OMNIEDITOR PROJECT

PHASE B: TEST PLAN

CODENAME: NIRVANA

CSC408F1
Group Members:

Dylan Durdle
Yani Ioannou
Ashish Kaila
Sandy Lam
Gilles Lising
Mark Maglaque

Table of Contents

1. INTRODUCTION 3

1.1 OBJECTIVES 3
1.2 LOGGER 3
1.3 REFLECTION 3

2. TESTING STAGES AND STRATEGIES 5

2.1 UNIT TESTING 5
2.2 INTEGRATION TESTING 5
2.3 USER ACCEPTANCE TESTING 6
JUNIT TEST 6
DETAILS OF OUR UNIT TEST STRATEGY 7
TIPS TO UNIT TESTING 8
2.4 WHITE-BOX AND BLACK-BOX TESTING 8

3. TEST CASE DESCRIPTIONS 10

3.1 LOADDOCUMENT() 10
Requirements to be Tested 11
3.2 CLOSEDOCUMENT() 11
Requirements to be Tested 12
3.3 SAVEDDOCUMENT() 12
Requirements to be Tested 13
3.4 GETDOCUMENT() 13
Requirements to be Tested 14
3.5 BROWSE() 14
Requirements to be Tested 15
3.6 SAMPLE LOCK, UNLOCK, SAVE ETC 15
3.7 DELETE() 16
3.8 DELETELINE() 17
3.9 GETLINE() 21
3.10 GETLOCKS() AND GETLOCKSANDUIDS() 27
3.11 GETSEQUENCENO() 32
3.12 INSERT() 33
3.13 LOCKLINE() 33
3.14 ADDLINE() 37
3.15 UPDATELINE() 42
3.16 UNLOCK() 43

4. APPENDICES 48

4.1 APPENDIX A – SHARED COMPONENTS 48
4.2.1 Reflection API 48
Provides reflection based support to objects. 48
4.2.1.1 ObjectNameSpace 48
public static String ObjectNamespace(Object Object) 48
4.2.1.2 ObjectQualifiedTypeName 48
public static String ObjectQualifiedTypeName (Object Object) 48
4.2.1.3 ObjectShortTypeName 48
public static String ObjectShortTypeName (Object Object) 49
4.2.1.4 MethodStackTrace 49
public static StackTraceElement[] MethodStackTrace() 49

4.2.2 Logging API 49
Logs program status to a log file. 49
4.2.2.1 SetLevel 49
4.2.2.2 LogMessage 49
4.2.2.3 LogEntrance 50
public void LogEntrance() throws ServiceException 50
public void LogEntrance(Object[] Params) throws ServiceException 50
4.2.2.4 LogExit 50
public void LogExit(Object[] Params) throws ServiceException 51
4.2 APPENDIX B - TEST FORM CHECKLIST 51
4.3 APPENDIX C - BUG TRACKING FORM 52
4.4 APPENDIX D – TEST SCHEDULES 52

1. Introduction

Every user of a software application needs to be provided with a high quality product. As a result, this test plan has been developed in order to ensure that the OmniEditor project meets high quality standards.

Included in the following sections are descriptions of the forms of testing used and a set of test cases illustrate our testing procedures of the software's functions. Section 1.2, 1.3, as well as the APIs included in Appendix A describe the components that our team will be taking advantage of as part of our testing and debugging tools, namely bug logger and reflection. Lastly, the amount of time allocated for the testing stages is detailed on a Gannt Chart presented at the end of this document.

1.1 Objectives

- Eliminate all critical and major bugs and as many minor bugs as possible before delivery
- Identify and cover all different types of input categories
- Ensure that the requirements are met

1.2 Logger

A simple shared component, called logger, has been specifically designed to help us effectively troubleshoot and find the location of any occurrences of a program crash. Logger is called at the beginning and end of all the functions and it keeps a record of the names of the functions that called it and performs proper indentations if multiple sub-function calls are made. Consequently, a program crash would result in a missing indentation and will thus make it easier for the programmers to locate the source of the error.

1.3 Reflection

The reflection shared component is a particularly useful component that will be helpful during the debugging of our software. The various functions provided can be used to conveniently retrieve information of

where certain objects are defined and referenced. Object information such as object qualified type name and namespaces would be readily available.

2. Testing Stages and Strategies

2.1 Unit Testing

Team members with a specific programming task will develop work that will later be part of the working software system. As per the definition of "a unit", a member's work is a piece of code and will be referred to in this document as a unit. To reduce the risk of contaminating another member's working code, which could introduce new bugs into the software making bug finding more expensive and difficult, each member is responsible for the first testing stage to be done. Members will initiate unit tests with adequate test cases on their piece of code that may incorporate the use of test drivers and stubs. The code should be proven to work independently from other components before it is accepted and integrated. Precautions like these are necessary to maintain stable versions of the working software.

2.2 Integration Testing

Code from a team member that has been proven to work independently by way of unit test results will need to be reviewed and then eventually accepted by other members to be integrated with the rest of the working software system. At this point some stubs for a unit that were initially used to prototype various methods or external components will need to be filled in and will thus require the need for integration testing. This type of testing will take a top-down approach since not all of the other members will need to review a certain member's work in detail, due to the fact that it has already been proven to work. Bugs found, documented and fixed at this stage are usually expected to result from integration errors. This process can happen every time new units or components are introduced to extend the software from its previous state.

2.3 User Acceptance Testing

Once the integration stages are complete and the integration testing results are recorded and reviewed, the software system is presented to prospective users. At this stage, the user is critical part of the development process. For the OmniEditor project, other teams define the users of our team's software system. Bugs found by the users at this stage are documented and fixed. User suggestions are also considered. After this stage is complete and depending on the program's handling of the requirements, the clients (other such users) can regard the program as satisfactory if requirements are fulfilled. The program is regarded as unsatisfactory if it did not adequately fulfill the requirements or because of the accumulation of any outstanding bugs. The team's capability to understand requirements and to follow-through with the design and implementation plays an important role.

All the methods were tested one by one using Bottom-Up integration testing. That is, all the leaves methods were thoroughly tested first before their parent was actually tested. For instance, according to the *structure of the server*, *lockLine* and *updateLine* were tested before *insert*.

While testing each module, if a bug was found the tester would immediately report the bug through *Bugzilla.* Then, the error report would be sent to the developer concerned. Once, the developer receives the email he would immediately check out the bug and fixed it, provided it is a valid error. Also, if several bugs have been found, the developer would proceed to debugging according to the severity and priority of the bug.

JUnit Test

We used JUnit to perform the unit testing. Each function of the system has a TestCase composed of one or more test methods.

Some sites that were useful for our testing were:

http://junit.sourceforge.net/javadoc/junit/framework/Assert.html

http://junit.sourceforge.net/doc/faq/faq.htm#tests_16

Details of our Unit Test Strategy

Some general tasks we tried to accomplish with unit testing are:

- Testing also involves performing actions and verifying these actions have really occured. This involves checking the return value *and* periodically checking the data on the system. For example, taking Sandy's test case as an example, she does some line deletes and then does a query on saveDocument() to ensure that the document in memory really does lack the lines she just deleted in her test case.

- We have to test all possible error conditions associated with the method. What happens if I delete line = -1? etc. You can figure out what methods throw what errors from the method documentation. Make sure you always do UPDATES in CVS to ensure you have the latest files.
- The naming conventions of all test files are *TestCase*. It has to have the word TestCase instead of Test or Cases or whatever else people are using.
- The more test cases --> less errors --> less bugs in phase C --> the higher the mark.
- Test cases should generate random documentHashes. Look at TestCase1.
- Also, for methods that update the system, it might be nice to test the following (i.e. it must be done):
  - lock the line and then perform my operation (update/insert/add)
  - don't lock anything and perform my operation
  - lock and unlock and then perform my operation
  - have another user lock the line and then perform my operation (should error out)
- BUGS ARE YOUR FRIENDS. If we get failures *and* it is failing for reasons that defy how the method works, then we open a bug. This will allow developers to know of the problem and look into it. We get much more done this way then just ignoring the problem until the developer come across running the test case.
- The interface to the methods is now frozen. It will not change for non-crucial feature changes. This helps with testing.

For a sample test class shows simple line lock testing. We try to answer/test the
following:

Can I lock a line?
What happens if I relock a line?
Can someone else lock the line?
Can someone else lock the line before the timeout?
Can someone else unlock my line?
Can I lock multiple lines?
Can I unlock lines - and get the correct lines unlocked?

It also runs a test against saveDocument(), asking the question: Can I retrieve the document as uploaded?

There are some new methods available- nothing of real interest except for closeDocument() that will close a given document. Handy for testing, but don't abuse it!

Tips to Unit Testing

Some tips for unit testing:

1) text with weird characters include '(the ' can trip up a database fairly easy - I think I've filtered it correctly)
2) permutations of add/update/delete lines - I'll be adding some method shortly that when run can verify the integrity of open documents
3) any method that uses getLineID/getLineNo - because the line number eclipse communicates with OmniWebServer can change often, this method attempts to correctly offset the line number with the updated line number or the true value of the lineid. There are a lot of cases to handle, so this is definitely one area where bugs lurk.
4) boundary cases
5) stress test - to ensure proper concurrency practices have been followed.

2.4 White-box and Black-box Testing

Black-box testing allows the testing of software usually in accordance to a guide, which could be API documentation or requirements

specification. Such information should be already available in order to explain how to use the piece of software or to develop test cases. The internal workings of the code are not known by the tester when testing. The testers employing this method need not be the original authors of the code being tested.

White-box testing follows a similar technique used in black-box testing but allows the tester to test the internal workings of a piece of software with specific test cases and test data that would not usually arise in the black-box approach. The testers are usually the original authors of the code being tested.

At the unit testing stage, the white-box approach will most likely be used since the implementer would have internal knowledge of the code being tested. For the integration and user acceptance testing stages, the black-box approach will be used since other members or users involved in these stages would most likely approach the testing in a top-down fashion making sure that the software behaves as expected. These two forms of testing are methods that will be fully utilized at each testing stage to ensure the production of robust and reliable code.

3. Test Case Descriptions

3.1 loadDocument()

String loadDocument(String documentName, String document)

Purpose: Loads the document in the OmniEditor Web-Service from a client.

Parameters:
- documentName – Name of the document.
- document – Contents of the document.

Return Value: Returns a document hash representing the document in the web-service, or null if the operation failed.

Test case #: 1	
Input:	(null, null)
Ouput:	Null
Test:	Successful

Test case #: 2	
Input:	(<repeated doc name>, <repeated contents>)
Ouput:	Null
Test:	Successful

Test case #: 3	
Input:	(<unique doc name>, <unique contents>)
Ouput:	Document hash string
Test:	Successful

Test case #: 4

Input:	(<unique doc name>, <4MB Contents>)
Ouput:	Document hash string
Test:	Successful

Test case #: 5	
Input:	(<unique doc name>, <Blank Contents>)
Ouput:	Document hash string
Test:	Successful

Requirements to be Tested

The LoadDocument function must be configured to upload text files over to the OmniServer web-service. File attachments up to 4MB (far exceeds project requirement for OmniEditor posted by Prof.) tested successfully along with boundary conditions.

3.2 closeDocument()

long closeDocument(String documentHash)

Purpose: Closes a document in the OmniEditor Web-Service from a client.

Parameters:

- documentHash – Hash Name of the document.

Return Value: Returns a long value corresponding to the operation sequence number or the error code.

Test case #: 1	
Input:	(null)
Ouput:	1
Test:	Successful

Test case #: 2	
Input:	(<non existing doc hash>)
Ouput:	1
Test:	Successful

Test case #: 3	
Input:	(<valid doc hash>)
Ouput:	1
Test:	Successful

Requirements to be Tested

The CloseDocument function must be configured to delete text files from the OmniServer web-service. In case a document hash does exist, function should just exit with an appropriate error code.

3.3 saveDdocument()

String saveDocument(String documentHash)

Purpose: Download/Save the document to the local client's buffer.

Parameters:

- documentHash – Hash Name of the document.

Return Value: Returns the name of the file along with the contents.

Test case #: 1	
Input:	(null)
Ouput:	Null
Test:	Successful

Test case #: 2	
Input:	(<non existing doc hash>)
Ouput:	Null
Test:	Successful

Test case #: 3	
Input:	(<valid doc hash>)
Ouput:	The file contents
Test:	Successful

Requirements to be Tested

The SaveDocument function must be configured to download text files from the OmniServer web-service. In case a document hash does exist, function should return null.

3.4 getDocument()

String getDocument(String document)

Purpose: To retrieve document hash given its contents.

Parameters:

- document – Contents of the document.

Return Value: Returns the document hash of the file.

Test case #: 1	
Input:	(null)
Ouput:	Null
Test:	Successful

Test case #: 2	
Input:	(<non existing doc hash>)
Ouput:	Null
Test:	Successful

Test case #: 3	
Input:	(<valid doc hash>)
Ouput:	The file hash name
Test:	Successful

Requirements to be Tested

The GetDocument function must return documents hash name from the OmniServer web-service if contents exist on the server. In case a document with the specified contents does exist, function should return null.

3.5 browse()

String browse()

Purpose: To retrieve document information from the OmniEditor Web-Service.

Return Value: A list of documents available (and open) on the server following the format
'document_hash','document_name','last_modified_timestamp'

Test case #: 1	
Input:	N/A
Ouput:	Names of all opened documents along with their details
Test:	Successful

Test case #: 2	
Procedure:	Loaded a document, browsed, then closed the same document and browsed.
Ouput:	In the first browse the document details was contained unlike the second browse.
Test:	Successful

Requirements to be Tested

The Browse function must return documents names along with the aforementioned details from the OmniServer web-service. The documents included in the output must be in open state. All closed documents must be deleted and hence should be excluded from the browse results.

3.6 Sample lock, unlock, save etc

Build document or fetch existing documentHash
Checks for correct:
document upload ability

Test the ability to retrieve the document.
Checks for correct:
document integrity
document filename

Test the lock and unlock
Checks for correct:
locking and unlocking line-by-line by 1 user
unlocking a line with wrong user (fail)
lock 1st line by user 1
look for lock
unlock 1st line by user 1
reunlock 1st line by user 1: should fail
look for lack of lock
lock 2nd line by user 1

unlock 2nd line by user 3: should fail
look for lock
unlock 2nd line by user 2

Test the lock and unlock
Checks for correct:
locking and unlocking line-by-line by 1 user
unlocking a line with wrong user (fail)

lock 1st line by user 1
look for lock
lock 1st line by user 2: should fail
look for lock
unlock 1st line by user 1
lock 1st line by user 2
look for lock
lock 1st line by user 1: should fail
look for lock
unlock 1st line by user 2

3.7 delete()

Test some trivial valid deletes
Delete "Line " in Line 1
Delete "8" in Line 8
Delete all "a"s from Line 6
Delete all "e"s from Line 6

Test some invalid deletes
Delete a non-existing line
Delete line -1
Delete out of boundary
Delete negatives positions
Delete Line 1 with a startPosition 0 and an endPosition that is out of boundary
Delete from a non-existing document
Check if filename contents matches the changes

Test delete by playing with lock and unlocking lines first
Lock and unlock first and delete "Line " from line 1
Lock first and delete "8" from line 8
Unlock first and delete "Line " from line 2

Test delete using 2 different ids to represent 2 clients
Lock and Unlock by user2 and delete "Line " from line 1 by user1
Lock by user2 and delete "8" from line 8 by user1 Build document or fetch existing documentHash
Checks for correct:
document upload ability

3.8 deleteLine()

Test case #: 1		
Function: long deleteLine(String documentHash, long line, long userID, long sequenceNo)		
Purpose: Boundary Test if the 1st line can be deleted		
Input:	documentHash: line: userid: sequenceNo:	From loadDocument() 1 123 0
Expected Output:	1	
Actual Output:	1	

Test case #: 2
Function: long deleteLine(String documentHash, long line, long userID, long sequenceNo)
Purpose: Boundary Test if a non-empty line can be deleted around the middle of the document

Test case #: 2		
Input:	documentHash: line: userid: sequenceNo:	From loadDocument() 6 123 1
Expected Output:	2	
Actual Output:	2	

Test case #: 3		
Function: long deleteLine(String documentHash, long line, long userID, long sequenceNo)		
Purpose: Boundary Test if a non-empty line can be deleted around the end of the document		
Input:	documentHash: line: userid: sequenceNo:	From loadDocument() 27 123 2
Expected Output:	3	
Actual Output:	3	

Test case #: 4		
Function: long deleteLine(String documentHash, long line, long userID, long sequenceNo)		
Purpose: Boundary Test if an empty line can be deleted around the middle of the document		
Input:	documentHash: line: userid: sequenceNo:	From loadDocument() 6 123 3

Test case #: 4	
Expected Output:	4
Actual Output:	4

Test case #: 5		
Function: long deleteLine(String documentHash, long line, long userID, long sequenceNo)		
Purpose: Boundary Test if an empty line can be deleted around the end of the document		
Input:	documentHash: line: userid: sequenceNo:	From loadDocument() 25 123 4
Expected Output:	5	
Actual Output:	5	

Test case #: 6		
Function: long deleteLine(String documentHash, long line, long userID, long sequenceNo)		
Purpose: Test if a non existing line can be deleted		
Input:	documentHash: line: userid: sequenceNo:	From loadDocument() 30 123 5
Expected Output:	-8	
Actual Output:	-8	

Test case #: 7	
Function: long deleteLine(String documentHash, long line, long userID, long sequenceNo)	
Purpose: Test if all the above delete operations were saved	
Input:	All of the above 6 cases were performed sequentially
Expected Output:	testDeleteLine +"\n" + "Test cases for deleteLine" + "\n" + "\n" + "A bungch of chars like #$%, CAPS and 53487" + "\n" + "\n" + "I guess I can put something on this line, right?" + "\n" + "\n" + "10th Line" + "\n" + "\n" + "\n" + "\n" + "\n" + "\n" + "\n" + "Many Many lines" + "\n" + "\n" + "This is long" + "\n" + "This" + "\n" + "is" + "\n" + "for" + "\n" + "a" + "\n" + "greater" + "\n" + "good!" + "\n" + "\n" + "\n" + "\n" + "Last line after all ops" + "\n"
Actual Output:	testDeleteLine +"\n" + "Test cases for deleteLine" + "\n" + "\n" + "A bungch of chars like #$%, CAPS and 53487" + "\n" + "\n" + "I guess I can put something on this line, right?" + "\n" + "\n" + "10th Line" + "\n" + "\n" + "\n" + "\n" + "\n" + "\n" + "\n" + "Many Many lines" + "\n" + "\n" + "This is long" + "\n" + "This" + "\n" + "is" + "\n" + "for" + "\n" + "a" + "\n" + "greater" + "\n" + "good!" + "\n" + "\n" + "\n" + "\n" + "Last line after all ops" + "\n"

Delete the last empty line
Delete the 1st line
Delete the 1st line
Delete the 2nd line
Delete an empty line on 6th line
Delete an empty line on 5th line
Delete a non-existent line
Delete the non-empty last line after all the above operations
Test if the above operations were performed and saved to the document

3.9 getLine()

Test case #: 1		
Function: String getLine(String documentHash, long line, long sequenceNo)		
Purpose: Test to see if a line can be obtained when the document is not loaded		
Input:	documentHash: line: sequenceNo:	Not_loaded 1 0
Expected Output:	-3	
Actual Output:	-3	

Test case #: 2		
Function: String getLine(String documentHash, long line, long sequenceNo)		
Purpose: Boundary Test to see if the 1st line can be obtained		
Input:	documentHash: line: sequenceNo:	From loadDocument() 1 0
Expected Output:	This is a test\n	

Test case #: 2	
Actual Output:	This is a test\n

Test case #: 3		
Function: String getLine(String documentHash, long line, long sequenceNo)		
Purpose: Boundary Test to see if the 2nd line can be obtained		
Input:	documentHash: line: sequenceNo:	From loadDocument() 2 0
Expected Output:	Test cases for deleteLine\n	
Actual Output:	Test cases for deleteLine\n	

Test case #: 4		
Function: String getLine(String documentHash, long line, long sequenceNo)		
Purpose: Boundary Test to see if an empty line, found around the middle of the document, can be obtained		
Input:	documentHash: line: sequenceNo:	From loadDocument() 6 0
Expected Output:	\n	
Actual Output:	\n	

Test case #: 5
Function: String getLine(String documentHash, long line, long sequenceNo)

Test case #: 5		
Purpose: Boundary Test to see if a non-empty line, found around the middle of the document, can be obtained		
Input:	documentHash: line: sequenceNo:	From loadDocument() 10 0
Expected Output:	10th Line\n	
Actual Output:	10th Line\n	

Test case #: 6		
Function: String getLine(String documentHash, long line, long sequenceNo)		
Purpose: Test to see if the 1st line of an empty document can be obtained		
Input:	documentHash: line: sequenceNo:	From loadDocument() 1 0
Expected Output:	\n	
Actual Output:	\n	

Test case #: 7		
Function: String getLine(String documentHash, long line, long sequenceNo)		
Purpose: Test to see if the same line is obtained when getLine is called more than once on a one line document		
Input:	documentHash: line: sequenceNo:	From loadDocument() 1 0
Expected Output:	abc123^&*(\n	

Test case #: 7	
Actual Output:	abc123^&*(\n

Test case #: 8		
Function: String getLine(String documentHash, long line, long sequenceNo)		
Purpose: Test to see if the same 2nd line is obtained when getLine is called more than once on a long document		
Input:	documentHash: line: sequenceNo:	From loadDocument() 2 0
Expected Output:	This is a test\n	
Actual Output:	This is a test\n	

Test case #: 9		
Function: String getLine(String documentHash, long line, long sequenceNo)		
Purpose: Test to see if the same 6th line is obtained when getLine is called more than once on a long document		
Input:	documentHash: line: sequenceNo:	From loadDocument() 6 0
Expected Output:	\n	
Actual Output:	\n	

Test case #: 10		
Function: String getLine(String documentHash, long line, long sequenceNo)		
Purpose: Test to see if the same last line is obtained when getLine is called more than once on a long document		
Input:	documentHash: line: sequenceNo:	From loadDocument() 31 0
Expected Output:	This is a new line, freshly added\n	
Actual Output:	This is a new line, freshly added\n	

Test case #: 11	
Function: String getLine(String documentHash, long line, long sequenceNo)	
Purpose: Test to see if a adding and deleting some lines do not affect anything	
Input:	documentHash: From loadDocument() line: 2 sequenceNo: 2 Procedure: Get 4th line = x Add a new line at line 3, Get 5th line = y Delete line 3 Get 4th line = z Compare x, y, z
Expected Output:	Equal
Actual Output:	Equal

Test consecutive getLines() using different sequenceNo must not change anything
Get 6th line
Add 2nd line
Get 7th line
Add 2nd line
Using different sequenceNo but same 30th line
Close the document if it is already loaded

Test that getLine cannot be used when a document is not loaded

Test if getLine can get the empty string
Close the document if it is already loaded
Load the empty document

Test if getLine always return the same string
One line document

Test on one line document

Test on a bigger document

Test getting the 2nd line of a big document

Test getting the 6th line of a big document

Test getting the empty last line of a big document

Test if getLine can correctly get the lines that are added to the document
Add a non-empty line on the 2nd line and see if getLine can retrieve it
Add a non-empty line on the last line and see if getLine can retrieve it

Test if getLine() works when adding and deleting some lines
Add a non-empty line on 2nd line
delete the line that was previously added

3.10 getLocks() and getLocksAndUIDs()

Test GetLocks and GetLocksAndUIDs on a document that is not loaded
- Close the document if it is already loaded
- User 1 locks a line
- Verify that null strings are received

Test GetLocks and GetLocksAndUIDs on an empty one-line document
- Close the document if it is already loaded
- Load the empty document
- Verify that there are no locks on the document
- User 3 locks the line
- Verify that the line is locked by User 3
- User 3 renews the lock on the line
- Verify that the line is still locked by User 3
- User 2 tries to lock the line
- User 1 tries to unlock the line
- Verify that the line is still locked by User 3
- User 3 unlocks the line
- Verify that there are no locks on the document
- User 3 tries to unlock the line again
- Verify that there are still no locks on the document
- close the empty document

Test GetLocks and GetLocksAndUIDs on a non-empty one line document
- Close the document if it is already loaded
- Load the one line document
- Verify that there are no locks on the document
- User 1 locks the line
- Verify that the line is locked by User 1
- User 2 tries to lock the line
- User 3 tries to lock the line
- User 1 renews the lock on the line
- Verify that the line is still locked by User 1
- User 1 unlocks the line
- Verify that there are no locks on the document
- User 1 tries to unlock the line again
- Verify that there are still no locks on the document

Close the document

Test GetLocks and GetLocksAndUIDs on a document with two lines
Close the document if it is already loaded
Load the two line document
Verify that there are no locks on the document
User 1 tries to unlock the first line
Verify that there are still no locks on the document
User 2 locks the first line
User 2 locks the second line
Verify that both the lines are locked by User 2
User 3 tries to unlock the first line
User 3 tries to lock the first line
User 3 tries to unlock the second line
User 3 tries to lock the second line
User 3 tries to unlock the first line again
User 3 tries to lock the second line again
Verify that both the lines are still locked by User 2
User 2 renews the lock on the second line
Verify that both the lines are still locked by User 2
Close the document

Test GetLocks and GetLocksAndUIDs on a document with two lines where the
first line is empty and the second line doesn't end with a \n
Close the document if it is already loaded
Load the two line document
Verify that there are no locks on the document
User 1 tries to unlock the first line
Verify that there are still no locks on the document
User 2 locks the second line
Verify that the second line is locked by User 2
User 2 locks the first line
Verify that both the lines are locked by User 2
User 3 tries to lock the first line
User 3 tries to unlock the first line
User 3 tries to unlock the second line
User 3 tries to lock the second line
User 3 tries to unlock the first line again
User 3 tries to lock the second line again

Verify that both the lines are still locked by User 2
User 2 renews the lock on the second line
Verify that both the lines are still locked by User 2
Close the document

Test GetLocks and GetLocksAndUIDs on a two line document where both the
lines are empty

Close the document if it is already loaded
Load the two line document
Verify that there are no locks on the document
User 1 tries to unlock the first line
Verify that there are still no locks on the document
User 2 locks the second line
User 2 locks the first line
Verify that both the lines are locked by User 2
User 3 tries to lock the first line
User 3 tries to unlock the first line
User 3 tries to unlock the second line
User 3 tries to lock the second line
User 3 tries to unlock the first line again
User 3 tries to lock the second line again
Verify that both the lines are still locked by User 2
User 2 renews the lock on the second line
Verify that both the lines are still locked by User 2
Again verify that both the lines are still locked by User 2
Close the document

Test GetLocks and GetLocksAndUIDs on a document with three lines where

the third line is empty
Close the document if it is already loaded
Load the two line document
Verify that there are no locks on the document
User 1 locks the first line
User 2 tries to lock the first line
User 2 tries to unlock the first line
User 1 unlocks the first line
Verify that there are still no locks on the document
User 3 locks the third line
User 3 locks the second line
Verify that the second and third lines are locked by User 3
User 2 tries to lock the second line

User 2 tries to lock the third line
User 2 tries to unlock the third line
User 2 tries to unlock the third line again
User 2 tries to unlock the second line
User 2 tries to unlock the second line again
Verify that the second and third lines are still locked by User 3
User 3 renews the lock on the third line
Verify that the second and third lines are still locked by User 3
User 3 unlocks the third line
User 3 unlocks the second line
User 2 locks the second line
Verify that the second line is locked by User 2
Close the document

Test GetLocks and GetLocksAndUIDs on a three line document where all the
lines are empty
Close the document if it is already loaded
Load the two line document
Verify that there are no locks on the document
User 1 locks the first line
User 2 tries to lock the first line
User 2 tries to unlock the first line
User 1 unlocks the first line
Verify that there are still no locks on the document
User 3 locks the third line
User 3 locks the second line
Verify that the second and third lines are locked by User 3
User 2 tries to lock the second line
User 2 tries to lock the third line
User 2 tries to unlock the third line
User 2 tries to unlock the third line again
User 2 tries to unlock the second line
User 2 tries to unlock the second line again
Verify that the second and third lines are still locked by User 3
User 3 renews the lock on the third line
Verify that the second and third lines are still locked by User 3
User 3 unlocks the third line
User 3 unlocks the second line
User 2 locks the second line
Verify that the second line is locked by User 2
Close the document

Test GetLocks and GetLocksAndUIDs on a large document
- Close the document if it is already loaded
- Load the document
- Verify that there are no locks on the document
- User 3 tries to lock an invalid line
- User 1 tries to lock an invalid line
- User 2 tries to lock an invalid line: one line past the last line in the document
- Verify that there are no locks on the document
- User 1 tries to unlock the ninth line
- User 1 locks the fifteenth line
- User 1 locks the sixteenth line
- User 1 locks the seventeenth line
- User 1 locks the eighteenth line
- User 2 locks the third line
- User 2 locks the fourth line
- User 2 locks the fifth line
- User 2 locks the sixth line
- User 1 renews the lock on the sixteenth line
- User 1 renews the lock on the seventeenth line
- User 1 renews the lock on the eighteenth line
- Verify that the line locks have occurred
- User 3 tries to lock the fifth line
- User 3 tries to lock the fifth line again
- User 3 locks the eighth line
- User 3 locks the ninth line
- User 3 locks the tenth line
- User 3 locks the eleventh line
- User 1 unlocks on the seventeenth line
- User 1 unlocks on the eighteenth line
- Verify that the line locks have occurred
- User 3 tries to lock the fifth line
- User 2 renews the lock on the third line
- User 1 locks the last line
- User 1 renews the lock on the last line
- User 1 unlocks the last line
- User 1 unlocks the fifteenth line
- User 1 unlocks the sixteenth line
- User 3 unlocks the eighth line
- User 3 unlocks the ninth line
- User 3 unlocks the tenth line
- User 3 unlocks the eleventh line
- User 2 unlocks the sixth line

User 2 unlocks the fifth line
User 2 unlocks the fourth line
User 2 unlocks the third line
Verify that there are no locks on the document
Close the document

3.11 getSequenceNo()

Test GetSequenceNo on a document that is not loaded
Close the document if it is already loaded
User 1 locks the first line : should fail
User 1 updates the first line
User 1 inserts text in the first line
User 1 adds a line
User 1 deletes the added line
User 1 unlocks the first line

Test GetSequenceNo after a LockLine operation
Should return same sequenceNo
User 1 tries to lock an invalid line (should return same sequenceNo)
User 1 locks the second line (should return same sequenceNo)

Test GetSequenceNo after an UnlockLine operation
User 1 tries to unlock an invalid line (should return same sequenceNo)
User 1 locks the second line (should return same sequenceNo)
User 1 unlocks the second line (should return same sequenceNo)

Test GetSequenceNo on AddLine operations
User 1 adds a line after the first line
User 1 adds a line after the first line

Test GetSequenceNo on Insert operations
User 1 inserts text in the third line
User 1 inserts text in the fourth line

Test GetSequenceNo on DeleteLine operations
User 1 deletes the first line

User 1 deletes the fourth line

Test GetSequenceNo on UpdateLine operations
User 1 updates the second line
User 1 updates the fifth line

3.12 insert()

Insert "middle" in between "Line" and "1" in line 1
Insert "before" as the first word in line 1
Insert "last" as the last word in line 2
Replace "Line" as "This is line" in line 3

3.13 lockLine()

Test lockLine on a document that is not loaded
Close the document if it is already loaded

Test that a line cannot be locked on a document that is not loaded

Test that a line cannot be locked on a document that is not loaded; ensure that the appropriate error code is received

Test that a line cannot be locked on a document that is not loaded; ensure that the appropriate error code is received

Test that a line cannot be locked on a document that is not loaded; ensure that the appropriate error code is received

Test lockLine on an empty one-line document
Close the document if it is already loaded
Load the empty document

Testing invalid lines as input:
User 1 tries to lock an invalid line
User 1 tries to lock an invalid line

User 1 tries to lock an invalid line: one line past the last line in the document

Testing line locking conflicts:
User 1 locks the line
User 1 renews the lock on the line
User 2 tries to lock the line
Verify that the above locks have occurred
Close the empty document

Test lockLine on a non-empty one line document
Close the document if it is already loaded
Load the one line document

Testing invalid lines as input:
User 2 tries to lock an invalid line
User 2 tries to lock an invalid line
User 2 tries to lock an invalid line: one line past the last line in the document

Testing line locking conflicts:
User 1 locks the line
User 2 tries to lock the line
User 3 tries to lock the line
User 1 renews the lock on the line
Verify that the above locks have occurred
Close the document

Test lockLine on a document with two lines
Close the document if it is already loaded
Load the two line document

Testing invalid lines as input:
User 3 tries to lock an invalid line
User 1 tries to lock an invalid line
User 2 tries to lock an invalid line: one line past the last line in the document

Testing line lock conflicts:
User 1 locks the first line
User 2 tries to lock the first line
User 2 locks the second line
User 3 tries to lock the first line

User 3 tries to lock the second line
User 2 renews the lock on the second line
Verify that the above locks have occurred
Close the document

Test lockLine on a document with two lines where the first line is empty and the second line doesn't end with a \n
Close the document if it is already loaded
Load the two line document

Testing invalid lines as input:
User 3 tries to lock an invalid line
User 3 tries to lock an invalid line
User 3 tries to lock an invalid line: one line past the last line in the document

Testing line lock conflicts:
User 1 locks the first line
User 2 tries to lock the first line
User 2 locks the second line
User 2 renews the lock on the second line
User 3 tries to lock the first line
User 3 tries to lock the second line
Verify that the above locks have occurred
Close the document

Test lockLine on a two line document where both the lines are empty
Close the document if it is already loaded
Load the two line document

Testing invalid lines as input:
User 1 tries to lock an invalid line
User 1 tries to lock an invalid line
User 1 tries to lock an invalid line: one line past the last line in the document

Testing line lock conflicts:
User 1 locks the first line
User 2 tries to lock the first line
User 2 locks the second line

User 3 tries to lock the first line
User 3 tries to lock the second line
User 2 renews the lock on the second line
Verify that the above locks have occurred
Close the document

Test lockLine on a document with three lines where the third line is empty
Close the document if it is already loaded
Load the two line document

Testing invalid lines as input:
User 1 tries to lock an invalid line
User 2 tries to lock an invalid line
User 3 tries to lock an invalid line: one line past the last line in the document

Testing line lock conflicts:
User 1 locks the first line
User 2 tries to lock the first line
User 3 locks the second line
User 3 locks the third line
User 2 tries to lock the second line
User 2 tries to lock the third line
User 2 tries to lock the first line again
User 3 renews the lock on the third line
Verify that the above locks have occurred
Close the document

Test lockLine on a three line document where all the lines are empty
Close the document if it is already loaded
Load the two line document

Testing invalid lines as input:
User 1 tries to lock an invalid line
User 1 tries to lock an invalid line
User 1 tries to lock an invalid line: one line past the last line in the document
Testing line lock conflicts:
User 1 locks the first line
User 2 tries to lock the first line

User 3 locks the second line
User 3 locks the third line
User 2 tries to lock the second line
User 2 tries to lock the third line
User 2 tries to lock the first line again
User 3 renews the lock on the second line
Verify that the above locks have occurred
Close the document

Test lockLine on a large document
Close the document if it is already loaded
Load the document

Testing invalid lines as input:
User 3 tries to lock an invalid line
User 1 tries to lock an invalid line
User 2 tries to lock an invalid line: one line past the last line in the document

Testing line lock conflicts:
User 1 locks the first line
User 1 locks the fifth line
User 1 renews the lock on the fifth line
User 2 tries to lock the first line
User 2 tries to lock the first line again
User 2 lock the eighth line
User 3 locks the third line
User 3 locks the fourth line
User 2 tries to lock the third line
User 2 tries to lock the fourth line
User 3 tries to lock the fifth line
User 2 tries to lock the third line again
User 2 locks the last line
User 2 renews the lock on the last line
Verify that the above locks have occurred
Close the document

3.14 addLine()

Test case #: 1

Test case #: 1		
Function: long addLine(long documentHash, long line, long userid, string toAdd, long sequenceNo)		
Purpose: Test if addLine can be done when a document is loaded Test if one non-empty line can be added		
Input:	documentHash: line: userid: toadd: sequenceNo:	From loadDocument() 1 123 Adding 1st line 0
Expected Output:	1	
Actual Output:	1	

Test case #: 2		
Function: long addLine(long documentHash, long line, long userid, string toAdd, long sequenceNo)		
Purpose: Test if more than one line can be added		
Input:	documentHash: line: userid: toadd: sequenceNo:	From loadDocument() 1 123 Addine more than 1 line\nThis is the 2nd line\n3rd line is here\n 1
Expected Output:	2	
Actual Output:	2	

Test case #: 3
Function: long addLine(long documentHash, long line, long userid, string toAdd, long sequenceNo)
Purpose: Test if a non-empty line can be added

Test case #: 3		
Input:	documentHash: line: userid: toadd: sequenceNo:	From loadDocument() 1 123 “” 2
Expected Output:	3	
Actual Output:	3	

Test case #: 4		
Function: long addLine(long documentHash, long line, long userid, string toAdd, long sequenceNo)		
Purpose: Test if a combination of various characters can be added in a line		
Input:	 documentHash: line: userid: toadd: sequenceNo:	From loadDocument() 1 123 a b 1 @ ~ JUNK\n oh & > ? <{ |||__-- 3
Expected Output:	4	
Actual Output:	4	

Test case #: 5		
Function: long addLine(long documentHash, long line, long userid, string toAdd, long sequenceNo)		
Purpose: Boundary test if a combination of various characters can be added on the 2nd line		
Input:	 documentHash: line: userid: toadd: sequenceNo:	From loadDocument() 2 123 a b 1 @ ~ JUNK\n oh & > ? <{ |||__-- 4

Test case #: 5	
Expected Output:	5
Actual Output:	5

Test case #: 6		
Function: long addLine(long documentHash, long line, long userid, string toAdd, long sequenceNo)		
Purpose: Boundary test if an empty line can be added around the middle of the document		
Input:	documentHash: line: userid: toadd: sequenceNo:	From loadDocument() 6 123 “” 5
Expected Output:	6	
Actual Output:	6	

Test case #: 7		
Function: long addLine(long documentHash, long line, long userid, string toAdd, long sequenceNo)		
Purpose: Boundary test if more than 1 line can be added at the end of the document		
Input:	documentHash: line: userid: toadd: sequenceNo:	From loadDocument() 15 123 Adding more than 1 line\nThis is the 2nd line\n3rd line is here\n 6
Expected Output:	7	

Test case #: 7	
Actual Output:	7

Test case #: 8		
Function: long addLine(long documentHash, long line, long userid, string toAdd, long sequenceNo)		
Purpose: Test if a line can be added when a document is not loaded		
Input:	documentHash: line: userid: toadd: sequenceNo:	notExist 15 123 Adding more than 1 line\nThis is the 2nd line\n3rd line is here\n 7
Expected Output:	-3	
Actual Output:	-3	

Test case #: 9	
Function: long addLine(long documentHash, long line, long userid, string toAdd, long sequenceNo)	
Purpose: Test if the newly added lines are saved to the document	
Input:	All the above 8 test cases were performed sequentially

Test case #: 9	
Expected Output:	testAddLine +"\n"+ "a b 1 @ ~ JUNK"+"\n"+ "a b 1 @ ~ JUNK"+"\n"+ " oh & > ? <{ |||__--"+"\n"+ " oh & > ? <{ |||__--"+"\n"+"\n"+"\n"+ "Adding more than 1 line"+"\n"+ "This is the 2nd line"+"\n"+ "3rd line is here"+"\n"+ "Adding 1st line"+"\n"+ "aasd"+"\n"+ "asd afsasdf asdfasddf"+"\n"+ "sdf asdfasdf" + identifier + "d"+"\n"+ "Adding more than 1 line"+"\n"+ "This is the 2nd line"+"\n"+ "3rd line is here"+"\n"
Actual Output:	testAddLine +"\n"+ "a b 1 @ ~ JUNK"+"\n"+ "a b 1 @ ~ JUNK"+"\n"+ " oh & > ? <{ |||__--"+"\n"+ " oh & > ? <{ |||__--"+"\n"+"\n"+"\n"+ "Adding more than 1 line"+"\n"+ "This is the 2nd line"+"\n"+ "3rd line is here"+"\n"+ "Adding 1st line"+"\n"+ "aasd"+"\n"+ "asd afsasdf asdfasddf"+"\n"+ "sdf asdfasdf" + identifier + "d"+"\n"+ "Adding more than 1 line"+"\n"+ "This is the 2nd line"+"\n"+ "3rd line is here"+"\n"

3.15 updateLine()

Update first Line
Update last line but lock line first
Update a non-existing line
Update Line 6 but lock and unlock first
Update Line 6 with empty String
Update Line 2 with a new line
Update Line in a Document that does not exist

Update Line 0
Update an empty line
Check if filename contents matches the changes

3.16 unlock()

Test unlockLine on a document that is not loaded
Close the document if it is already loaded

Test that a line cannot be unlocked on a document that is not loaded; ensure that the appropriate error code is received

Test that a line cannot be unlocked on a document that is not loaded; ensure that the appropriate error code is received

Test that a line cannot be unlocked on a document that is not loade d; ensure that the appropriate error code is received

Test that a line cannot be unlocked on a document that is not loaded; ensure that the appropriate error code is received
User 1 locks the first line

Test that the first line cannot be unlocked on a document that is not loaded; ensure that the appropriate error code is received

Test unlockLine on an empty one-line document
Close the document if it is already loaded
Load the empty document

Testing invalid lines as input:
User 3 tries to unlock an invalid line
User 2 tries to unlock an invalid line
User 1 tries to unlock an invalid line: one line past the last line in the document
Testing line unlocking conflicts:
User 1 locks the line
User 2 tries to unlock the line
User 1 unlocks the line
Verify the lack of locks
Close the empty document

Test unlockLine on a non-empty one line document
Close the document if it is already loaded
Load the two line document

Testing invalid lines as input:
User 3 tries to unlock an invalid line
User 3 tries to unlock an invalid line
User 3 tries to unlock an invalid line: one line past the last line in the document

Testing line unlocking conflicts:
User 1 locks the line
User 1 unlocks the line
User 2 tries to unlock the line
Verify the lack of locks
Close the document

Test unlockLine on a document with two lines
Close the document if it is already loaded
Load the two line document
Testing invalid lines as input:
User 1 tries to unlock an invalid line
User 1 tries to unlock an invalid line
User 2 tries to unlock an invalid line: one line past the last line in the document

Testing line unlocking conflicts:
User 1 locks the first line
User 2 locks the second line
User 3 tries to unlock the second line
User 2 tries to unlock the first line
User 2 unlocks the second line
Verify that the first line is still locked
Close the document

Test unlockLine on a document with two lines where the first line is empty

Close the document if it is already loaded
Load the two line document

Testing invalid lines as input:

User 2 tries to unlock an invalid line
User 1 tries to unlock an invalid line
User 3 tries to unlock an invalid line: one line past the last line in the document

Testing line unlocking conflicts:

User 1 locks the second line
User 2 locks the first line
User 2 unlocks the first line
User 2 tries to unlock the first line again
User 2 tries to unlock the second line
User 1 unlocks the second line
Verify the lack of locks
Close the document

Test unlockLine on a two line document where both the lines are empty

Close the document if it is already loaded
Load the two line document

Testing invalid lines as input:

User 3 tries to unlock an invalid line
User 3 tries to unlock an invalid line
User 2 tries to unlock an invalid line: one line past the last line in the document

Testing line unlocking conflicts:

User 1 locks the first line
User 2 locks the second line
User 1 tries to unlock the second line
User 1 tries to unlock the second line again
User 1 unlocks the first line

User 1 tries to unlock the first line again
Verify that line 2 is locked
Close the document

Test unlockLine on a document with three lines where the third line is empty
Close the document if it is already loaded
Load the two line document

Testing invalid lines as input:
User 3 tries to unlock an invalid line
User 2 tries to unlock an invalid line
User 3 tries to unlock an invalid line: one line past the last line in the document

Testing line unlocking conflicts:
User 2 locks the first line
User 2 locks the second line
User 1 locks the third line
User 3 tries to unlock the second line
User 2 unlocks the first line
User 1 unlocks the third line
User 1 tries to unlock the third line again
Verify that the above unlocks have occurred
Close the document

Test unlockLine on a three line document where all the lines are empty
Close the document if it is already loaded
Load the two line document

Testing invalid lines as input:
User 3 tries to unlock an invalid line
User 2 tries to unlock an invalid line
User 3 tries to unlock an invalid line: one line past the last line in the document

Testing line unlocking conflicts:
User 3 locks the first line
User 2 locks the second line
User 1 locks the third line

User 3 tries to unlock the third line
User 2 tries to unlock the first line
User 1 tries to unlock the second line
User 1 tries to unlock the second line again
Verify that the above locks haven't been unlocked
Close the document

Test unlockLine on a large document
Close the document if it is already loaded
Load the document

Testing invalid lines as input:
User 1 tries to unlock an invalid line
User 1 tries to unlock an invalid line
User 3 tries to unlock an invalid line: one line past the last line in the document

Testing line unlocking conflicts:
User 1 locks the first line
User 1 locks the fifth line
User 2 locks the eighth line
User 3 locks the third line
User 3 locks the fourth line
User 2 tries to unlock the first line
User 2 tries to unlock the first line again
User 3 tries to unlock the eighth line
User 3 tries to unlock the ninth line
User 3 locks the ninth line
User 2 unlocks the eighth line
User 3 tries to unlock the seventh line
User 2 tries to unlock the third line
User 3 locks the last line
User 1 tries to unlock the last line
Verify that the above unlocks have occurred
Close the document

4. Appendices

4.1 Appendix A – Shared Components

The following shared components Reflection and Logging will be used while testing and debugging new components for the software system. Their APIs detail their use.

4.2.1 Reflection API

Provides reflection based support to objects.

4.2.1.1 ObjectNameSpace

```
/**
 * Determines the namespace to which an object belongs to.
 *
 * @Object the object whose namespace is to be determined
 * @return namespace of Object, null if Object is null
 */

 public static String ObjectNamespace(Object Object)
```

4.2.1.2 ObjectQualifiedTypeName

```
/**
 * Determines fully qualified type name of an object.
 *
 * @Object the object whose qualified type is to be determined
 * @return qualified type name of Object, null if Object is null
 */

  public static String ObjectQualifiedTypeName (Object Object)
```

4.2.1.3 ObjectShortTypeName

```
/**
 * Determines short type name of an object (namespace excluded).
 *
 * @Object the object whose type is to be determined
 * @return short type name of Object, null if Object is null
 */
```

public static String ObjectShortTypeName (Object Object)

4.2.1.4 MethodStackTrace

```
/**
 * Determines stack trace of a program at any stage excluding
 * call to MethodStackTrace.
 *
 * @return stack trace at a current point in the program execution
 */
```

public static StackTraceElement[] MethodStackTrace()

4.2.2 Logging API

Logs program status to a log file.

4.2.2.1 SetLevel

```
/**
 * Sets the new default log level for log messages.
 *
 * @NewLevel the new default log level of output logs
 * @ServiceException if an error incurred while changing the service or
 * if service is inactive
 */
```

public void SetLevel (Level NewLevel) throws ServiceException

4.2.2.2 LogMessage

```
/**
 * Logs a message under the default log level.
 *
 * @Message string representation of the message to be logged
 * @ServiceException if an error incurred while logging the message or
 * if service is inactive
 */
```

public void LogMessage(String Message) throws ServiceException

```
/**
 * Logs a message under the default log level.
 *
 * @Message string representation of the message to be logged
 * @Param parameters to be logged along with message
 * @ServiceException if an error incurred while logging the message or
 * if service is inactive
 */
```

public void LogMessage(String Message, Object Param) throws ServiceException

```
/**
 * Logs a message with the specified log level.
 *
 * @LogLevel level of output log message
 * @Message string representation of the message to be logged
 * @ServiceException if an error incurred while logging the message or
 * if service is inactive
 */
```

public void LogMessage(Level LogLevel, String Message) throws ServiceException

```
/**
 * Logs a message with the specified log level.
 *
 * @LogLevel level of output log message
 * @Message string representation of the message to be logged
 * @Param parameters to be logged along with message
 * @ServiceException if an error incurred while logging the message or
 * if service is inactive
 */
```

public void LogMessage(Level LogLevel, String Message, Object Param) throws ServiceException

4.2.2.3 LogEntrance

```
/**
 * Logs a procedure entrance under the DEBUG log level.
 *
 * @ServiceException if an error incurred while logging the message or
 * if service is inactive
 */
```

public void LogEntrance() throws ServiceException

```
/**
 * Logs a procedure entrance under the DEBUG log level.
 *
 * @Param parameters to be logged along with message
 * @ServiceException if an error incurred while logging the message or
 * if service is inactive
 */
```

public void LogEntrance(Object[] Params) throws ServiceException

4.2.2.4 LogExit

```
/**
 * Logs a procedure departure under the DEBUG log level.
```

```
 *
 * @ServiceException if an error incurred while logging the message or
 * if service is inactive
 */
public void LogExit() throws ServiceException

/**
 * Logs a procedure departure under the DEBUG log level.
 *
 * @Param parameters to be logged along with message
 * @ServiceException if an error incurred while logging the message or
 * if service is inactive
 */
 public void LogExit(Object[] Params) throws ServiceException
```

4.2 Appendix B - Test Form Checklist

Test Form Checklist		
Test case #:		Date(dd/mm/yy):
Program Name:		Function Name:
Purpose:		
Expected Output		
Input		
Output		
Additional Notes:		
PASS?	*Yes*	*No*
Tester's Name		Tester's signature

4.3 Appendix C - Bug Tracking Form

Bug Tracking Form		
Bug form ID:	Date bug found (dd/mm/yy):	
Bug reported by:		
Bug Name or ID:		
Bug description / Actions to replicate bug:		
Comments:		
Bug classification (circle one): *Minor* *Major* *Severe*		
Bug status (complete all that apply):		
Verified By:	*Assigned* To:	*Fixed* Date:
Other comments:		

4.4 Appendix D – Test Schedules

- Phase B will start on the 8th of October 2004
- Phase C will start on the 20th of November 2004

For more details, please refer to the Gannt Chart on the following pages.