules given by the npm community such as request, express etc.

Once we add these modules the dependencies object in package.json file will be updated with the names and version of there modules.

There are two ways by which node modules can be installed.

- **node install --save xyz** : This will install the module in the app directory and save it in the dependencies list in package.json.
- **node install xyz :** install module temporarily, and not add it to the dependencies list, omit the `--save` option:

Node modules installed with the --save option are added to the dependencies list in the package.json file. Then using npm install in the app directory will automatically install modules in the dependencies list.

The above commands will install the latest version of the modules. But if you want to install a different/older version of module. Use the below command :

```
npm install module@2.x.x
```

Below is the list of the modules being used:

1. **Express**: It is a light-weight web application framework to help organize your web application into MVC architecture on the server side. This is similar to what Ruby on Rails or Sinatra is to Ruby.

Command to install Express: npm install --save express

For more details visit: http://expressjs.com/

2. **Request:** Simplified HTTP request client. Request module is designed to be the simplest way possible to make http calls. It supports HTTPS and follows redirects by default. We will use this module to make all the request from our server to PG&E services and provide the response back to the client.

- **Command to install Express: npm install --save request**
- Example of how to require this module :

```
request(options,callback);
```

For more details visit: https://www.npmjs.com/package/request

3. **Querystring:** Node's querystring module for all engines. This module provides utilities for dealing with query strings. Such as JSON stringify, JSON parse etc.
   - **Command to install Express: npm install --save querystring**
   - Example of how to require this module :

     qs.stringify(params);

For more details visit: https://nodejs.org/api/querystring.html

4. **Body-Parser:** Node.js body parsing middleware. This will help us in parsing the incoming request body to our server. This does not handle multipart bodies, due to their complex and typically large nature. For multipart bodies, you may be interested in the following modules.

   - **Command to install Express: npm install --save body-parser**
   - Example of how to require this module :

     app.use(bodyParser.json())

For more details visit: https://nodejs.org/api/querystring.html

5. **Fs:** This module will be used as a file system in order to read the cert files and share it with PG&E services.
   - **Command to install Express: npm install --save body-parser**
   - Example of how to require this module :
     fs.readFileSync('apitst_client.p12')

For more details visit: https://nodejs.org/api/fs.html

Please note that the above mentioned modules will also install dependencies modules required by them internally.

4.0 Custom node modules created:

These are the modules created by us as per the requirements and to make the code structured by separating the different flows such as OAuth flow , client Credentials Flow and API request flow .

These modules are as explained below :

4.1 Config: This holds the configuration object which is returned when this file is required.

- Code Snippet in file:

```
var fs = require('fs');

module.exports = {

  clientID: "323835384B6579",

  clientSecret: "32383538536563726574",

  clientKey : "2858",

  login :"https://sharemydata.pge.com/myAuthorization/",

  site: 'https://api.pge.com/datacustodian',

  authorizationPath : '/oauth/v2/authorize',

  dataRequestURL :
"https://api.pge.com/GreenButtonConnect/espi/1_1/resource/Batch/Subscription/",

  tokenPath: '/oauth/v2/token',

  agentOptions : {

    ////read the certificate and also pass the password

    pfx: fs.readFileSync('apitst_client.p12'),

    passphrase: 'temp12',

  }

}
```

- Example of how to require this module :

```
var appConfig = require('./config');
```

4.2 OAuth2.0 : This file requires a config file and returns two other modules defined for oAuth and login flow by passing the config object

a. authCode
b. client.

- Code Snippet in file:

```
var appConfig = require('./config');

module.exports = function() {

        return {

   'authCode': require('./auth-code')(appConfig),

   'client': require('./client')(appConfig)

  }

};
```

The code above returns back two other modules which will be used for Login Redirect, OAuth access token and Client access Token

- Example of how to require this module :

```
var oauth2 = require('./lib/OAuth2.0')();
```

4.3 authCode: This module file contains all the methods to implement the Login redirection, OAuth access Token and Refresh OAuth access Token.

4.4 client: This module file contains all the methods to implement the business logic to get the OAuth Client access Token.

4.5 api: This module returns method written for implementing api calls for both Synchronous and Asynchronous GET requests.

5.0 Implementation Guide:

5.1 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.

The method named *goToLogin* is defined in authCode module file. Below are the details of the method defined to implement this logic :

a. *goToLogin*: This method is called during the login process. This will make a request to the login URL by passing certain URl params. It takes only one parameter i.e. callback method. This method will construct the required body object for POST request.

- Example of how to use this method :

  oauth2.authCode.goToLogin(redirectCallback);

- Code Snippet in file:

```
function goToLogin(callback){
   var options= {
    "url": config.login,
     "headers" :{
      "Content-Type": "application/json"
     },
    "qs" :{
                    "clientId" : config.clientKey,
                    "verified" : "true"
          },
     "method": "POST"
   };
   request(options, callback);
}
```

The attributes are explained below:

Key	Value
url	The Url to which the request should be made to Ex: https://sharemydata.pge.com/myAuthorization/
config. login	Base url login request,
config.clientKey	holds the client key
verified	Holds Boolean value
method	What kind of request we are going to make

- To make the request we use npm request module:

  request(options, callback);

- The request will initiate a 302 which should be handled in the callback method and the application is redirected to the Location.

5.2 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

The method named *authorizeURL* is defined in authCode module file. Below are the details of the method defined to implement this logic :

a. *authorizeURL*: This method constructs and returns the URL to which a redirection should be made in order to get authorization code. This methods uses the config object given by config Module and param object passed while invoking this method

- Example of how to use this method:

```
var authorization_uri = oauth2.authCode.authorizeURL({
   redirect_uri: 'http://localhost:3000/OAuthCallback',
   scope: '9951',
 });
res.redirect(authorization_uri);
```

- Code Snippet in file:

```
function authorizeURL(params) {
  params.response_type = 'code';
  params.client_id = config.clientID;
  params.action = "Grant"
  return config.site + config.authorizationPath + '?' + qs.stringify(params);

}
```

The attributes are explained below:

Key	Value
config.site	Base url for token request,
config.authorizationPath	path to be appended to site key for making the authorization URL.
redirect_uri	The redirect URL where the PG&E service should redirect to Ex : http://localhost:3000/OAuthCallback
scope	defines the scope of the user
response_type	Defines what kind of response is expected. In this case it is ”code
action	Holds the client ID
verified	what type of action is required. In this case it is “grant”.
method	What kind of request we are going to make

i. redirect –

5.3 OAuth access Token Request:

This section provides info on how to implement the logic on how to make a request and get the OAuth access Token.

The method named *getOAuthToken* is defined in authCode module file. Below are the details of the method defined to implement this logic:

a. *getOAuthToken*: This method makes a request and returns OAuth Access Token object. It takes two parameters i.e. params object and callback method.

- Example of how to use this method:

```
var params = {
  code: req.query.code,
  redirect_uri: 'http://localhost:3000/OAuthCallback'
 }

oauth2.authCode.getToken(params, saveToken);
```

The params attribute are defined below:

Key	Value
code	The authorization code received after log-in
redirect_uri	The redirect URL where the PG&E service should redirect to Ex : http://localhost:3000/OAuthCallback

- Code Snippet in file:

```
function  getOAuthToken (params, callback) {
  var options= {
     "url": config.site+config.tokenPath,
       "headers" :{
        "Content-Type": "application/json",
        "Authorization" : base64String
       },
     "qs" :{
       "grant_type" : "authorization_code",
       "code" : params.code,
       "redirect_uri"  : params.redirect_uri,
     },
    "agentOptions": config.agentOptions,
       "method": "POST"
   };
  request(options,callback);
}
```

The attributes are explained below:

Key	Value
url	The Url to which the request should be made to Ex: https://api.pge.com/datacustodian/oauth/v2/authorize
config.site	Base url for token request,
config.tokenPath	holds the client access token path
Authorization	Basic base64 encoded String This is made of “Basic ” + “ClientID:ClientSecret” Ex : “ Basic “ + “MzIzODM1Mzg0QjY1Nzk6MzIzODM1Mzg1MzY1NjM3MjY1NzQ=”
grant_type	Defines what kind of grant type it is
redirect_uri	The redirect URL where the PG&E service should redirect to Ex : http://localhost:3000/OAuthCallback
code	The authorization code received
agentOptions	Holds the SSL cert required for 2-way SSL.
method	What kind of request we are going to make

- To make the request we use npm request module:
  request(options, callback);

- Response Parameters received in callback method based on success/error:

Key	Value
statusCode	Holds the request StatusCode
token_type	Defines what kind of token it is
'refresh_token'	Holds the refresh token value this will be used in order to refresh the OAuth Access Token
'access_token'	Holds the OAuth access token value
resourceURI	The base Uri to which request should be made in order to make Sync and Async requests to get the XML data
authorizationURI	The authorized URI
expires_in	Expiry duration of the token
error	Defines the type of error if any

- Example data for Successful Response:
  {‘ statusCode: 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”}

5.4 Refresh OAuth access Token:

This section provides info on how to implement the logic on how to make a request and refresh the OAuth access Token.

The method named *refreshOAuthToken* is defined in authCode module file. Below are the details of the method defined to implement this logic:

a. *refreshToken:* This method is used to refresh the OAuth access token received in the above step. . It takes two parameters i.e. refresh_token received from Client and callback function.

- Example of how to use this method:

  oauth2.authCode.refreshToken(req.body.refresh_token,refreshTokenCallback);

Key	Value
req.body.refresh_token	The refresh token value

- Code Snippet in file:

```
function refreshOAuthToken(refreshToken,callback){
  var options= {
     "url": config.site+config.tokenPath,
       "headers" :{
        "Content-Type": "application/json",
         "Authorization" : base64String
       },
      "qs" :{
       "grant_type" : "refresh_token",
       "refresh_token" : refreshToken
      },
     "agentOptions": config.agentOptions,
       "method": "POST"
    };
    request(options,callback);
}
```

The request object is similar to OAuth acess Token request object.
The Difference is as defined below in the table

Key	Value
grant_type	It holds a different value i.e “refresh_token”
refresh_token	The refresh token received from Client

Please note that Response Parameters, Success/failure scenarios are similar to OAuth access token

5.5 OAuth Client Access Token Request:

This section provides info on how to implement the logic on how to make a request and get the OAuth Client access Token.

The method named *getClientToken* is defined in client module file. Below are the details of the method defined to implement this logic:

a. *getClientToken: This method returns Client Access Token object. It takes only one parameter i.e. callback function.*

- Example of how to use this method:
  oauth2.client.getClientToken(saveToken);

- Code Snippet in file:

```
function getClientToken(callback) {
        var options = {
            "url"  : config.site +"/test" + config.tokenPath,
            "headers" :{
                "Content-Type": "application/json",
                "Authorization" : base64String
            },
            "qs" :{
                "grant_type" : "client_credentials",
            },
            "agentOptions": config.agentOptions,
            "method": "POST"
        }
        request(options, callback);
}
```

The attributes are explained below:

Key	Value
url	The Url to which the request should be made to Ex: https://api.pge.com/datacustodian/test/oauth/v2/token
config.site	Base url for token request,
config.tokenPath	holds the client access token path
Authorization	Basic base64 encoded String This is made of “Basic ” + “ClientID:ClientSecret” Ex : “ Basic “ + “MzIzODM1Mzg0QjY1Nzk6MzIzODM1Mzg1MzY1NjM3MjY1NzQ=”
grant_type	Defines what kind of grant type it is
agentOptions	Holds the SSL cert required for 2-way SSL.
method	What kind of request we are going to make

- To make the request we use npm request module:

  request(options, callback);

- Response Parameters:

Key	Value
statusCode	Holds the request StatusCode
token_type	Defines what kind of token it is
client_access_token	Holds the token value
expires_in	Expiry duration of the token
Scope	The scope of the connected user
error	Defines the type of error if any

- Example data for Successful Response:

  { statusCode: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”}

5.6 API request using OAuth access token:

This section provides info on how to implement the logic on how to make a api request and get the Data.

The method named *makeRequest* is defined in api module file. Below are the details of the method defined to implement this logic:

a. *makeRequest: This method returns appropriate data/response based on what type of request is made. It checks for what type of request is to be made and constructs the request URL accordingly. It takes two parameters i.e. params object and callback function.*

- Example of how to use this method:
apiRequest.makeRequest(req.body,dataRequestCallback);

The attributes in *req.body* are explained below:

Key	Value
type	the type of request to be made to PG&E servers. This will be used on the node server end to create the exact Url from the config Object. Ex: sync or Async
subscriptionID	the subscription ID which was provided during the OAuth access token process
UsagePointID	the usagePoint ID as provided during registration
token	OAuth access token received earlier this will be used to add the Authorization Bearer header
maxDate	Epoch time in seconds. Defines the upper limit of the data duration
minDate	Epoch time in seconds. Defines the lower limit of the data duration
dataRequestURL	path to be appended to site key for making the dataRequestURL URL
tokenPath	holds the client access token path. This will be appended to site key while making the request for Client token request
agentOptions	This will hold the SSL cert and its passphrase as well. We use the fs module in order to read the .p12 cert file

- Code Snippet in file:

```
function makeRequest(params,callback){
if (params.type==="Async"){
params.url = appConfig.dataRequestURL + params.subscriptionID
}
else{
params.url = appConfig.dataRequestURL + params.subscriptionID + "/UsagePoint/" + params.usagePointID
}
        var options = {
          "url": params.url,
            "headers" :{
             "Content-Type": "application/json",
             "Authorization" : "Bearer " + params.token
            },
            "qs" : {
              "published-max" : params.maxDate,
              "published-min" : params.minDate
            },
             "agentOptions": appConfig.agentOptions,
            "method": "GET"
        }
        request(options,callback);
}
```

As you can see above the method checks for what type of request is to be made and accordingly builds the url where the request should be made to.

Please note that the request method is “GET” here

The attributes are explained below:

Key	Value
url	The Url to which the request should be made to Ex: For Async : https://api.pge.com/GreenButtonConnect/espi/1_1/resource/Batch/Subscription/38499?published-max=1428663954&published-min=1427886354 For Sync : https://api.pge.com/GreenButtonConnect/espi/1_1/resource/Batch/Subscription/38499/UsagePoint/8970920701?published-max=1428663954&published-min=1427886354
Authorization	This is made of “Bearer ” + oAuth Access Token Ex : "Bearer " + “6bed6ff4-cfa9-4eda-9903-b3d2ae7c29d4”
grant_type	Defines what kind of grant type it is
agentOptions	Holds the SSL cert required for 2-way SSL.

- To make the request we use npm request module:
  request(options, callback);

- Response Parameters:

Key	Value
statusCode	Holds the request StatusCode
data	The XML data
error	Defines the type of error if any

- Example data for successful Response :

  {‘status’: 200, ‘data : XMLdata}

- Example data for failure Response :

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