Contents

[Overview 2](#_Toc31343)

[How to read this documents? 2](#_Toc28033)

[What’s the code coverage? 2](#_Toc11257)

[Code coverage criteria 2](#_Toc25653)

[Background 2](#_Toc28807)

[FTVP font-end tests 2](#_Toc15534)

[Usage tools or framework 2](#_Toc5202)

[Part 1 Unit Test for VP 3](#_Toc27341)

[Setup unit test environment (Jasmine) 3](#_Toc23067)

[Unit Test for Graphics 4](#_Toc21286)

[Unit Test for Angular 5](#_Toc30098)

[Part 2 Code Coverage Practice 7](#_Toc32338)

[Setup code coverage environment (Karma-Coverage) 7](#_Toc2549)

[Configure karma-coverage 8](#_Toc21399)

[Run it 8](#_Toc22146)

[Conclusion 11](#_Toc32255)

[Reference 11](#_Toc18363)

# Overview

## How to read this documents?

This document will tell you how to write your tests codes and run the code coverage tool to measure your unit test codes quality. If you are familiar with unit test for JavaScript, you can skip Part 1 and read Part 2 directly. Part 1 will tell you how to setup and write unit test for FTVP. Part 2 will tell you how to setup and configure the code coverage Karma-Coverage.

## What’s the code coverage?

In computer science, code coverage is a measure used to describe the degree to which the source code of a program is tested by a particular test suite.

## Code coverage criteria

There are a number of coverage criteria, the main ones beings:

Function coverage - Has each function (or subroutine) in the program been called?

Statement coverage - Has each statement in the program been executed?

Branch coverage - Has each branch of each control structure been executed?

Condition coverage - Has each Boolean sub-expression evaluated both to true and false?

# Background

## FTVP font-end tests

We have two application ends there are html5 (H5) and mobile web (M). Mobile web and H5 have shared the graphics that is the core framework for our app. At our font-end, whatever H5 and M, we use AngularJS to build font-end structure.

|  |  |  |
| --- | --- | --- |
| Html5 | Unit Test | Integration Test (E2E) |
| Graphics | practicable | not included |
| Angular | hard to test | not included |

|  |  |  |
| --- | --- | --- |
| Mobile Web | Unit Test | Integration Test (E2E) |
| Graphics | practicable | not included |
| Angular | practicable | not included |

Notes: Integration Test will be involved by TAF or other automatic test tools.

At this documents it will focus on Mobile Web (both graphics and angular) of Unit Test, since the graphic part is common for H5 and M.

## Usage tools or framework

|  |  |  |
| --- | --- | --- |
| **Name** | **URL** | **Description** |
| Karma-coverage | <https://www.npmjs.com/package/karma-coverage> | Code coverage tool |
| Jasmine | <http://jasmine.github.io/> | Unit test framework |
| ngMock | https://code.angularjs.org/1.3.16/angular-mocks.js | Mock Angular services |

# Part 1 Unit Test for VP

## Setup unit test environment (Jasmine)

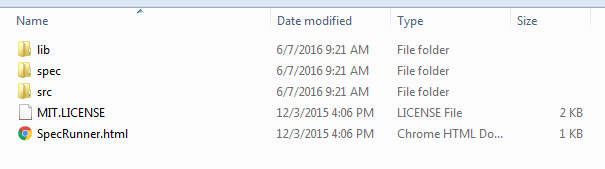
Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests.

Download <https://github.com/jasmine/jasmine/releases>

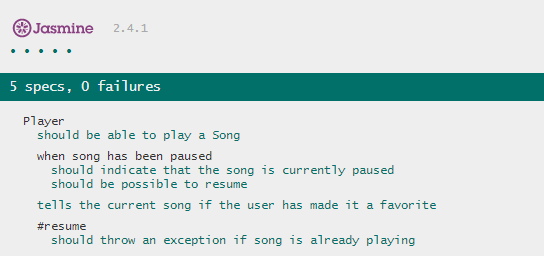
You can get the latest version to use the project.

We can download the jasmine-standalone-2.4.1.zip and unzip to your location.

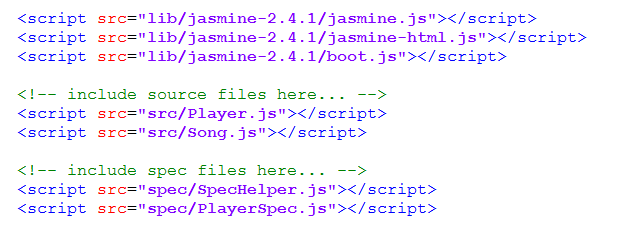
Below is the basic structure of jasmine.



Open the SpecRunner.html, then you can see the sample project.



Open the source code of SpecRunner.html. You can find the parts of codes:



Notes the order of scripts you included. Source files should be included before the Test files.

## Unit Test for Graphics

I’ve chosen the Core.js since it contains many individual functions and created the CoreSpec.js to write unit test codes.

Included the files at below order.

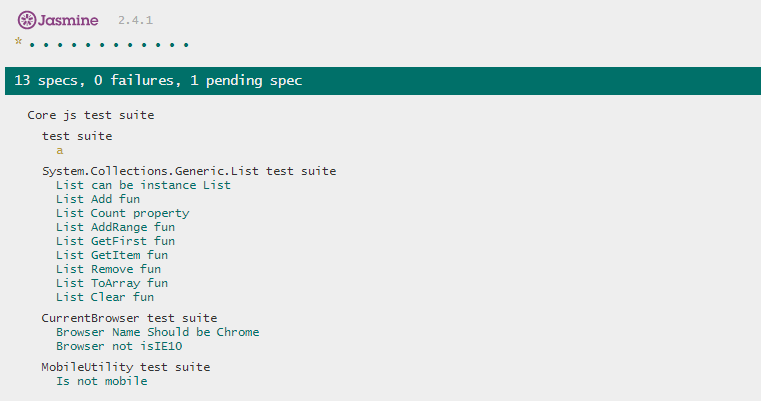
<**script src="src/Core.js"**></**script**><**script src="spec/CoreSpec.js"**></**script**>

**CoreSpec.js**

describe(**'Core js test suite'**, **function** () {  
 xdescribe(**'test suite'**, **function** () {  
 **var** a;  
 beforeEach(**function** () {  
 a = **'ftvp'**;  
 });  
 it(**'a'**, **function** () {  
 expect(a).*toEqual*(**'ftvp'**);  
 });  
 });  
  
 describe(**'System.Collections.Generic.List test suite'**, **function** () {  
 **var** listObj;  
 beforeEach(**function** () {  
 listObj = **new *System***.**Collections**.**Generic**.*List*();  
 listObj.Add(**'listItem01'**);*//0* listObj.Add(**'listItem02'**);*//1* listObj.Add([23, **'er'**]);*//2* });  
 it(**'List can be instance List'**, **function** () {  
 expect(listObj **instanceof *System***.**Collections**.**Generic**.*List*).*toEqual*(**true**);  
 });  
 it(**"List Add fun"**, **function** () {  
 expect(**'listItem01'**).*toEqual*(listObj.**itemArray**[0])  
 });  
 it(**"List Count property"**, **function** () {  
 expect(listObj.Count).*toEqual*(3);  
 });  
 it(**"List AddRange fun"**, **function** () {  
 listObj.AddRange([**'aaa'**, **'bbb'**, **false**, 12]);*//3+4 = 7* expect(listObj.Count).*toEqual*(7);  
 });  
 it(**"List GetFirst fun"**, **function** () {  
 expect(listObj.GetFirst()).*toBe*(**'listItem01'**);  
 expect(listObj.GetFirst()).*toEqual*(**'listItem01'**);  
 });  
  
 it(**"List GetItem fun"**, **function** () {  
 listObj.Add(**'fff'**);  
 expect(listObj.GetItem(3)).*toEqual*(**'fff'**);  
 });  
 it(**"List Remove fun"**, **function** () {  
 listObj.Add(**'bbb'**);  
 listObj.Add(23);  
 expect(listObj.GetItem(3)).*toEqual*(**'bbb'**);  
 listObj.Remove(**'bbb'**);  
 expect(listObj.GetItem(3)).*toEqual*(23);  
 });  
 it(**"List ToArray fun"**, **function** () {  
 **var** isArr = listObj.ToArray();  
 expect(isArr **instanceof** Array).*toBe*(**true**);  
 });  
  
 it(**"List Clear fun"**, **function** () {  
 listObj.Clear();  
 expect(listObj.Count).*toEqual*(0);  
 });  
 });  
 describe(**'CurrentBrowser test suite'**, **function** () {  
 **var** browserObj;  
 beforeEach(**function** () {  
 browserObj = ***CurrentBrowser***;  
 });  
 it(**'Browser Name Should be Chrome'**, **function** () {  
 **var** bn = browserObj.browserName();  
 expect(bn).*toEqual*(**'Chrome'**);  
 });  
 it(**'Browser not isIE10'**, **function** () {  
 expect(browserObj.isIE10()).*toEqual*(**false**);  
 })  
 });  
  
 describe(**'MobileUtility test suite'**, **function** () {  
 it(**'Is not mobile'**, **function** () {  
 expect(***MobileUtility***.IsMobile()).*toBe*(**false**);  
 });  
 });  
  
});

At this case my codes only include List, CurrentBrowser and MobileUtility functions.

It will get below results.



## Unit Test for Angular

We should use ngMock module to test angular module and include angular referent libraries. It can test controllers,directives and filters.

Included the files at below order.

<**script src="../lib/angular.min.js"**></**script**>  
<**script src="../lib/angular-route.min.js"**></**script**>  
<**script src="../lib/angular-mocks.js"**></**script**>  
<**script src="../app/client/src/js/filters/****vpAlarmStateDisplayFilter.js"**></**script**>  
*<!--spec-->*<**script src="spec/vpAlarmStateDisplayFilterSpec.js"**></**script**>

**vpAlarmStateDisplayFilter.js**

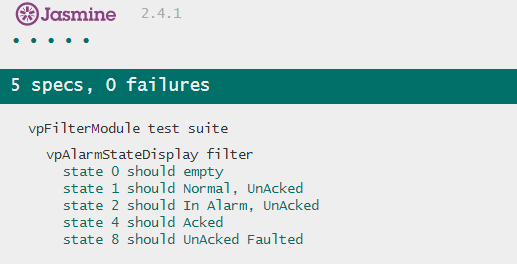
angular.module(**'vpFilterModule'**)

.filter(**'vpAlarmStateDisplay'**, **function**() {  
 **return function**(state) {  
 **var** parsedState = **''**;  
 **if**(!state){  
 **return** parsedState;  
 }  
  
 **if** ((state & 1) !== 0) {  
 parsedState += **'Normal,'** + **' '**;  
 }  
  
 **if** ((state & 2) !== 0) {  
 parsedState += **'In Alarm,'** + **' '**;  
 }  
  
 **if** ((state & 4) !== 0) {  
 parsedState += **'Acked'** + **' '**;  
  
 } **else** {  
 parsedState += **'UnAcked'** + **' '**;  
 }  
  
 **if** ((state & 8) !== 0) {  
 parsedState += **'Faulted'**;  
 }  
  
 **return** parsedState;  
 }  
});

**vpAlarmStateDisplayFilterSpec.js**

describe(**'vpFilterModule test suite'**, **function** () {  
 beforeEach(**angular**.**mock**.module(**'vpFilterModule'**));  
 **var** $filter;  
 describe(**'vpAlarmStateDisplay filter'**, **function** () {  
 beforeEach(**angular**.**mock**.inject(**function** (\_$filter\_) {  
 $filter = \_$filter\_;  
 }));  
 it(**'state 0 should empty'**, **function** () {  
 **var** r = $filter(**'vpAlarmStateDisplay'**)(0);  
 expect(r).toEqual(**''**);  
 });  
 it(**'state 1 should Normal, UnAcked'**, **function** () {  
 **var** r = $filter(**'vpAlarmStateDisplay'**)(1);  
 expect(r).toEqual(**'Normal, UnAcked '**);  
 });  
 it(**'state 2 should In Alarm, UnAcked'**, **function** () {  
 **var** r = $filter(**'vpAlarmStateDisplay'**)(2);  
 expect(r).toEqual(**'In Alarm, UnAcked '**);  
 });  
 it(**'state 4 should Acked'**, **function** () {  
 **var** r = $filter(**'vpAlarmStateDisplay'**)(4);  
 expect(r).toEqual(**'Acked '**);  
 });  
 it(**'state 8 should UnAcked Faulted'**, **function** () {  
 **var** r = $filter(**'vpAlarmStateDisplay'**)(8);  
 expect(r).toEqual(**'UnAcked Faulted'**);  
 });  
 });  
});

Jasmine runner results:



# Part 2 Code Coverage Practice

This part is the main content of this documents, it will use code coverage tool to run Parts 1 examples. CoreSpec.js and vpAlarmStateDisplayFilterSpec.js are our test codes ;

Core.js and vpAlarmStateDisplayFilter.js are our source codes.

## Setup code coverage environment (Karma-Coverage)

Step1: Install node.js

Go to <https://nodejs.org/en/> and download and install it. We want to use npm tool to download others packages.So install it firstly.

Step2: Install karma and karma-coverage

npm install -g karma

npm install -g karma-coverage

Notes: Install it as global to run karma easily.

Step3: Install some plugins

npm install -g karma-jasmine (adapt the jasmine at karma)

npm install -g karma-chrome-launcher (use chrome browser to run karma)

Step4: Put our sources and tests files into the test project

At our project , we have many sources files to test, it will generate many tests files also. We should manage them kindly, So we can put it into ‘test’ directory.

For now we have set up the karma-coverage environment.

## Configure karma-coverage

Below is karma.conf.js for this example:

*// karma.conf.js -- this config file is used for UnitTest Project***module**.exports = **function** (config) {  
 config.set({  
 **basePath**:**'./'**,  
 **frameworks**: [**'jasmine'**],  
 **files**: [  
 ,**'lib/angular.min.js'** ,**'lib/angular-route.min.js'** ,**'lib/angular-mocks.js'** ,**'app/client/src/js/app.js'** ,**'app/client/src/js/filters/vpAlarmStateDisplayFilter.js'** ,**'unit\_test/spec/vpAlarmStateDisplayFilterSpec.js'** ,**'unit\_test/src/Core.js'** ,**'unit\_test/spec/CoreSpec.js'** ],  
 *// coverage reporter generates the coverage* **reporters**: [**'progress'**, **'coverage'**],  
 **preprocessors**: {  
 *// source files, that you wanna generate coverage for  
 // do not include tests or libraries  
 // (these files will be instrumented by Istanbul)* **'app/client/src/js/filters/vpAlarmStateDisplayFilter.js'**: **'coverage'** ,**'unit\_test/src/Core.js'**:**'coverage'** },  
 **plugins**: [  
 **'karma-coverage'**,  
 **'karma-jasmine'**,  
 **'karma-chrome-launcher'** ],  
 **browsers**: [**'Chrome'**],  
 *// optionally, configure the reporter* **coverageReporter**: {  
 **type**: **'html'**,  
 **dir**: **'coverage\_ut'** }  
 });  
};

Notes:

*frameworks is our test framework to use*

*files are the watched files, it usually include library , source and specs files.*

*Preprocessors are the detected files for code coverage.*

*coverageReporter will include the format for report and store directory.*

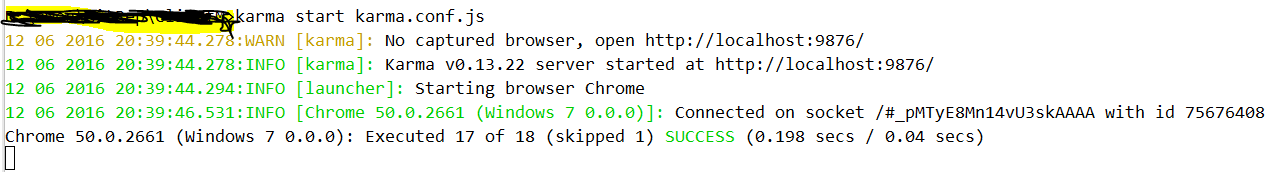
If you want to understand all config you can visit: <https://github.com/karma-runner/karma-coverage/blob/master/docs/configuration.md>

## Run it

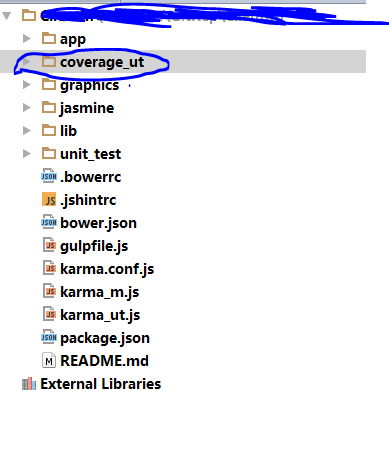
All the prepared things are OK , we can run karma. Use below command:

karma start karma.conf.js

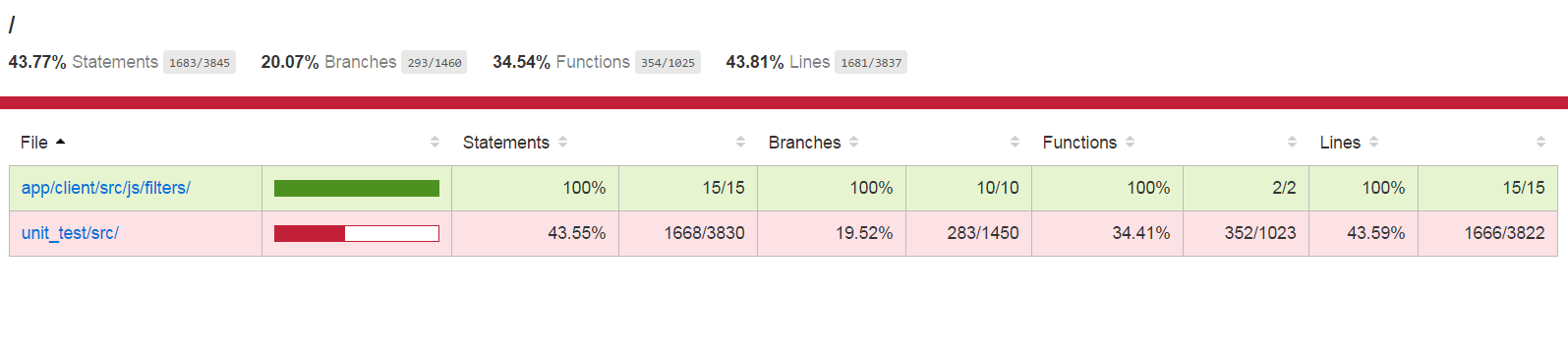
The shell will show like this:



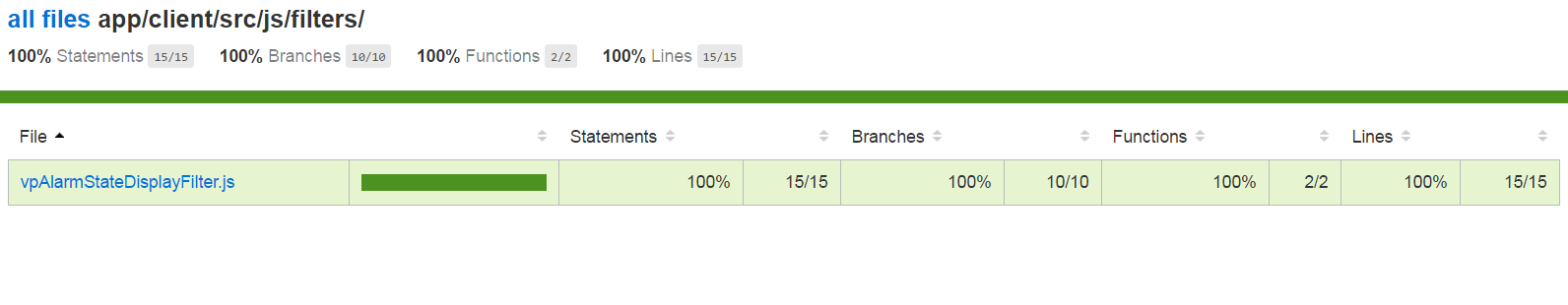
After run the command, code coverage report be generated at our project.

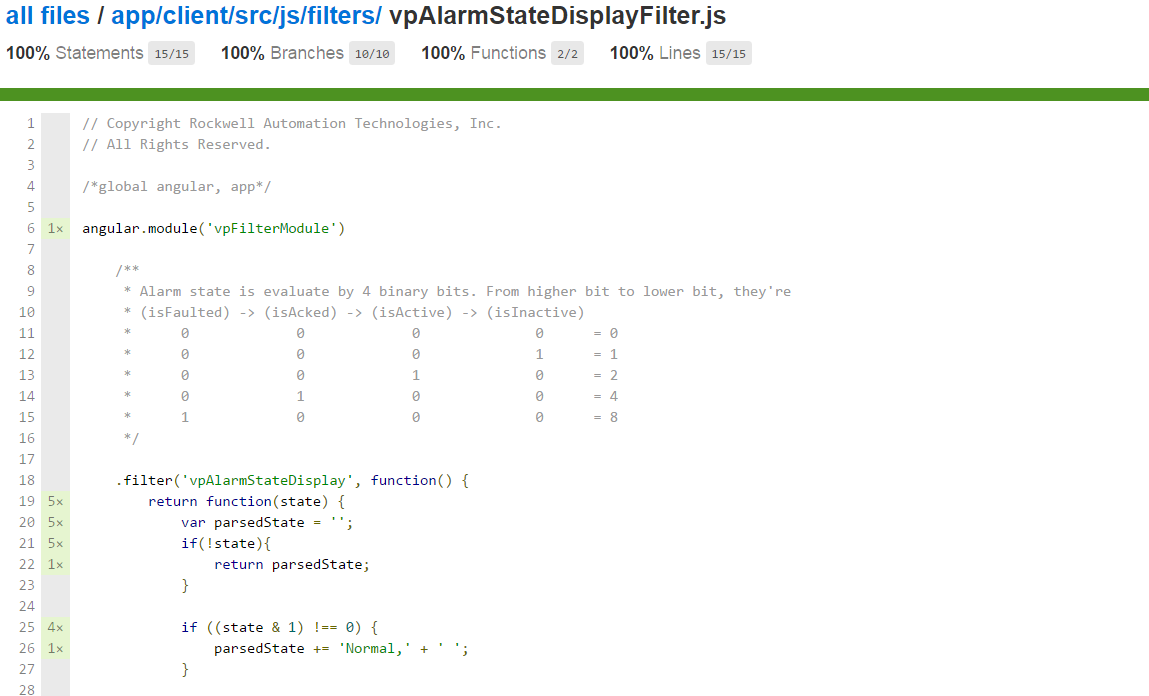


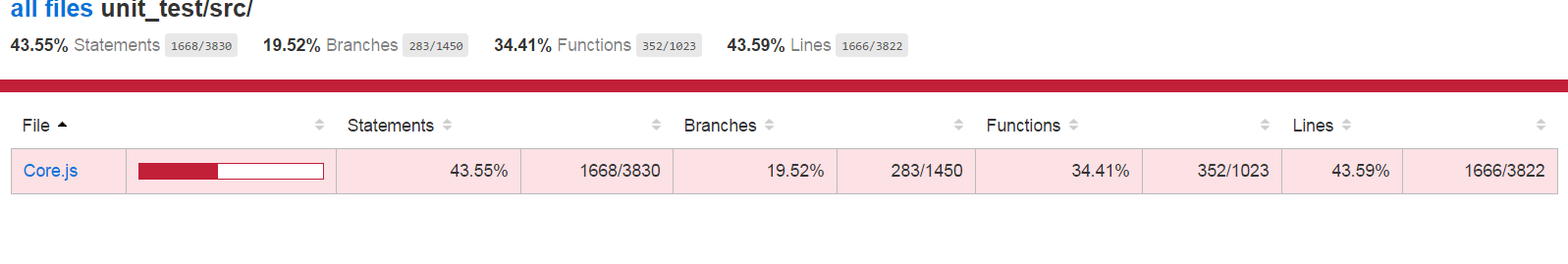
Open the coverage\_ut folder and click the index.html , it will generate the code coverage results:



Here some results for screenshots









# Conclusion

# Reference

<https://github.com/karma-runner/karma-coverage>