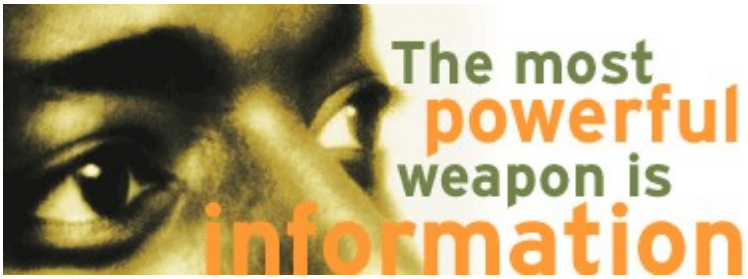
Martus Testing Project

Chapter 3
Deliverable Expectation Document



October 29, 2013

Global Human Rights Abuse Reporting System

3

3.1 Purpose

This chapter of the deliverable is meant to give the stock holders an inside look at how the software is progressing. This will describe the structure of the framework, the changes made this far, the how-to document, and a gage at how far along it is.

3.2 References

Main reference used (Martus web page) (https://www.martus.org/).

3.3 Experiences

Number	Summary	Date	Affected
1	Martus project files uploaded into repository	10/10/2013	All
2	Testing script uploaded	10/14/2013	Scripts folder
3	First 3 test cases added	10/16/2013	Folders affected include Test Case text files, Oracles, and TestCase Executalbes
4	Script moved to top level directory	10/16/2013	runAllTests.sh, all TestCase text files
5	6 test cases added	10/22/2014	Folders affected include Test Case text files, Oracles, and TestCase Executalbes

The team had met once a week up until our presentation of deliverable 2. Every since then, we've been meeting every Monday and Friday for a few hours being sure all members of the group are up to date and understand the changes that have been made to the repository. There was a bit of confusion as to how the framework was to interact with the project files and what files were to be generated by tests. Once we arrived a reasonable agreement, the project and script were uploaded and all members of the group were uploading test cases remotely. Our remaining meetings were then about updates made in the cloud.

3.3 Test Cases

Test Case	Requirement	Method Tested	Input Used	Output Expected
#1	Application must be able to encrypt data	common.crypto.MartusSecurity.encrypt()	A string to encrypt	String encrypted = true
#2	Application must be able to decrypt encrypted data	common.crypto.MartusSecurity.decrypt()	./temp/encryptedBytestring.txt	Decrypted bytestring = A string to encrypt
#3	Username must not match password	client.core.MartusUserNameAndPassword.validateUsernameAndPassword()	--username=Password --password=Password	org.martus.common.Exceptions$PasswordMatchedUserNameException
#4	Password must be at least 8 characters	client.core.MartusUserNameAndPassword.validateUsernameAndPassword()	--username=Username --password=1234567	org.martus.common.Exceptions$PasswordTooShortException
#5	Username must not be blank	client.core.MartusUserNameAndPassword.validateUsernameAndPassword()	--username= --password=Password	org.martus.common.Exceptions$BlankUserNameException
#6	Testing if setting a database key to draft works after its been initialized to "sealed" by default	common.database.DatabaseKey.isDraft()	true	true
#7	Testing if database keys are set to "sealed" by default	common.database.DatabaseKey.isSealed()	true	true
#8	Testing for difference in keys set to sealed vs keys set to drafted	common.database.DatabaseKey.isSealed()	true	false
#9	Application must be able to store user data	common.database.FileDatabase.createRecord()	--entries-to-create=5	Database record count = 5
#10	Application must be able to delete stored user data	common.database.FileDatabase.discardRecord()	./temp/databaseKeys.txt	Database record count = 0
#11	Application will try allowed secure ports until a connection is established	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()	--good-port-middle=7	Number of tried ports = 3
#12	Application will try only enough secure ports to establish a connection	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()	--good-port-first=7	Number of tried ports = 1
#13	Application will try all available ports to establish a connection	clientside.ClientSideNetworkHandlerUsingXmlRpc.callServer()	--fail-all=true --good-port-first=7	Number of tried ports = 5
#14	Application must create a universal ID from account and local IDs	common.packet.UniversalId.createFromAccountAndLocalId()	--account-id=someAccountID --local-id=someLocalID --prefix= --from-string=false	Account ID = someAccountID & Local ID = someLocalID
#15	Application must create a universal ID from account ID and a prefix	common.packet.UniversalId.createFromAccountAndPrefix()	--account-id=someAccountID --local-id=someLocalID --prefix=D- --from-string=false	Account ID = someAccountID & Local ID length = 26
#16	Application must create a universal ID from the string representation of account and local IDs	common.packet.UniversalId.createFromString()	--account-id=someAccountID --local-id=someLocalID --prefix= --from-string=true	Account ID = someAccountID & Local ID = someLocalID

3.4 Framework Descriptions

The framework is an extensible testing framework made for Java and run on Unix systems. The framework makes use of test case files to define parameters to run java test classes. Test case files are named in the format: 'testCaseX.txt' where X is the test number. The framework operates by looping through all valid test case files located in /top_level_directory/testCases, running the java test classes defined by each test case, creating the oracle and recording the

results. Evaluation of test results are made by comparing actual output of the test class to the expected output - any mismatch results in test failure. Results of each test run are generated as testReport_mm-dd-yyyy_hh:mm:ss.html and are stored in /top_level_directory/reports. Each report gives detailed testing parameters for each test case, and, in the event of failure, gives reasons for test failures. The framework may be extended by defining your own test cases that reference your java test cases.

3.4.1 Framework Directory Structure

/team4

- runAllTests.sh
- /project
  - /martus-amplifier
  - /martus-client
  - /martus-cleintside
  - /martus-common
  - /martus-hrdag
  - /martus-jar-verifier
  - /martus-js-xml-generator
  - /martus-logi
  - /martus-meta
  - /martus-mspa
  - /martus-server
  - /martus-swing
  - /martus-thirdparty
  - /martus-utils
- /scripts
  - runTest.bash
- /testCases
  - testCase1.txt
  - testCase2.txt
  - testCase3.txt
  - testCase4.txt
  - testCase5.txt
  - testCase6.txt
  - testCase7.txt
  - testCase8.txt
  - testCase9.txt
  - testCase10.txt
  - testCase11.txt
  - testCase12.txt
  - testCase13.txt
  - testCase14.txt
  - testCase15.txt
  - testCase16.txt
- /testCasesExecutables
  - testDatabaseKeyDrafted.java
  - testDatabaseKeySealed.java
  - testDatabaseKeyStatuses.java
  - testDatabaseRecordCreation.java

testDatabaseRecordDelete.java
testDecrytp.java
testEncrypt.java
testEncryptDecrypt.java
testSSLPortSelect.java
testUniversalID.java
testUsernamePassword.java
/temp
/oracles
/docs
README.txt
team4_deliverable1.pdf
team4_deliverable2.pdf
/reports
/img
fail.png
pass.png
report.css

3.5 How-To

3.5.1 Overview

The testing framework is packaged with .java test files for the open source project Martus (www.Martus.org), but can easily be adapted to test any Java project. The framework uses text files to define test cases to run, and creates a detailed report for each test run.

3.5.2 Requirements

The framework should run in any Unix environment and the Java version compatibility is entirely dependant on the methods used in your classes. However, testing of the framework with the pre-packaged class files has been limited to Ubuntu 12.04/Java 1.6.

3.5.3 How to use

Using terminal, navigate to the top level folder (/team4 by default) and type 'sh runAllTests.sh'. This will execute all available test cases, and, upon completion, will open the report file created in '/team4/reports'.

3.5.4 Creating your own Test Cases

The framework is made extensible through the use of test case plain text files. Once you have written a test in Java it is a simple matter of creating a test case with the necessary parameters to add the new test to the suite. Test case files must be named using the convention 'testCaseX.txt' where X is any number. The following parameters are contained in a test case file:

- testNumber= (required) Must be unique, conventionally the same number used in the file name.
- comments= (optional) Comments displayed during test compilation/running.
- requirement= (optional) The requirement being tested, used in the report file.
- methodTested= (optional) The method being tested, used in the report file.
- testDriverPath= (required) The path to the .java file that runs the test.
- testDriver= (required) The name of the .java file that runs the test.
- compileClassPath= (required) The classpath entries needed to compile. Exactly as they would be typed

in the terminal. e.g. '-cp /cp/entry/one:/cp/entry/two' see Java documentation for further help in using the classpath flag.
runClassPath= (required) The classpath entries needed to run. Entered as above.
input= (optional) The input, if any, used by your Java test class. Input is passed to the class exactly as typed. The only exception being that input will accept the relative path to a .txt file as input, and retrieve literal input from the file. Any spaces in the input parameter will be interpreted by your Java main class as delimiting String array elements. String building must be handled in your Java class.
expectedOutput= (required) The expected output of your Java class. Determines test pass or fail status.