GMU-Team 1 : Documentation Report

IDT Spring 2014

Brannon Crymes - Akshay Karthik – Sriram Rajaraman

2014

Contact Details

Brannon Crymes - bwinter2@gmu.edu

Akshay Karthik - akshay.karthik@gmail.com

Sriram Rajaraman - s.rajaaraman89@gmail.com

Contents

[3 Introduction 2](#_Toc377500112)

[3.1 Running the System 2](#_Toc377500113)

[3.2 Building the System 3](#_Toc377500114)

[4 Usage 4](#_Toc377500115)

[4.1 Configuration 4](#_Toc377500116)

[4.2 Standard Testing API 4](#_Toc377500117)

[4.3 Report Writers 6](#_Toc377500118)

[4.4 Testing Race Conditions 6](#_Toc377500119)

[4.5 Log Viewer 6](#_Toc377500120)

[5 Architecture 7](#_Toc377500121)

[5.1 GMUT 7](#_Toc377500122)

[5.2 Test Builder 7](#_Toc377500123)

[5.3 Predicates 7](#_Toc377500124)

[5.4 Report Writer 7](#_Toc377500125)

[5.5 Log Viewer 7](#_Toc377500126)

[6 Requirements Traceability 8](#_Toc377500127)

# Introduction

We were asked to implement a system that allows developers to place checks within their code that can record the results of their code and asses the logic of the system. We call our system GMUT or GMU Tester. When we architected the system, we focused on simplicity of use, extensibility, and performance. We used plain java with no additional libraries to minimize the footprint of our system. The system is well documented and tested and fails gracefully if the application crashes. It does not leak memory and its thread safe API makes it easy to use in a variety of applications.

## Running the System

The logging framework can be run directly through the com.idt.contest.college.winter2014.Main interface. It works in both batch script mode as well as menu driven mode. The system returns results in a log.txt file that follows a specific format. This format was chosen to be able to search through via regular expressions and is documented in documentation/report\_format.txt. This format is used in logviewer as well as all of the included ReportWriters however, the extensible nature of the system allows for custom ReportWriters in any format that a developer wishes to use. There are three key access points to the system:

1. gmu\_main.jar is the com.idt.contest.college.winter2014 jar with the functions annotated by the GMUT testing API. It can be run without arguments to access the menu driven application. If an argument is provided, it runs using a batch script.   
   java –jar gmu\_main.jar  
   java –jar gmu\_main.jar batchscript.txt
2. gmu\_logviewer.jar is the application to view the logs.  
   java –jar gmu\_logviewer.jar
3. gmu\_tests.jar is the unit tests for the GMUT API. It is a simple console application that runs all of the unit tests that verify that the system meets the given requirements.  
   java –jar gmu\_tests.java

## Building the System

Building the system is simple, it can be built from eclipse.

1. Import project into eclipse (file > import existing project)
2. Run Configurations
   1. LogViewer.java for log viewer
   2. Main.java for the IDT application
   3. InternalTestRunner.java for the internal tests application

Also provided is an Ant build.xml file for Apache Ant build system. This system automatically builds all three jars as well as places the default batchfile.txt file in the main directory. It also builds the javadocs for the project and places it in the documentation directory.

To run the ant build file, simply run the following console command. This assumes that java as well and ant is available on the console’s PATH.

ant -buildfile build.xml

The ant file performs five actions:

1. Build /src into gmu\_main.jar with com.idt.contest.college.winter2014.Main as the main class.
2. Build /src into gmu\_logviewer.jar with edu.gmu.team1.idt2014.logviewer.LogViewer as the main class
3. Build /test into gmu\_tests.jar with edu.gmu.team1.idt2014.test.InternalTestRunner as the main class
4. Copy batchscript.txt into the main directory.
5. Generates javadocs and places them into the documentation/javadoc folder.

# Usage

Before working with the GMUT system, it is essential to understand what a test is in this framework. This framework is developed in terms of predicates. In math, a predicate is a function that takes a set of values, and returns a Boolean value. An example of a predicate is equality (if x = 5, return true, else false). In this library, a test is defined as a mapping between an input predicate (one that evaluates the input) and an output predicate (one that evaluates the output). For example, let us take the example function isGreaterThanTwo. We can explain each test as a predicate.

if input is 0 output should be false - ( input == 0 ) -> ( output == false )  
if input is 3 output should be true - ( input == 3 ) -> ( output == true )

Similarly, any test in this library is a mapping of a predicate to another predicate. There are many predicates provided and many more can be used by subclassing the Predicate class.

## Configuration

By default, the system starts out enabled with the reporting system configured to output to a file called log.txt. It is very simple to enable or disable the API. Simply call GMUT.disable() to disable the testing system. The output location can be configured by using GMUT.setReportWriter where you may pass in a new ReportWriter. Further details on ReportWriter usage is in section 4.3.

## Standard Testing API

When looking at the problem statement, we split our solution into two key sections. Declaring the test cases, and actually performing the tests. These requirements, along with emphasis on API simplicity lead to the usage pattern described here. The standard testing API has a single point of access. “edu.gmu.team1.idt2014.GMUT”. GMUT is a thread safe singleton and should be accessed statically. GMUT has two key functions, addTest and test. addTestis the test creation interface and testperforms the actual testing. To best understand how the system works, let us first take a look at the test function (the one that actually logs the test results). IIt is necessary to get under the hood of the test method and see the underlying logic.

1. Check if testing is enabled, if not exit test method.
2. Check if test cases exist for a given method. If they don’t, exit test method.
3. Get the test cases (list of predicate to predicate mappings)
4. For each test in the test cases
   1. Evaluate the first predicate with the input of the function (e.g if input = 10)
      1. Evaluate the output with the second predicate (output = true)
      2. Log the value of the second predicate (true = passed test, false = failed test)

Let us take a look at the example function (isEven) which takes an integer and returns true if the number is even, false if the number is odd).



Here, the creation of the test cases are done in lines 2-6, the actual testing is done in lines 8 and 11. One could read line 4 as if saying, if the input is 10 (first predicate is the input predicate), the output should be true (the second evaluates against the output). Finally in lines 8 and 11, we say, run the test (with the arguments (output, branch, input)).

GMUT.addTest returns an instance of ITestBuilder. ITestBuilder is a system that follows the factory pattern and sequentially builds up a test case until line 6 where it builds the tests and inserts it into the main GMUT system. There are two variants of the test creation method. .test expects two predicates in the order of input and output. .testNote does the same except it has the added functionality of marking a note next to each test.

Actually running the tests is as simple as calling GMUT.test with the arguments in the order, output, branch, inputs. The test method supports variable length arguments so one may simply list all of the inputs to the method being tested. Thus, usage of the API is very simple, in the beginning of the function, create the test cases

## Report Writers

ReportWriters are pluggable classes that can be set in GMUT that allow for results of tests to be logged. The Library comes with three ReportWriters already implemented. These are FileReportWriter, ConsoleReportWriter, and StringReportWriter. By default these reportwriters follow the report format expected by logviewer however it is trivial for a developer to subclass AbstractReportWriter and have it write the files in a new format. FileReportWriter is the default ReportWriter for and has a variable FILENAME which is (by default) set to “log.txt”. By changing this variable, a developer may easily change the location of the file. As expected ConsoleReportWriter writes the results of the tests to System.out. StringReportWriter logs the results of tests to a public variable report.

The report format is detailed in documentation/report\_format.txt however, a quick description of the file format follows:

[<DATE>][<TIME>][<PASS/FAIL>][c:<CLASS>][m:<METHOD>][b:<BRANCHES>][i:<INPUTS>][o:<OUTPUT>][n:<NOTES>]

The above is a description of a single line, each line ends with windows style line endings “\r\n”. Multiline objects replace “\r\n” with “{r}{n}”. A line starting with tilde (~) is a comment line and is not parsed for ReportWriter. A line starting with tilde + exclamation mark (~!) is for custom testers (especially ThreadTester).

## Testing Race Conditions

## Log Viewer

# Architecture

## GMUT

## Test Builder

## Predicates

## Report Writer

## Log Viewer

# Requirements Traceability