ProtoTest-Motorola Nightshade Framework Setup Instructions

Melissa Tondi

Brian Kitchener

Roland Burrows

May 2014

Table of Contents

Nightshade Overview

Framework Summary

Nightshade is a C# Eggplant Test Runner, built in Visual Studio 2012 with .Net 4.0. Specifically, it's built on top of Gallio and MbUnit. MbUnit is a unit testing framework similar to nUnit, but with more advanced features for UI automation. Gallio is a unit test runner that contains advanced reporting along with both GUI and Command line execution. In addition, tests can be executed from Visual Studio using the TestDriven.net plugin.

The problem that the Framework solves

Motorola desired the ability to run a 100 hour long multi-device stability test against a number of different mobile devices and operating systems using Testplant's Eggplant automation tool. However, the Eggplant GUI and scripting language SenseTalk is limited in its ability to execute these types of tests for the reasons outlined below:

- Memory Issues - Eggplant Runs out of memory around 40 hours in.
- Multi Device Support - The Eggplant GUI offers only basic multi device testing and didn't support the scope, complexity, and robustness required by Motorola.
- SenseTalk scripting limitations. Have to learn a new scripting language, and are required to work around its limitations was time consuming. Resulting code was difficult to reuse.
- Code reusability - Reusable code is hard to create and maintain in SenseTalk, but it's much easier in a strongly typed language like C# using the Page Object design pattern.
- Controlling test execution from outside of Eggplant allows us to customize and stabilize Eggplant to Motorola's needs.

Framework Architecture [Overview]

Nightshade requires .Net 4.0, Gallio version 3.14, Visual Studio 2012 or later, and Eggplant installed on the local machine with a valid license. Nightshade launches Eggplant Drive as a java jar, and communicates to it via an XmlRpc service. The Nightshade runner connects to the devices using an IP address, so the devices must be reachable via the IP addresses supplied in the config file. All test settings are configurable either in the App.config file or by setting the properties in the Config class.

Nightshade is built using multiple layers of abstraction. This is to organize and categorize where information lives to help maximize code reuse. Fundamentally, all code comes down to a series of conditions. We want to avoid having a giant series of nested if statements, as debugging and maintaining that code isn't realistic.

Framework Architecture (Code)

Tests should all inherit the EggPlantTestBase. This class automatically starts/stops eggDrive and handles connecting to hosts. The test class should contain a number of test methods, each marked with the [Test] attribute. Each method will call a series of page interface methods. Inside is a switch, which looks in our App.config file for the current device, and returns page objects for that type of device. This switch makes the tests device agnostic, and can be run on different modes by simply changing the config file value.

Tests are written using MbUnit attributes, and are executed using Gallio Icaurs. Icarus allows the tests to be filtered, and a sub-set executed. The MbUnit attributes are what provide all the logic to the tests. For example, to repeat a test 10 times, we simply add the [Repeat(10)] attribute to our test, and it automatically repeats. There is one final variable called SuiteRepetitions, which repeats the entire suite of tests the number of times specified.

Each test will build an HTML style report containing all diagnostic information (including a step-by-step list of commands), all log statements, and a screenshot of the device under failure. Each test should be marked as passed, failed, or skipped. Tests that are not valid for a specific device will be marked as skipped.

The test suite is configured using the App.config file. All configurable information, such as IP addresses, user names, etc, should be contained within.

Framework Test Execution Flow

When a test is started, the following functionality occurs:

1) Create batch file to start drive.
2) Stop Eggplant if its currently running
3) Try to start Eggplant up to 5 times.
4) Start a Suite Session using the Config.SuitePath. (defaults to MotorolaMaster.suite)
5) Connect to a host.
6) Execute commands against host.

Nightshade Framework Core

By.cs - This contains the information on how to build a 'locator' string, which follows the Eggplant syntax.

Common.cs - Random helper methods that don't have another place to live.

Config.cs - This file reads from the app.config, and stores all values in properties. To check or update a config setting in code, use Config.VariableName. Also contains methods for reading/writing to the App.config file.

EggplantDriver.cs - This contains helper methods for all major commands available in eggplant, including starting/stopping eggplant, starting a suite, clicking an element, etc.

EggplantElement.cs - This is an abstract representation of a UI element. Can be defined using the By class for ease of use. Wraps the EggplantDriver commands in a nice API.

EggPlantScript.cs - Contains the methods necessary for executing and validating an eggplant .script file. Not being currently used.

EggplantTestBase.cs - Test base class, all test classes should inherit from this. Framework functionality is triggered by the FixtureSetup, SetUp, TearDown, and FixtureTearDown methods.

IEggplantDriveService.cs - This is a client used to communicate with the XMLRPC service that Eggplant creates when launched in drive mode.

SearchRectangle.cs - Builds a rectangle to be used when building locators.

VerificationErrors.cs - Non-terminating error, which will fail the test but not halt script execution.

Framework Custom Attributes

TestTeardownAttribute.cs - This defines a method to be used after the test is finished. Used for script cleanup.

RepeatForConfigValueAttribute.cs - Reads from the App.Config for a Key, converts it to an int, and repeats a test that many times.

RepeatForTime.cs - Repeat a test for a given amount of time.

RepeatOnFailureAttribute.cs - Repeat a test if it fails.

Test Class - This corresponds to a 'suite' of tests to be executed, and tests should be organized into functionally-grouped classes. Tests will be marked with the [Test] attribute, and can be executed independently, or as an entire suite.

Test Step Interface - These are high level functions like “OpenApplication()”, or “TurnOnWiFi()”. These commands don’t know or care how they are implemented, as they are an interface. The test should be a series of these high level commands written in fairly plain english syntax, and the interface commands can be implemented for many different types of devices. (For example, “turning on wifi” might be the same for every windows CE device, yet there will probably be different access paths for different versions of the “Contacts” app. In this case we would create a new class that implements the Contacts app commands, and reuse the rest. Inside of each test step will be a series of page object commands.

Page Objects are classes that store information about a particular screen. The page object contains all the EggPlant elements (such as buttons, fields, etc), and any reusable methods or logic available to the user on that page. Any changes made to a device, such as updating an app, requires only updating the page object.

EggPlant Elements - The Eggplant Element class gives a simple a clean api for manipulating UI elements, and will contain methods for clicking/typing/verifying UI elements. They are instantiated with an advanced By locator (similar in design to webdriver), that allows specifying optional parameters, such as screen area, wait time, and click offset. This class executes Eggplant Driver Commands.

Eggplant Driver is the class that communicates with Eggplant. It handles starting and stopping the Eggplant application, and sending messages to and from it. All commands eventually get sent to eggplant via the driver class. Any sensetalk command that is not supported by the Nightshade API can be sent using driver.Execute(“command”).

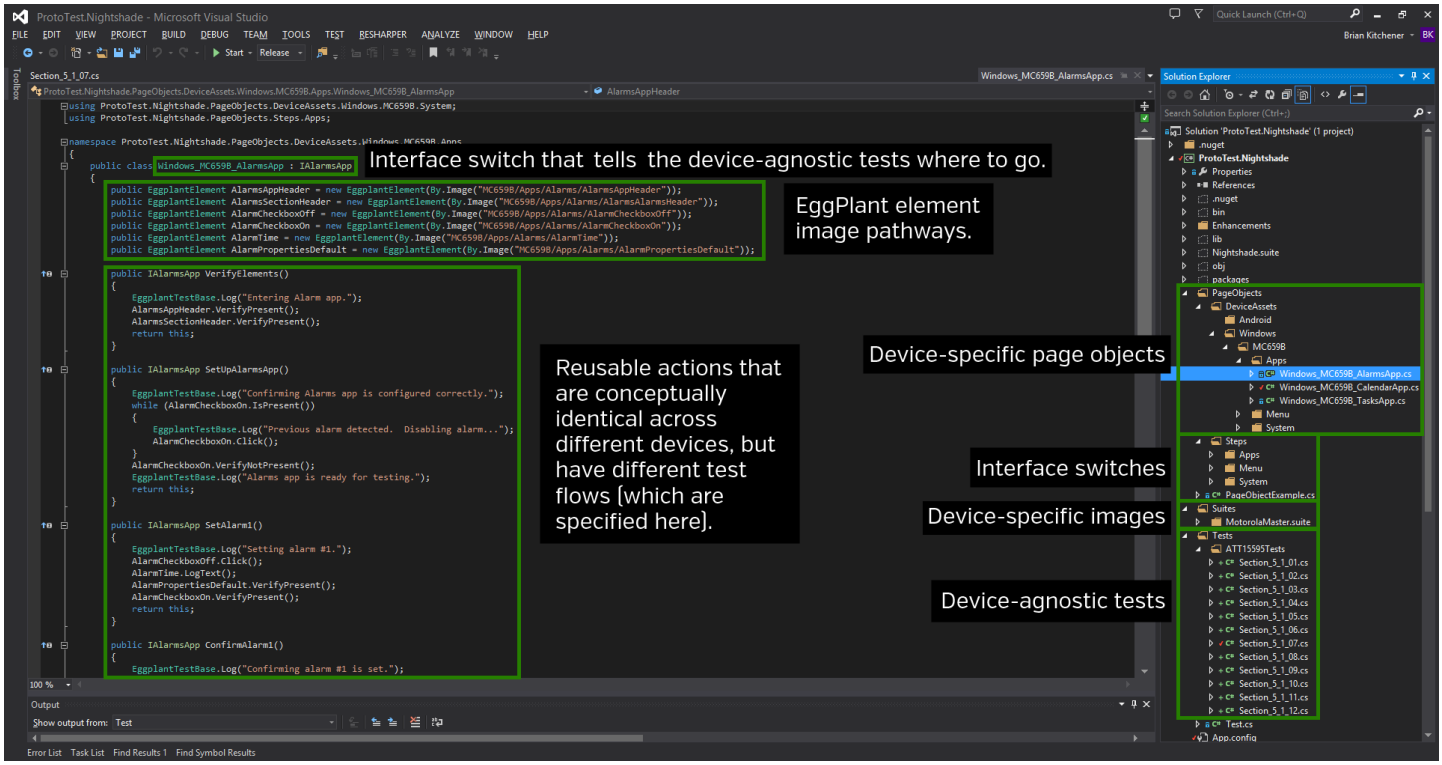
Nightshade Setup Instructions

Setup for Windows

1. Download and set up EggPlant (v14.1 minimum) and Visual Studio (2010 minimum, 2012 preferred) for Windows.
2. Download and set up TestDriven.net for Visual Studio.
3. Download and set up Gallio Icarus Test Runner for Windows.
4. Download the ProtoTest.Nightshade solution from Github: https://github.com/ProtoTest/ProtoTest.Nightshade
5. In Microsoft Visual Studio, open the ProtoTest.Nightshade Visual Studio solution file.

Nightshade Test Flow Instructions

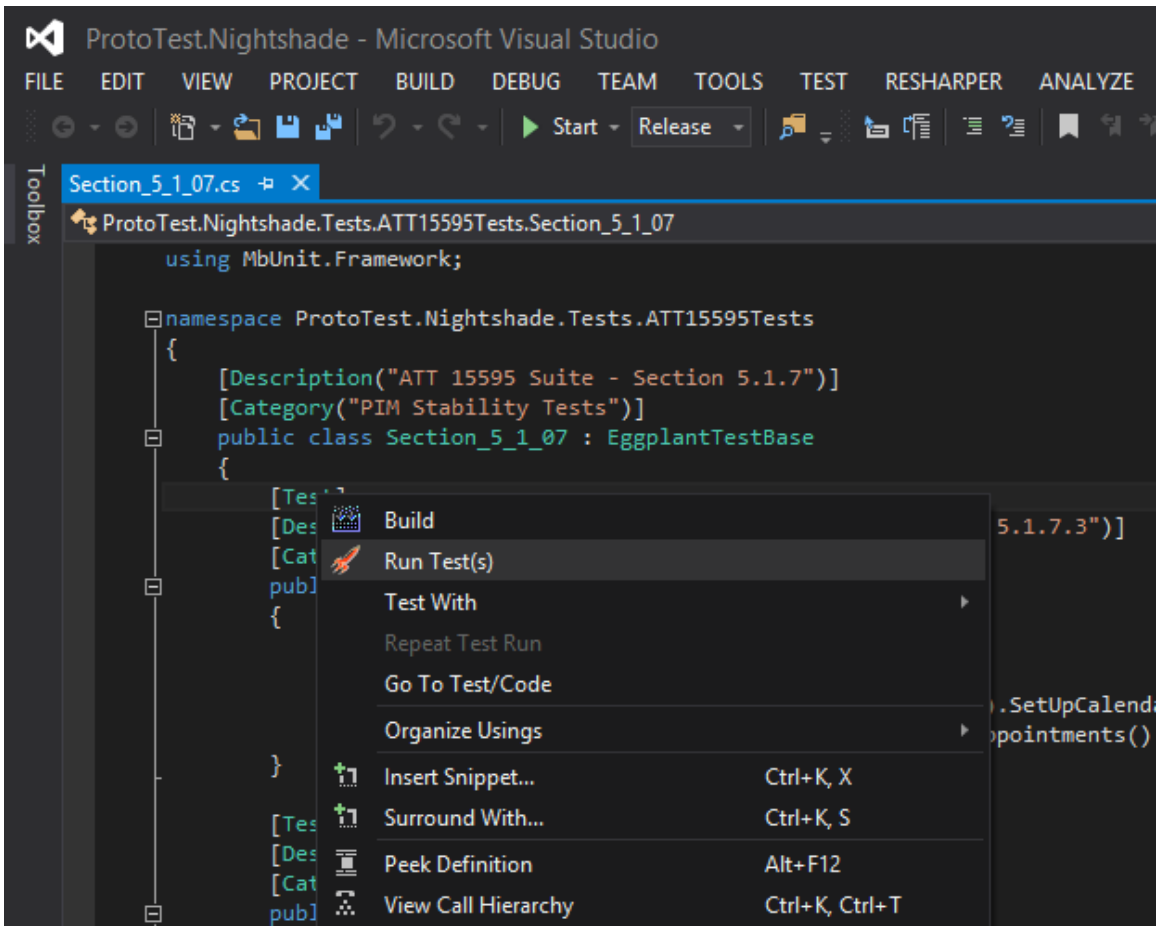
1. Consult the below diagram for Test code workflow.



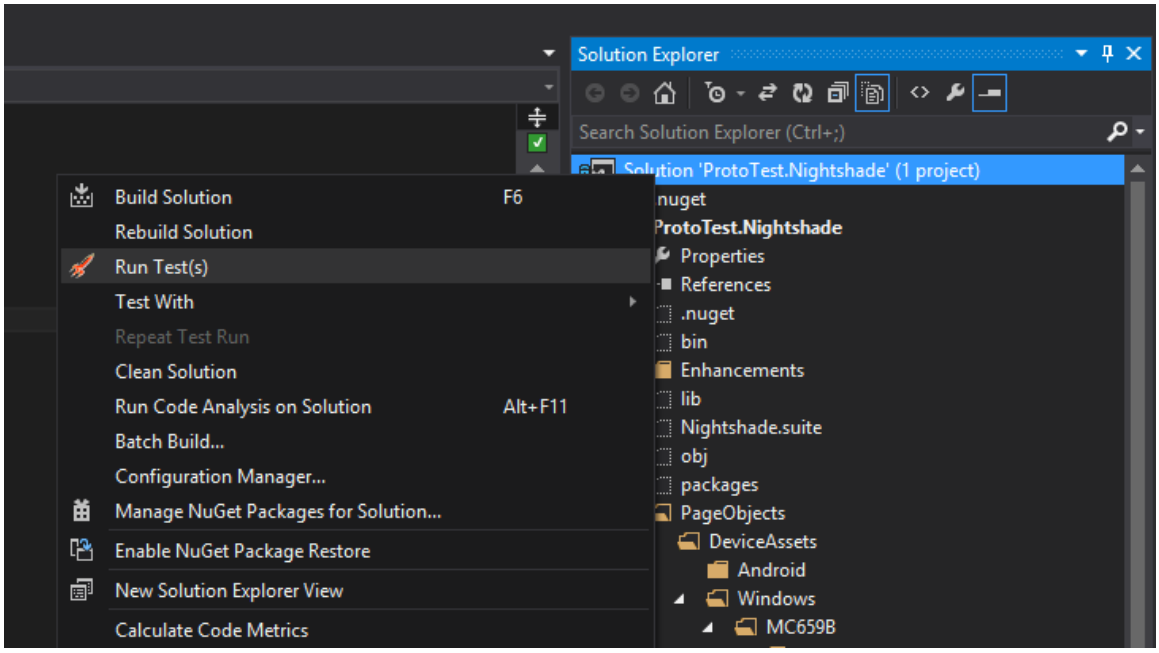
2. Set config values:

```
<configuration>
  <appSettings>
    <add key="DeviceType" value="Windows_MC659B" />
    <add key="Host1" value="169.254.2.1" />    <!--Device Under Test-->
    <add key="Host2" value="10.10.1.165" />    <!--Control Device for paired tests-->
    <add key="DriveTimeoutSec" value="60" />
    <add key="TestTimeoutMin" value="Null" />
    <add key="DeviceResolution" value="1024x768" />
    <add key="RecordVideo" value="False"/>
    <add key="LogDriveCommands" value="False"/>
    <add key="SuitePath" value="C:\Users\Roland Burrows\Documents\GitHub\ProtoTest.Nightshade\Suites\MotorolaMaster.suite"/>
    <add key="CalendarAppointmentsIterations" value="5"/>
  </appSettings>
</configuration>
```

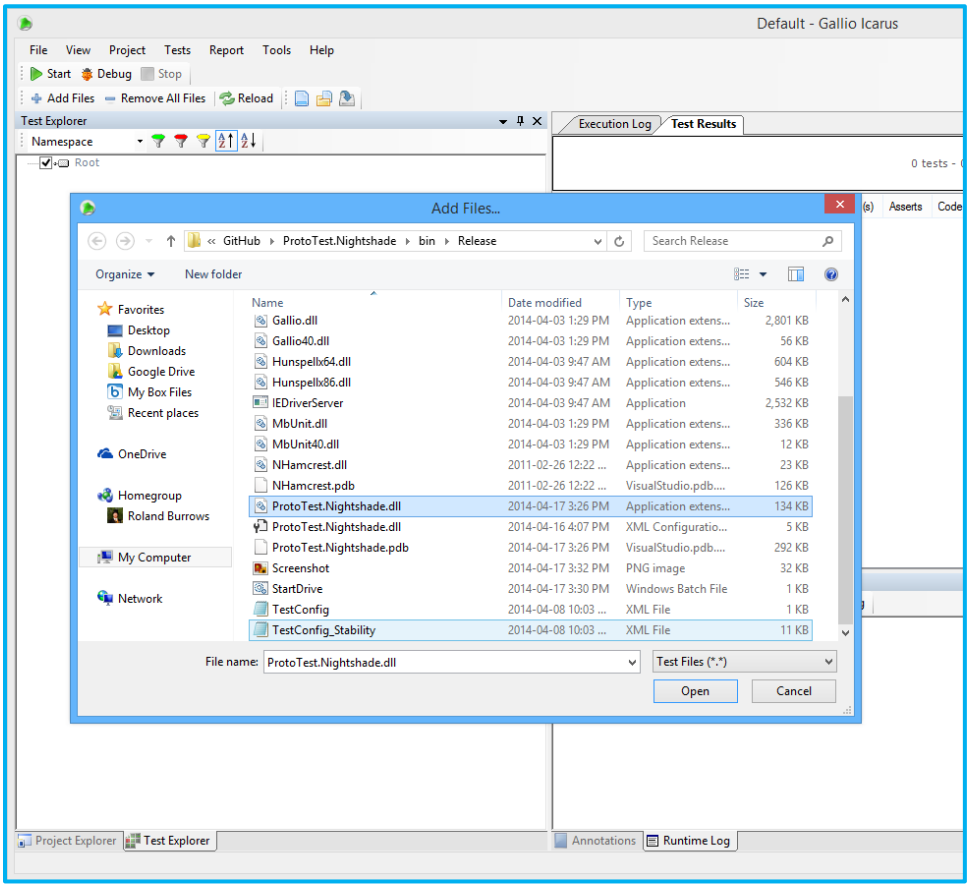
3. To run one test from Visual Studio:



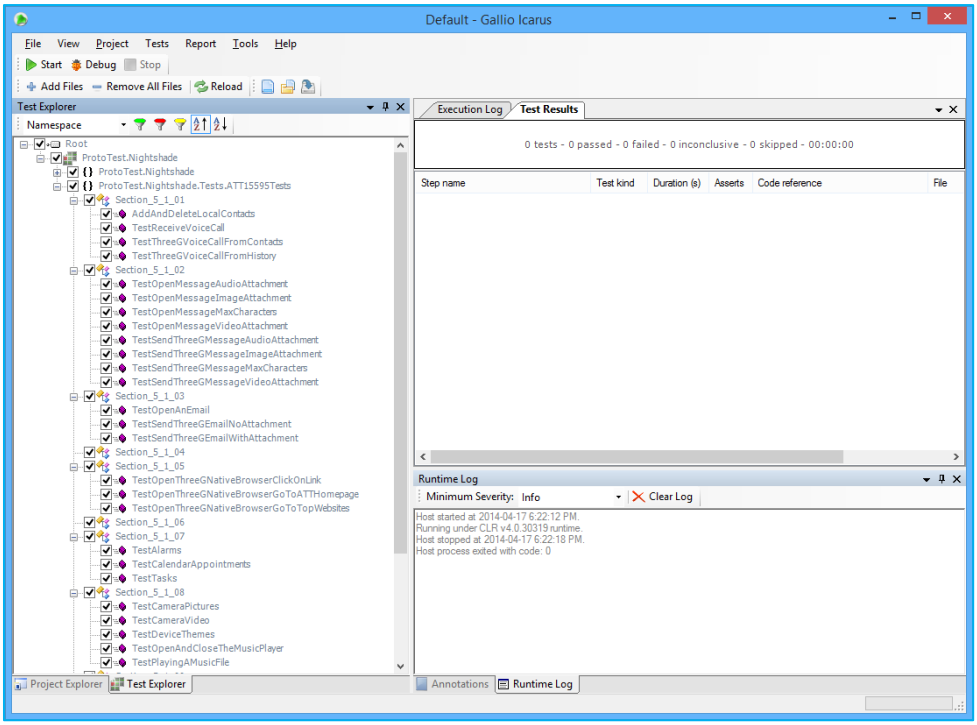
4. To run all tests from Visual Studio:



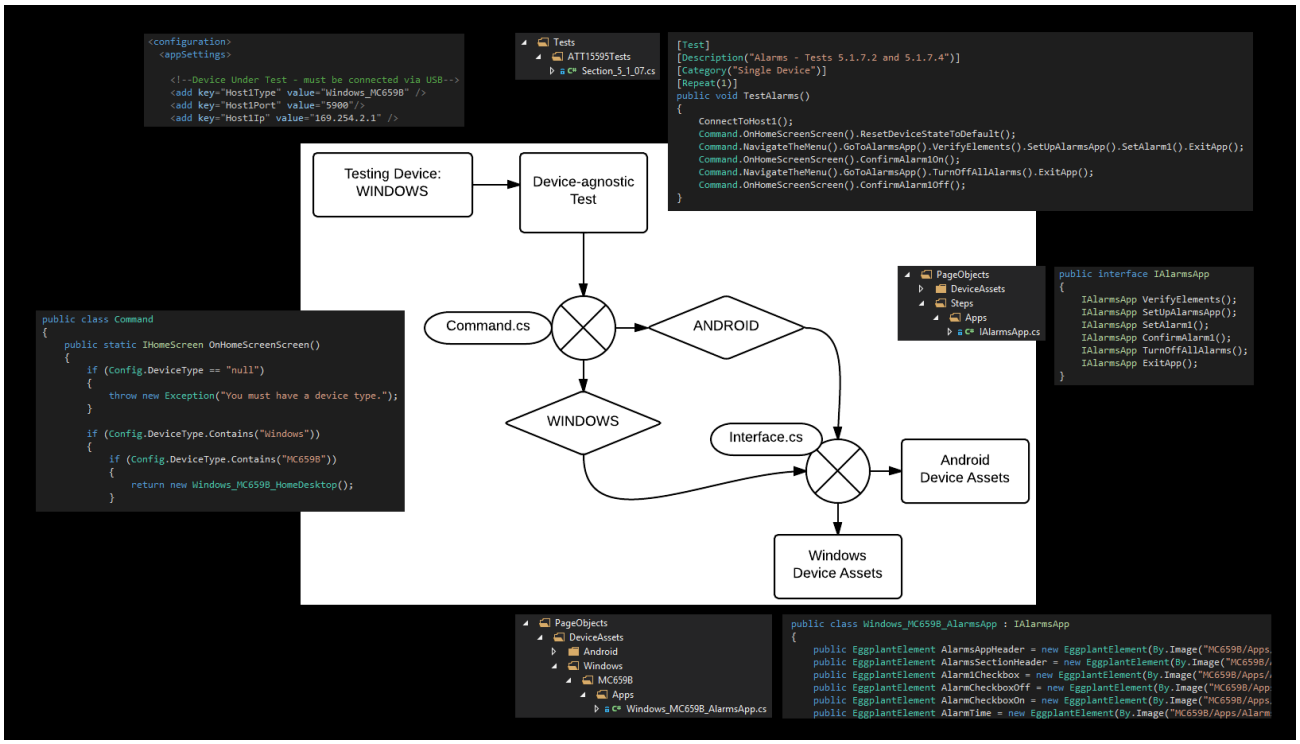
5. To run tests from Gallio Icarus:
    a. Load the .dll file from the Tests' bin\release folder (e.g. MotorolaTests.dll).



    b. Select the tests to execute and press the "Start" button. Don't forget to set the app.config file!



Nightshade Test Tech Stack



Framework Level	Purpose	Example File
Visual Studio – app.config	Global Settings	App.config
Visual Studio – [Test]	Test Steps	Section_5_1_01.cs
Visual Studio – command.[]	Set Device Type	Command.cs
Visual Studio – Interface	Steps Routing	IHomeScreen.cs
Visual Studio – Page Object	Device Elements	Windows_MC65_HomeDesktop.cs
Visual Studio – EggplantElement	Element Actions	EggplantElement.cs (Click, Type, Verify)
Visual Studio – EggplantTestBase	Code Actions	EggplantTestBase.cs (Connect, Log)
Visual Studio – EggplantDriver	C# Commands	EggplantDriver.cs
XML RPC Struct (by Cook Computing)	Translates C# to Sensetalk	EggplantDriver.cs XmlRpcStruct[] -> Third Party Library
Eggplant Drive	Runs Eggplant Tool	Documentation Link

Eggplant Device Connection Instructions (MC65)

Device Under Test

Since one of the ATT&T 15595 tests calls for disconnecting the Wifi radio and then reconnecting to a wireless network, the mobile device under test must be connected to Eggplant via USB:

- VNC app settings
  - Listening Port: 5900 + Disable "encrypt connection" + Authentication "none"
- VNC app
  - Stop all connections until "Ready" state is achieved.
  - With device plugged into the computer via USB, select Menu -> Listen.
  - Device should now be connectable to Eggplant through USB. If you specify Wifi or USB connections, the connection will reset upon test completion with the Nightshade framework (because of the way we're powering up and down Eggdrive).
  - Setting it to general "Listen" mode resolves this issue.
- In Eggplant, set the connection using IP 169.254.2.1 Port 5900.
- If your local IP for the USB connection is different, we can provide instructions on how to find it.
- Device should now be able to connect to Eggplant.

NOTE: The above steps where from a combination of our own troubleshooting and diagnosis with Testplant directly. If these steps do not allow your device to connect to Eggplant via USB, please contact Testplant support.

Partnered Control Device

- VNC app settings same as above
- VNC app
  - Stop all connections until "Ready" state is achieved.
  - With device connected to Wifi only (not USB), select Menu -> Listen.
  - Device should now be connectable to Eggplant through Wifi.
- In Eggplant, set the connection using the IP address and port of the device.
- Device should now be able to connect to Eggplant.

NOTE: If the device cannot connect to Eggplant directly, the Nightshade framework will not be able to run tests against the device. Please contact Testplant support for help.

Helpful Eggplant Documentation

Main Eggplant Documentation Portal

Using Eggplant Drive

Example Java Script

List of Eggplant Functional Commands

List of Eggplant TypeText Keywords

List of Eggplant Global Property Lists

List of Eggplant Runtime Options

About ProtoTest

Since 1998, ProtoTest has helped clients test software to ensure a positive user experience. Our Denver mobile app test lab is equipped with a collection of virtual and physical smartphones and tablets, including popular models from Apple, Samsung, HTC, LG, Motorola, Amazon, and more. The ProtoTest lab is staffed with experts skilled in functional and performance test automation with leading commercial and open source tools. Our staffing practice draws from a diverse talent community to fill contract, contract-to-hire, and direct-hire openings for testing professionals and other roles in the software lifecycle. ProtoTest's clients include industry leaders such as DigitalGlobe, Pearson, Covidien, and MyForce. For more information, please visit ProtoTest.com.