FP

# Object Oriented JS

## Features:

* Define class
* Define instance members
* Define instance functions
* Define static members
* Define static functions
* Define constructor function to initialize the object
* Access static members from instance functions
* Create class instances
* Access instance members using class instance
* Invoke instance functions using class instance
* Access/Invoke static members/functions
* Automatic getter/setter functions for instance properties
* Invoke apply\* function to execute business logic before updating instance properties
* Define inheritance chain
* Invoke base constructors
* Invoke base functions
* Define members/functions in derived classes
* override base functions
* Support for classes defined with namespaced-naming convention
* Access base object functions using "base" keyword
* Mixins
  + use Mixin for borrowing behavior
* Define a singleton class
* Define static class
* Prevent inheritance using “sealed” concept.
* Support function overloading
* Use strict mode
* Node
  + Works in node environment
* Comprehensive test suite using Jasmine
* Grunt support:
  + Linting (uses JSHint)
  + Minification (uses uglify)
  + Running unit tests

Pending:

* Inheritance:
  + Access base object instance properties using “base” keyword
* getter/setter/apply – nested objects
* Provide ability to define “abstract class”.
* ~~Rename the constructor function to be the class name instead of framework-defined one. This cannot be done.~~ Hence exposing a property “\_className” which returns the class name which can be used during debugging.
* Private members
* Protected members
* Preprocessor to improve performance
* Duplicate class definition checks. What should be the behavior? Should we throw exception or override the class definition?
* Node
  + Create NPM module to use in Node.js applications (npm install fp)

# Other utility functions

* Adding remove to array –
* includes function to check if element is present in array. Better since it returns true/false then index/-1

# Miscellaneous

* Make it available using bower (bower install fp.js)
* Add to GitHub
* Add documentation using JSDuck. This should generate a html file
* JSPerf to check performance
* Add custom rules for JSHint.
* ~~Add unit tests~~
* Use Grunt:
  + generate jsduck documentation

References:

<https://javascriptweblog.wordpress.com/2011/05/31/a-fresh-look-at-javascript-mixins/>

# BDT Flow

# Appendix A – Dev Environment and Source Control

This section describes the development environment and source control configuration used for FP development.

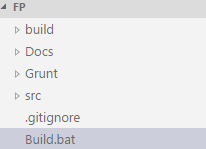
* IDE – Visual Studio Code
* Source Control System – git and GitHub
* Git Easy VS Code extension

Steps:

1. Install Visual Studio Code - <https://code.visualstudio.com>
2. Install git - <https://git-scm.com/downloads>
3. Open Visual Studio Code and point to the root folder of the dev directory
4. Using the Git Easy extension , integrate with git. Make commits after code changes.

## Project Structure

This section describes the project structure.



|  |  |  |
| --- | --- | --- |
| Folder | Files | Description |
| Build/ | All binaries are copied to the build folder.   * Minified version of FP-0.1.0.js * FPModule-0.1.0.js for Node |  |
| Docs/ | Contains documentation | TBD. JSDuck-based documentation |
| Grunt/ | Gruntfile.js  Package.json  Installed NPM modules | Grunt is used for build automation |
| Src/Code | FP-0.1.0.js  FPModule-0.1.0 | Code folder is where the core framework source files are available. |
| Src/Tests | Unit tests using Jasmine  Specs:   * FPSpec.Core.js – shared test code * FP-spec.js - specs for core framework * FPUtil-spec.js – specs for FP.Util.js   /lib  Jasmine files  SpecRunner.html |  |
| Harness/ | Sample code files   * App.html – * Client.js – Sample examples | To use:  Open app.html in browser to see the sample code in action |
| Build.bat  Deploy To Github.bat | Batch file to build the library and create production binaries  Batch file to push local commits to FP GitHub repository | To use:   * CD to root directory * Type build -force   To use:  CD to root directory  Type Deploy to Github |

## Github Integration

* Repository - <https://github.com/fshaikh/FP>
* Remote Repository URL - <https://github.com/fshaikh/FP.git>

### Steps to push existing repository

1. Open Git Cmd
2. Change the current working directory to local project
3. In command prompt, add the URL for remote repository where the local repository will be pushed.

git remote add origin <https://github.com/fshaikh/FP.git>

1. Verify if the remote has been set correctly

git remove –v

1. Push changes in your local repository

git push origin master

OR

1. Run Deploy to GitHub.bat

NOTE: Ensure the above is run after local commits to git repository.

### Steps to clone GitHub FP Repository

1. Open Git Cmd/Git Bash
2. Change the current working directory to the location where you want the cloned directory to be made.
3. Type

Git clone <https://github.com/fshaikh/FP.git>

1. Press ENTER. Your local clone will be created

## References:

<https://help.github.com/articles/adding-an-existing-project-to-github-using-the-command-line/>

# Appendix B - Grunt

## Grunt

* Pre-requisites
  + Install Nodejs
  + Install NPM
* Install
  + Install GRUNT cli
    - Open command prompt in administrator mode
    - type: npm install -g grunt-cli (This will install the grunt-cli module globally)
  + Install GRUNT
    - Open command prompt in administrator mode
    - cd to directory where the project is
    - type: npm install grunt --save-dev
* Configure
  + Create package.json
  + Create gruntfile.js

## Linting Steps

To help improve the quality of the code, we will be using a linter to identify issues with the library code. For this project, JSHint will be used. (<http://jshint.com/>)

1. npm install grunt-contrib-jshint --save-dev
2. add the folllwing line in Gruntfile.js : grunt.loadNpmTasks("grunt-contrib-jshint");
3. add the folllwing line in Gruntfile.js : grunt.registerTask("default",['jshint']);
4. Add the following configuration in Gruntfile.js in initConfig function call:

// JSHint configuration

jshint:{

options:{ },

files:['../src/Code/\*.js']

},

## Minification steps:

1. npm install grunt-contrib-uglify --save-dev
2. add the folllwing line in Gruntfile.js : grunt.loadNpmTasks("grunt-contrib-uglify");
3. add the folllwing line in Gruntfile.js : grunt.registerTask("default",['uglify']);
4. Add the following configuration in Gruntfile.js in initConfig function call:

// uglify configuration

uglify: {

development:{

files:[{

expand:true,

cwd:'../src/Code/', // current working directory

src:'\*\*/\*.js', // what files to minify

dest:'../build/' // destination folder where the minified files will be copied

}]

},

options:{

mangle:true,

compress:{

drop\_console:true // drops all references to console from the minified file

}

}

}

## Replace Steps

1. npm install grunt-replace --save-dev
2. add the following line in Gruntfile.js : grunt.loadNpmTasks('grunt-replace'); // replace
3. add the folllwing line in Gruntfile.js : grunt.registerTask("default",['replace']);
4. Add the following configuration in Gruntfile.js in initConfig function call:

// replace configuration

replace: {

dist: {

options: {

patterns: [

{

match: 'CODE',

replacement: '<%= grunt.file.read("../src/Code/FP-0.1.0.js") %>'

}

]

},

files: [

{

expand: true,

flatten: true,

src: ["../src/Code/FPModule-0.1.0.js"],

dest: "../src/Code/"

}

]

}

},

References:

<https://github.com/gruntjs/grunt-contrib-uglify>

<http://jshint.com/docs/options>

<http://gruntjs.com/getting-started>

<https://github.com/outaTiME/grunt-replace>

# Appendix C – Jasmine

## Setting up

1. Get the latest Jasmine from : <https://github.com/jasmine/jasmine/releases>. Download jasmine-standalone-\*.zip. (\* - version)

NOTE: As of this writing, the version is 2.5.0.

1. Unzip the files to a directory

|  |  |  |
| --- | --- | --- |
| Concept | Description | Comment |
| Describe |  |  |
| It() | * Container for each unit test * Must be nested within a describe function | it("should be true",function(){  expect(true).toBeTruthy();  }); |
| beforeEach and afterEach | Setup and tear down are done in these functions |  |
| Matchers | Assert statements. Expect function |  |
| Custom Matchers |  |  |

## Grunt Task

1. npm install grunt-contrib-jasmine --save-dev
2. add the following line in Gruntfile.js : grunt.loadNpmTasks("grunt-contrib-jasmine");
3. add the following line in Gruntfile.js : grunt.registerTask("default",['jasmine']);
4. Add the following configuration in Gruntfile.js in initConfig function call:

// jasmine configuration

jasmine: {

all: {

src: [

'../src/Code/\*.js', // location of javascript files

],

options: {

'specs': '../src/Tests/FP-spec.js' // location of spec files

}

}

},

## References:

<https://github.com/JamieMason/Jasmine-Matchers>

<https://www.jpetersson.se/blog/post/getting-started-with-javascript-testing-using-grunt-jasmine-and-istanbul/>

# Appendix D: Applications

This lists all the applications to be built to showcase the capability and usefulness of the framework.

* AWS NodeJS SDK
* Simple Promise Framework