**Roku**

Unit Test Framework for Roku Channels (BETA)

Software Design Document

| Author | Revision | Comments | Date |
| --- | --- | --- | --- |
| Oleksandr Raiev  Roman Kuzik  Vitalii Mazurok | 1.0 | Initial draft |  |
|  |  |  |  |

List of Changes

Revision 1.0:

| Section / Paragraph | Change/Addition Description |
| --- | --- |
|  |  |
|  |  |
|  |  |

Contents

[1 Introduction 5](#_Toc441516161)

[1.1 Purpose 5](#_Toc441516162)

[1.2 Scope 5](#_Toc441516163)

[1.2.1 Current Implementation 5](#_Toc441516164)

[1.2.2 Possible new features and improvements 5](#_Toc441516165)

[1.3 Overview 5](#_Toc441516166)

[2 System Architecture 6](#_Toc441516167)

[2.1 System diagram 6](#_Toc441516168)

[2.2 Sequence diagram 7](#_Toc441516169)

[3 Major components features 8](#_Toc441516170)

[3.1 Test Runner 8](#_Toc441516171)

[3.2 Test Suite 8](#_Toc441516172)

[3.3 Item Generator 10](#_Toc441516173)

[4 Usage and integration 11](#_Toc441516174)

[5 Example 14](#_Toc441516175)

# Introduction

## Purpose

The purpose of this document is to document a clear and comprehensive design of the system. It documents major classes design and relationships and is to be used as a guideline in development process. This document is targeted to software developers who will be developing Roku channels with BrightScript.

## Scope

Create framework for writing code coverage unit test, run these tests and collect statistic.

### Current Implementation

Current implementation consists of Test Runner which collects all test cases and run them one by one, Base Test Suite contains assertion methods and Item Generator which allows to create different objects by given scheme.

### Possible new features and improvements

* Find a better way to run tests.
* Add more assertion methods.
* Add mocks.
* Update framework according to SDK2.0 needs.
* Integration Test Framework into Roku BrightScript eclipse plugin.
* Creation generalize tests for channels

## Overview

Test Runner collects all test suites with their test cases under the given directory. When all test cases were collected Test Runner runs them one by one and collects the results. After running all the test cases Test Runner outputs the statistic log with selected level of verbosity.

Every test suite must extend BaseTestSuite and must contain at least one test case.

# System Architecture

## System diagram

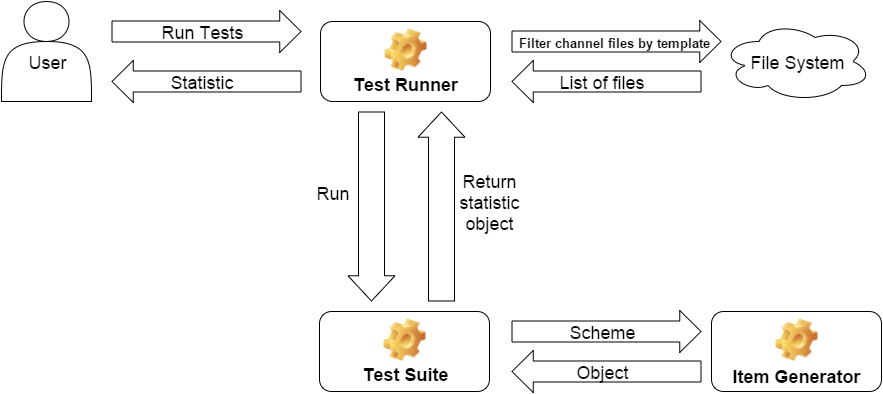


Figure 1. System architecture diagram

## Sequence diagram

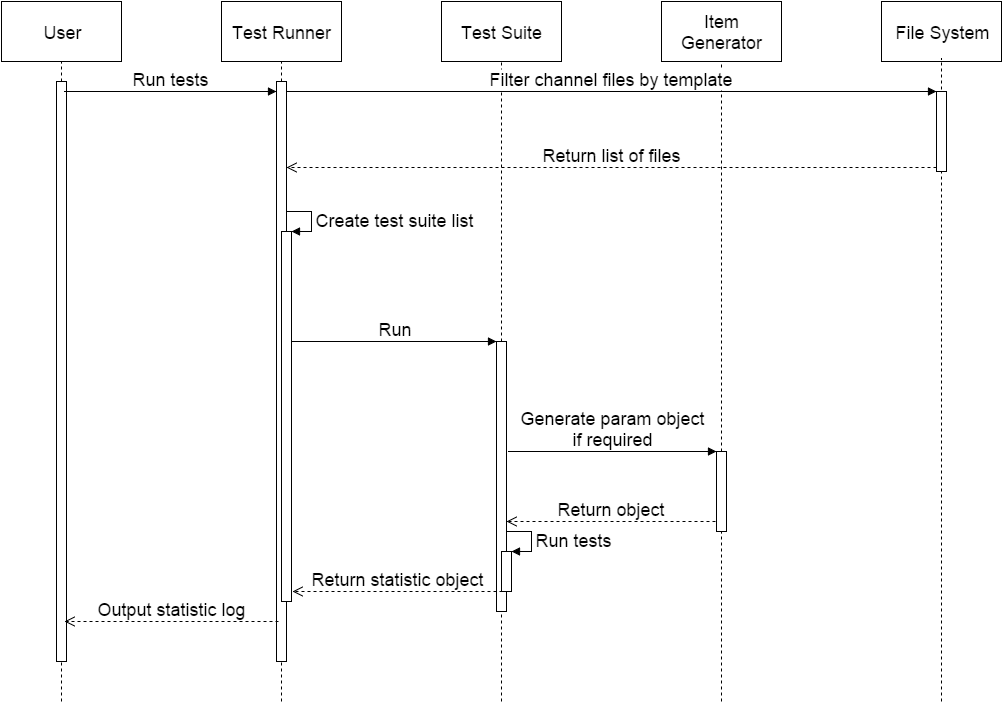


Figure 2. Flow diagram

# Major components features

## Test Runner

Test Runner is main object in this framework. There are 4 properties of this this object that can be set to create instance of it:

* verbosity – level of output log verbosity.
* testsDirectory – path to directory which contains all tests.
* testFilePrefix - string to identify all test files in testsDirectory and subdirectories.
* testSuitePrefix - string to identify all test suites in test files.

Test Runner searches for .brs files with testFilePrefix in testsDirectory and subdirectories. When list of files is ready Test Runner parses these files and extracts from them test suites with testSuitePrefix. Every test suite has a list of test cases. Test Runner compiles a single list of all test cases and then run them one by one.

When Test Runner ran all test cases it outputs a statistic log into console with selected verbosity level.

## Test Suite

BaseTestSuite is parent class for all test suites. The main feature of it is a list of assertion methods. All these methods have one common argument “*msg*”. This is an error message returned by assertion in case of failure.

Table 1. Assertions list

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Arguments** | **Description** |
| **Fail** | *msg (string)* An error message. Default value: "Error". | Fail immediately, with the given message. |
| **AssertFalse** | *expr (dynamic)* An expression to evaluate. | Fail the test if the expression is true. |
| **AssertTrue** | *expr (dynamic)* An expression to evaluate. | Fail the test unless the expression is true. |
| **AssertEqual** | *first (dynamic)* A first object to compare.  *second (dynamic)* A second object to compare. | Fail if the two objects are unequal as determined by the '<>' operator. |
| **AssertNotEqual** | *first (dynamic)* A first object to compare.  *second (dynamic)* A second object to compare. | Fail if the two objects are equal as determined by the '=' operator. |
| **AssertInvalid** | *value (dynamic)* A value to check. | Fail if the value is not invalid. |
| **AssertNotInvalid** | *value (dynamic)* A value to check. | Fail if the value is invalid. |
| **AssertAAHasKey** | *array (dynamic)* A target array.  *key (string)* A key name. | Fail if the array doesn't have the key. |
| **AssertAANotHasKey** | *array (dynamic)* A target array.  *key (string)* A key name. | Fail if the array has the key. |
| **AssertAAHasKeys** | *array (dynamic)* A target associative array.  *keys (object)* A key names array. | Fail if the array doesn't have the keys list. |
| **AssertAANotHasKeys** | *array (dynamic)* A target associative array.  *keys (object)* A key names array. | Fail if the array has the keys list. |
| **AssertArrayContains** | *array (dynamic)* A target array.  *value (dynamic)* A value to check.  *key (object)* A key name for associative array. | Fail if the array doesn't have the item. |
| **AssertArrayNotContains** | *array (dynamic)* A target array.  *value (dynamic)* A value to check.  *key (object)* A key name for associative array. | Fail if the array has the item. |
| **AssertArrayContainsSubset** | *array (dynamic)* A target array.  *subset (dynamic)* An items array to check. | Fail if the array doesn't have the item subset. |
| **AssertArrayNotContainsSubset** | *array (dynamic)* A target array.  *subset (dynamic)* A items array to check. | Fail if the array have the item from subset. |
| **AssertArrayCount** | *array (dynamic)* A target array.  *count (integer)* An expected array items count. | Fail if the array items count <> expected count. |
| **AssertArrayNotCount** | *array (dynamic)* A target array.  *count (integer)* An expected array items count. | Fail if the array items count = expected count. |
| **AssertEmpty** | *item (dynamic)* An array or string to check. | Fail if the item is not empty array or string. |
| **AssertNotEmpty** | *item (dynamic)* An array or string to check. | Fail if the item is empty array or string. |
| **AssertArrayContainsOnly** | *array (dynamic)* A target array.  *typeStr (string)* An item’s type name. | Fail if the array doesn't contains items of specific type only. |

## Item Generator

Item Generator is an object to generate random items according to specified scheme. The scheme represents the structure of our desired object. Scheme could be one of simple types (integer, string or float), array or associative array.

If scheme is one of simple type, Item Generator returns random value for this type.

The array scheme shall include the string array of type names that we want to have in the resulting array item.

For example:

scheme: [“int”, “string”, “string”, “boolean”]

result: [5, “ghn56f”, “7sb2td”, true]

The associative array scheme shall have the following structure:

{ propertyName1: "propertyType1"

propertyName2: "propertyType2"

...

propertyNameN: "propertyTypeN" }

Item Generator will return the associative array object with specified property names and random values of specified property type.

For example:

scheme:

{

id: "integer"

name: "string"

address: {city: “string”, street: “string”}

active: “boolean”

}

result:

{

id: 75

name: "ht7d9nt5zp"

address: {city : “9gbst5mpw7”, street : “nfdt7ns5p2”}

active: false

}

# Usage and integration

Unit test framework is located under //depot/edelivery/STB-Client/apps/common/framework/testFramework/ folder.

To run your unit tests, follow these instruction:

1. Get latest revision of framework.
2. (Re)Build your project:
   1. Get latest revisions of files:

//depot/edelivery/STB-Client/apps/app.mk

//depot/edelivery/STB-Client/apps/framework.mk

//depot/edelivery/STB-Client/apps/tests.mk.

* 1. Create Makefile in the root of your project if you don’t have it. Include these files in your Makefile in following sequence:

1. framework.mk.
2. app.mk.
3. tests.mk.
   1. Run *make INCLUDETESTS=true* command to build your project with included framework and all tests or just *make* to build without framework and tests.
4. Create new folder “tests” under source directory of your project: “pkg:/source/tests”.

This will be the root folder for all your unit tests. In this folder you can also create subfolders for every test suites collection.

1. Create test suite files in “tests” folder or subfolders.

Create new .brs file for test suite. Default prefix for test suite files is “Test\_\_”. You can use any prefix you want just don’t forget to specify it in the next step. In new file define function with prefix “TestSuite\_\_” (default prefix for test suite) *Function TestSuite\_\_Sample() as Object*. This function will return test suite object. Extend new test suite object from BaseTestSuite *this = BaseTestSuite()*, set its name *this.Name = "SampleTestSuite"* and add test cases:

*this.addTest("CheckInputItemType",TestCase\_\_MyApp\_TestObjectName\_CheckInputItemType)*. TestCase\_\_MyApp\_TestObjectName\_CheckInputItemType is a test function.

*Function TestCase\_\_MyApp\_TestObjectName\_CheckInputItemType() as string*

*inputObject = 0*

*result = m.targetTestObject.SomeFunction(inputObject)*

*return m.assertInvalid(result)*

*End Function*

1. Run all your tests.

To run test you have to add this code *TestRunner().Run()* to entry point of a channel. This should be a first line of RunUserInterface function. By setting second argument of function TestRunner you can run tests from specific folder *TestRunner(1, “pkg:/source/tests/SomeFolder”).Run()* by setting second and third arguments you can run tests from specific test suite *TestRunner(1, “pkg:/source/tests/SomeFolder”, “TestSuite\_\_Sample”).Run().*

1. See the statistic log.

First argument of function TestRunner is verbosity level of statistic log. There are three verbosity levels in this framework:

“0” – basic level

“1” – normal level

“2” – detailed level.

With basic level framework outputs total number of tests, number of passed, failed tests, number of tests that caused crash and time spent for running these tests:

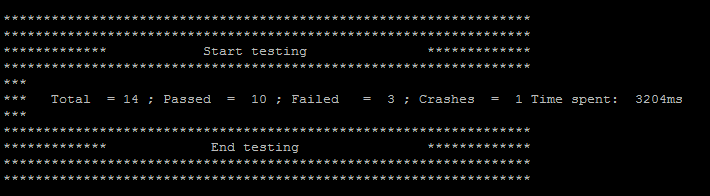


Figure 3. Basic verbosity level

Normal level includes the list of all tests with their names and results:

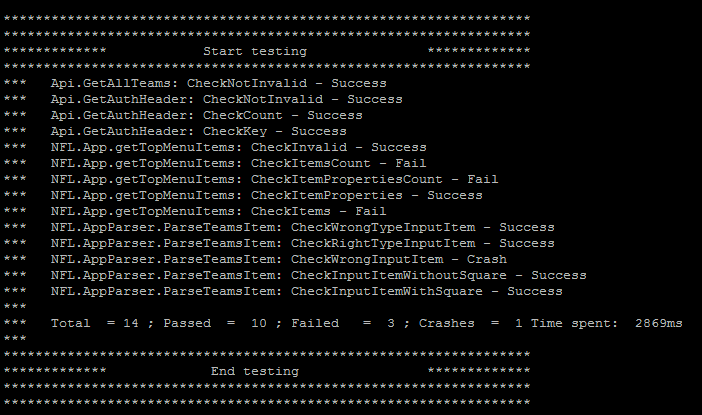
**

Figure 4. Normal verbosity level

Detailed level shows verbose statistic by every test suite and error messages of failed tests:

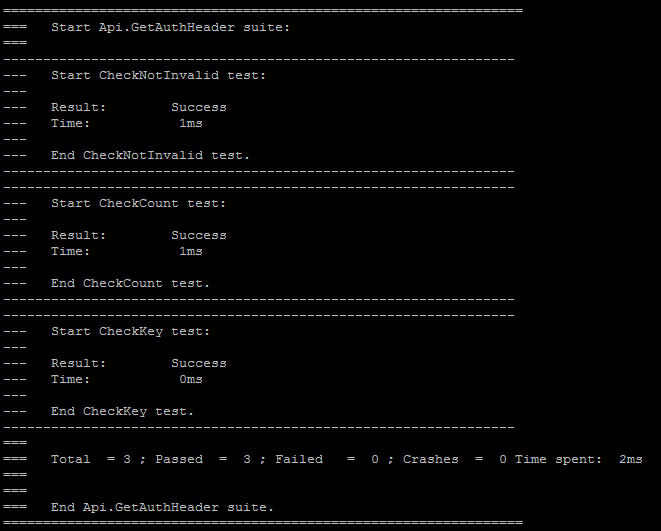


Figure 5. Detailed verbosity level

# Example

Please find example of a test suite below. You can also find this sample in perforce:

//depot/edelivery/STB-Client/apps/common/framework/testFramework/tests/sample/Test\_\_Sample.brs.

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

'\* Copyright Roku 2011-2016

'\* All Rights Reserved

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

' Functions in this file:

'

' TestSuite\_\_Sample

' TestCase\_\_MyApp\_TestObjectName\_CheckInputItemType

' TestCase\_\_MyApp\_TestObjectName\_CheckInputItem

' TestCase\_\_MyApp\_TestObjectName\_CheckExpectedItem

'----------------------------------------------------------------

' Main setup function.

'

' @return A configured TestSuite object.

'----------------------------------------------------------------

Function TestSuite\_\_Sample() as Object

' Inherite your test suite from BaseTestSuite

this = BaseTestSuite()

' Test suite name for log statistics

this.Name = "SampleTestSuite"

' Target testing object. To avoid the object creation in each test

' we create instance of target object here and use it in tests as m.targetTestObject.

this.targetTestObject = MyApp\_\_TargetObject()

' Add tests to suite's tests collection

this.addTest("CheckInputItemType", TestCase\_\_MyApp\_TestObjectName\_CheckInputItemType)

this.addTest("CheckInputItem", TestCase\_\_MyApp\_TestObjectName\_CheckInputItem)

this.addTest("CheckExpectedItem", TestCase\_\_MyApp\_TestObjectName\_CheckExpectedItem)

return this

End Function

'----------------------------------------------------------------

' Result should be invalid if type of inputObject

' is not the one we expect.

'

' @return An empty string if test is success or error message if not.

'----------------------------------------------------------------

Function TestCase\_\_MyApp\_TestObjectName\_CheckInputItemType() as string

inputObject = 0

result = m.targetTestObject.SomeFunction(inputObject)

return m.assertInvalid(result)

End Function

'----------------------------------------------------------------

' SomeFunction should return not invalid object for

' any random input associative array.

'

' @return An empty string if test is success or error message if not.

'----------------------------------------------------------------

Function TestCase\_\_MyApp\_TestObjectName\_CheckInputItem() as string

scheme = {

key1 : "integer"

key2 : "string"

key3 : "boolean"

key4 : {subKey1: "string"}

}

inputObject = ItemGenerator().getItem(scheme)

result = m.targetTestObject.SomeFunction(inputObject)

return m.assertNotInvalid(result)

End Function

'----------------------------------------------------------------

' SomeFunction should return object equal to expectedItem.

'

' @return An empty string if test is success or error message if not.

'----------------------------------------------------------------

Function TestCase\_\_MyApp\_TestObjectName\_CheckExpectedItem() as string

inputObject = {

key1 : "Value1"

key2 : "Value2"

key3 : "Value3"

key4 : {subKey1: "subValue1"}

}

expectedItem = {

expectedKey1 : "Value1"

expectedKey2 : "Value2"

expectedKey3 : "Value3"

expectedKey4 : {expectedSubKey1: "subValue1"}

}

resultItem = m.targetTestObject.SomeFunction(inputObject)

return m.assertEqual(expectedItem, resultItem)

End Function