OMNIEDITOR PROJECT

PHASE B: UNIT TEST PLAN

CODENAME: NIRVANA

CSC408F1
Group Members:

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

1 Functional 3
1.1 Structure of Server 3
1.2 Strategy 5
1.2.1 JUnit Test 5
1.2.2 Details of our Unit Test Strategy 5
1.2.3 Tips to Unit Testing 6
1.3 Unit Test Cases 6
1.3.1 loadDocument 7
1.3.2 closeDocument 8
1.3.3 saveDocument 8
1.3.4 getDocument 9
1.3.5 browse 10
1.3.6 updateLine 11
1.3.7 insert 12
1.3.8 delete 14
1.3.9 lockLine 16
1.3.10 unlockLine 19
1.3.11 getLocks 21
1.3.12 getLocksAndUIDs 23
1.3.13 getSequenceNo 25
1.3.14 addLine 26
1.3.15 deleteLine 29
1.3.16 getLine 31
1.4 Bug Reports 35
1.4.1 Bug Report 35
1.4.2 Severity versus Resolution 36
1.4.3 Status versus Resolution 36
1.5 Integration Test Plan 37
1.5.1 GENERAL INFORMATION 37
1.5.2 Test Definition 38
1.5.3 Test Execution 39
2 Non Functional 41
2.1 Measuring Correctness and Reliability 41
2.2 Measuring Performance/Scalability and Complexity 42
Time 44
Time --verbose 44
-Prof 45
The Heap Summary 46
The Heap Allocation Sites 48
The Heap Allocation Sites (continued) 49
The Heap Dump 50
The Heap Dump (continued) 51
Halstead Complexity 51
Mccabe Complexity 52

1 Functional

1.1 Structure of Server

1.2 Strategy

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.

1.2.1 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

1.2.2 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!

1.2.3 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.

1.3 Unit Test Cases

1.3.1 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.

1.3.2 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.

1.3.3 saveDocument

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: Test:	Null Successful

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

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

1.3.4 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: Test:	Null 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.

1.3.5 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.

1.3.6 updateLine

long updateLine(String documentHash, long line, long userID, String updatedLine, long sequenceNo)

Purpose: Update line with the replacement line.

Parameters:

- documentHash – The documentHash desired for updating
- line – Line # where to make the update
- userId – The user identity
- updatedLine – The line to be added
- sequenceNo – The state of the user's document

Return Value: Returns the sequence number or an appropriate ErrorCode.

Test case #: 1	
Input:	updateLine(<valid docHash>, <valid line>, <userId>, <valid updatedLine>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

Test case #: 2	
Input:	updateLine(<valid docHash>, <inValid line>, <userId>, <valid updatedLine>, <sequenceNo>)
Output:	ErrorCodes.CANNOT_LOCK
Test:	Successful

Test case #: 3	
Input:	updateLine(<valid docHash>, <valid line>, <userId>, <Invalid updatedLine>, <sequenceNo>)
Output:	ErrorCodes.CANNOT_EDIT_MULTIPLE_LINES
Test:	Successful

Test case #: 4	
Input:	updateLine(<Invalid docHash>, <valid line>, <userId>, <valid updatedLine>, <sequenceNo>)
Output:	ErrorCodes.DOCUMENT_DOES_NOT_EXIST
Test:	Successful

Test case #: 5

Input:	For testing lock with updateLine: Lock the line first : lockLine(<valid documentHash>, line 8, <userid>, <sequenceNo>) updateLine(<Valid docHash>, line 8, <userId>, <valid updatedLine>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

Test case #: 6	
Input:	For testing lock and unlock with updateLine: Lock the line first : lockLine(<valid documentHash>, line 6, <userid>, <sequenceNo>) Then, unlock line : unlockLine(<valid documentHash>, line 6, <userid>, <sequenceNo>) updateLine(<Invalid docHash>, line 6, <userId>, <valid updatedLine>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

1.3.7 insert

insert(String documentHash, long line, long userID, int startPosition, int endPosition, String toInsert, long sequenceNo)

Purpose: Insert text to the current position.

Parameters:

- documentHash – The documentHash desired for updating
- line – Line # where to make the update
- userId – The user identity
- startPosition – The start position for this operation
- endPosition – The end position for this operation
- toInsert – The line to be inserted
- sequenceNo – The state of the user's document

Return Value: Returns the sequence number or an appropriate ErrorCode.

Test case #: 1	
Input:	insert(<valid docHash>, <valid line>, <userId>, <valid startPos>, <valid endPos>, <valid updatedLine>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

Test case #: 2	
Input:	insert(<valid docHash>, <valid line>, <userId>, <invalid startPos>, <invalid endPos>, <valid updatedLine>, <sequenceNo>)
Output:	ErrorCodes.EDITING_OUTSIDE_BOUNDS

Test:	Successful

Test case #: 3	
Input:	insert(<valid docHash>, <invalid line>, <userId>, <valid startPos>, <valid endPos>, <valid updatedLine>, <sequenceNo>)
Output:	ErrorCodes.CANNOT_LOCK
Test:	Successful
Test case #: 4	
Input:	insert(<invalid docHash>, <valid line>, <userId>, <valid startPos>, <valid endPos>, <valid updatedLine>, <sequenceNo>)
Output:	ErrorCodes.DOCUMENT_DOES_NOT_EXIST
Test:	Successful

Test case #: 5	
Input:	For testing lock with insert: lockLine(<valid docHash>, line 1, <userid>, <sequenceNo>) insert(<valid docHash>, line 1, <userId>, <valid startPos>, <valid endPos>, <valid updatedLine>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

Test case #: 6	
Input:	For testing lock, unlock with insert: lockLine(<valid docHash>, line 1, <userid>, <sequenceNo>) unlockLine(<valid docHash, line 1, <userid>, <sequenceNo>) insert(<valid docHash>, line 1, <userId>, <valid startPos>, <valid endPos>, <valid updatedLine>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

Test case #: 7	
Input:	For testing 2 clients: lockLine(<valid docHash>, line 1, user 1, <sequenceNo>) unlockLine(<valid docHash>, line 1, user 1, <sequenceNo>) insert(<valid docHash>, line 1, user 2, <valid startPos>, <valid endPos>, <valid updatedLine>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

Test case #: 8

Input:	For testing 2 clients: lockLine(<valid docHash>, line 1, user 1, <sequenceNo>) insert(<valid docHash>, line 1, user 2, <valid startPos>, <valid endPos>, <valid updatedLine>, <sequenceNo>)
Output:	ErrorCodes.CANNOT_LOCK
Test:	Successful

1.3.8 delete

delete(String documentHash, long line, long userID, int startPosition, int endPosition, long sequenceNo)

Purpose: Delete text at the current position.

Parameters:

- documentHash – The documentHash desired for updating
- line – Line # where to make the update
- userId – The user identity
- startPosition – The start position for this operation
- endPosition – The end position for this operation
- sequenceNo – The state of the user's document

Return Value: Returns the sequence number or an appropriate ErrorCode.

Test case #: 1	
Input:	delete(<valid docHash>, <valid line>, <userId>, <valid startPos>, <valid endPos>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

Test case #: 2	
Input:	delete(<valid docHash>, <valid line>, <userId>, <invalid startPos>, <invalid endPos>, <sequenceNo>)
Output:	ErrorCodes.EDITING_OUTSIDE_BOUNDS
Test:	Successful

Test case #: 3	
Input:	delete(<valid docHash>, <invalid line>, <userId>, <valid startPos>, <valid endPos>, <sequenceNo>)
Output:	ErrorCodes.CANNOT_LOCK
Test:	Successful

Test case #: 4	
Input:	delete(<invalid docHash>, <valid line>, <userId>, <valid startPos>, <valid endPos>, <sequenceNo>)
Output:	ErrorCodes.DOCUMENT_DOES_NOT_EXIST
Test:	Successful

Test case #: 5	
Input:	For testing lock with delete: lockLine(<valid docHash>, line 1, <userid>, <sequenceNo>) delete(<valid docHash>, line 1, <userId>, <valid startPos>, <valid endPos>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

Test case #: 6	
Input:	For testing lock, unlock with delete: lockLine(<valid docHash>, line 1, <userid>, <sequenceNo>) unlockLine(<valid docHash, line 1, <userid>, <sequenceNo>) delete(<valid docHash>, line 1, <userId>, <valid startPos>, <valid endPos>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

Test case #: 7	
Input:	For testing 2 clients: lockLine(<valid docHash>, line 1, user 1, <sequenceNo>) unlockLine(<valid docHash>, line 1, user 1, <sequenceNo>) delete(<valid docHash>, line 1, user 2, <valid startPos>, <valid endPos>, <sequenceNo>)
Output:	sequenceNo+1
Test:	Successful

Test case #: 8	
Input:	For testing 2 clients: lockLine(<valid docHash>, line 1, user 1, <sequenceNo>) delete(<valid docHash>, line 1, user 2, <valid startPos>, <valid endPos>, <valid updatedLine>, <sequenceNo>)
Output:	ErrorCodes.CANNOT_LOCK
Test:	Successful

1.3.9 lockLine

long lockLine(String documentHash, long line, long userID, long sequenceNo)

Purpose: Lock or renew the lock for the line for exclusive editing.

Parameters:

- documentHash – the hashcode representing the open document on the system
- line – the line number
- userID - the userid for the lock
- sequenceNo - the state of the user's document

Return value: Returns a corresponding lineID or an appropriate ErrorCode.

Test case #: 1	
Input:	(<documentNotLoaded hashcode>, <invalid line>, <userID>, <repeated sequenceNo>)
Ouput:	ErrorCodes.DOCUMENT_DOES_NOT_EXIST
Test:	Successful

Test case #: 2	
Input:	For testing invalid lines: (<emptyDocument hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line locking conflicts: (<emptyDocument hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 3	
Input:	For testing invalid lines: (<oneLineDocument hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line locking conflicts: (<oneLineDocument hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 4

Input:	For testing invalid lines: (<twoLineDocument hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line locking conflicts: (<twoLineDocument hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 5	
Input:	For testing invalid lines: (<twoLineDocument_FirstLineEmpty hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line locking conflicts: (<twoLineDocument_FirstLineEmpty hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 6	
Input:	For testing invalid lines: (<twoLineDocument_BothEmpty hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line locking conflicts: (<twoLineDocument_BothEmpty hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 7	
Input:	For testing invalid lines: (<threeLineDocument_ThirdLineEmpty hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line locking conflicts: (<threeLineDocument_ThirdLineEmpty hashcode >, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 8

Input:	For testing invalid lines: (<threeLineDocument_AllEmpty hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line locking conflicts: (<threeLineDocument_AllEmpty hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 9	
Input:	For testing invalid lines: (<largeDocument hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line locking conflicts: (<largeDocument hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Requirements to be tested

The lockLine function must be able to handle any type of line on any given document. If the line is valid and the appropriate conditions exist for the line to be unlocked, the locking process occurs and the line is returned. If the line is invalid or a line locking error occurred due to a conflict or unsatisfiable conditions, the appropriate error code is returned.

1.3.10 unlockLine

long unlockLine(String documentHash, long line, long userID, long sequenceNo)

Purpose: Release a lock before it times out.

Parameters:

- documentHash – the hashcode representing the open document on the system
- line – the line number
- userID - the userid for the lock
- sequenceNo - the state of the user's document

Return value: Returns a corresponding lineID or an appropriate ErrorCode.

Test case #: 1	
Input:	(<documentNotLoaded hashcode>, <invalid line>, <userID>, <repeated sequenceNo>)
Ouput:	ErrorCodes.DOCUMENT_DOES_NOT_EXIST
Test:	Successful

Test case #: 2	
Input:	For testing invalid lines: (<emptyDocument hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line unlocking conflicts: (<emptyDocument hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 3	
Input:	For testing invalid lines: (<oneLineDocument hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line unlocking conflicts: (<oneLineDocument hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 4	
Input:	For testing invalid lines: (<twoLineDocument hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line unlocking conflicts: (<twoLineDocument hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 5	
Input:	For testing invalid lines: (<twoLineDocument_FirstLineEmpty hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line unlocking conflicts: (<twoLineDocument_FirstLineEmpty hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 6	
Input:	For testing invalid lines: (<twoLineDocument_BothEmpty hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line unlocking conflicts: (<twoLineDocument_BothEmpty hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 7	
Input:	For testing invalid lines: (<threeLineDocument_ThirdLineEmpty hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line unlocking conflicts: (<threeLineDocument_ThirdLineEmpty hashcode >, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 8	
Input:	For testing invalid lines: (<threeLineDocument_AllEmpty hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line unlocking conflicts: (<threeLineDocument_AllEmpty hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Test case #: 9	
Input:	For testing invalid lines: (<largeDocument hashcode>, <invalid line>, <userID>, <repeated sequenceNo>) For testing line unlocking conflicts: (<largeDocument hashcode>, <line>, <userID>, <repeated sequenceNo>)
Ouput:	line or appropriate ErrorCode
Test:	Successful

Requirements to be tested

The unLockLine function must be able to handle any type of line on any given document. If the line is valid and the appropriate conditions exist for the line to be unlocked, the unlocking process occurs and the line is returned. If the line is invalid or a line unlocking error occurred due to a conflict or unsatisfiable conditions, the appropriate error code is returned.

1.3.11 getLocks

String getLocks(String documentHash)

Purpose: Get a list of locks on the document.

Parameters:

- documentHash - representing the documentHash desired for opening

Return value: A string composed of "linenumber\n" for each line that is locked.

Test case #: 1	
Input:	(<documentNotLoaded hashcode>)
Ouput:	empty string
Test:	Successful

Test case #: 2	
Input:	(<emptyDocument hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber\n”
Test:	Successful

Test case #: 3	
Input:	(<oneLineDocument hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber\n”
Test:	Successful

Test case #: 4	
Input:	(<twoLineDocument hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber\n”
Test:	Successful

Test case #: 5	
Input:	(<twoLineDocument_FirstLineEmpty hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber\n”
Test:	Successful

Test case #: 6	
Input:	(<twoLineDocument_BothEmpty hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber\n”
Test:	Successful

Test case #: 7	
Input:	(<threeLineDocument_ThirdLineEmpty hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber\n”

Test case #: 7	
Input:	(<threeLineDocument_ThirdLineEmpty hashcode>)
Test:	Successful

Test case #: 8	
Input:	(<threeLineDocument_AllEmpty hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber\n”
Test:	Successful

Test case #: 9	
Input:	(<largeDocument hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber\n”
Test:	Successful

Requirements to be tested

The getLocks function must be able to accurately retrieve the required list of locks in the valid format for any locked line on any given document.

1.3.12 getLocksAndUIDs

String getLocksAndUIDs(String documentHash)

Purpose: Get a list of locks on the document with their respective UIDs who have locked them.

Parameters:

- documentHash - representing the documentHash desired for opening

Return value: A string composed of "linenumber,userid\n" for each line that is locked.

Test case #: 1	
Input:	(<documentNotLoaded hashcode>)
Ouput:	empty string
Test:	Successful

Test case #: 2	
Input:	(<emptyDocument hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber,userid\n”
Test:	Successful

Test case #: 3	
Input:	(<oneLineDocument hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber,userid\n”
Test:	Successful

Test case #: 4	
Input:	(<twoLineDocument hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber,userid\n”
Test:	Successful

Test case #: 5	
Input:	(<twoLineDocument_FirstLineEmpty hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber,userid\n”
Test:	Successful

Test case #: 6	
Input:	(<twoLineDocument_BothEmpty hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber,userid\n”
Test:	Successful

Test case #: 7	
Input:	(<threeLineDocument_ThirdLineEmpty hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber,userid\n”
Test:	Successful

Test case #: 8	
Input:	(<threeLineDocument_AllEmpty hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber,userid\n”
Test:	Successful

Test case #: 9	
Input:	(<largeDocument hashcode>)
Ouput:	Various strings from an empty string to strings composed of “linenumber,userid\n”
Test:	Successful

Requirements to be tested

The getLocksAndUIDs function must be able to accurately retrieve the required list of locks and respective user IDs in the valid format for any locked line on any given document.

1.3.13 getSequenceNo

long getSequenceNo(String documentHash, long sequenceNoOld, long lineID)

Purpose: Fetch the current sequence number.

Return value: Return the latest sequence number or an appropriate error code.

Test case #: 1 (All operations against unloaded document)	
Input:	(<documentNotLoaded hashcode>, <oldSequenceNo>, <line>)
Ouput:	ErrorCodes.DOCUMENT_DOES_NOT_EXIST
Test:	Successful

Test case #: 2 (lockLine)	
Input:	(<document hashcode>, <sequenceNo>, <line>)
Ouput:	same sequenceNo
Test:	Successful

Test case #: 3 (unlockLine)	
Input:	(<document hashcode>, <sequenceNo>, <line>)
Ouput:	same sequenceNo
Test:	Successful

Test case #: 4 (addLine)	
Input:	(<document hashcode>, <oldSequenceNo>, <line>)
Ouput:	incremented sequenceNos
Test:	Successful

Test case #: 5 (Insert)	
Input:	(<document hashcode>, <oldSequenceNo>, <line>)
Ouput:	incremented sequenceNos
Test:	Successful

Test case #: 6 (deleteLine)	
Input:	(<document hashcode>, <oldSequenceNo>, <line>)
Ouput:	incremented sequenceNos
Test:	Successful

Test case #: 7 (updateLine)	
Input:	(<document hashcode>, <oldSequenceNo>, <line>)
Ouput:	incremented sequenceNos
Test:	Successful

Requirements to be tested

The getSequenceNo function must be able to correctly increment sequence numbers for a line when an operation does a transaction on a line. The function should not increment the sequence number if a line is only being locked or unlocked.

1.3.14 addLine

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

Test case #: 2	
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		
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
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: From loadDocument() line: 6 userid: 123 toadd: “” sequenceNo: 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: From loadDocument() userid: 15 toadd: 123 Adding more than 1 line\nThis is the 2nd line\n3rd line is here\n sequenceNo: 6
Expected Output:	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: notExist userid: 15 toadd: 123 Adding more than 1 line\nThis is the 2nd line\n3rd line is here\n sequenceNo: 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

Test case #: 9	
Input:	All the above 8 test cases were performed sequentially
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"

1.3.15 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: From loadDocument() line: 1 userid: 123 sequenceNo: 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	
Input:	documentHash: From loadDocument() line: 6 userid: 123 sequenceNo: 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: From loadDocument() line: 27 userid: 123 sequenceNo: 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: From loadDocument() line: 6 userid: 123 sequenceNo: 3
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: From loadDocument() line: 25 userid: 123 sequenceNo: 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: From loadDocument() line: 30 userid: 123 sequenceNo: 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"

1.3.16 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

Test case #: 1	
Input:	documentHash: Not_loaded line: 1 sequenceNo: 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: From loadDocument() line: 1 sequenceNo: 0
Expected Output:	This is a test\n
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: From loadDocument() line: 2 sequenceNo: 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: From loadDocument() line: 6 sequenceNo: 0
Expected Output:	\n
Actual Output:	\n

Test case #: 5
Function: String getLine(String documentHash, long line, long sequenceNo)
Purpose: Boundary Test to see if a non-empty line, found around the middle of the document, can be obtained

Test case #: 5	
Input:	documentHash: From loadDocument() line: 10 sequenceNo: 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: From loadDocument() line: 1 sequenceNo: 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: From loadDocument() line: 1 sequenceNo: 0
Expected Output:	abc123^&*(\n
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: From loadDocument() line: 2 sequenceNo: 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

Test case #: 9	
Input:	documentHash: From loadDocument() line: 6 sequenceNo: 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: From loadDocument() line: 31 sequenceNo: 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

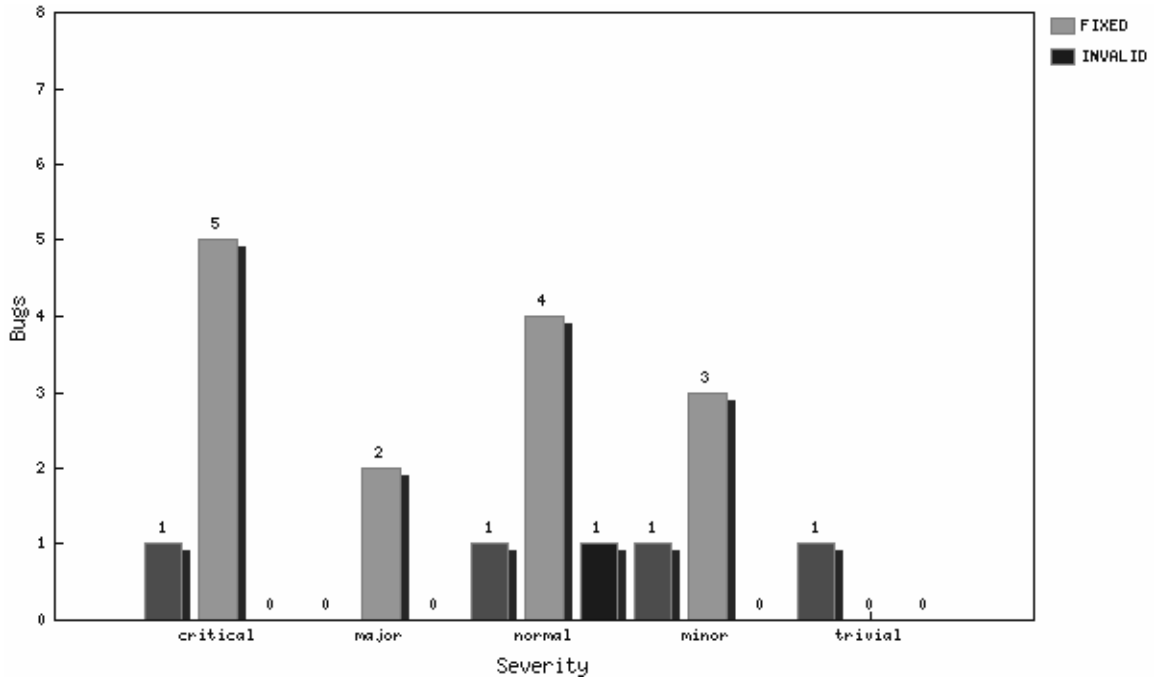
1.4 Bug Reports

1.4.1 Bug Report

ID	Severity	Priority	Plt	Assignee	Status	Resolution	Summary
1	critical	P2	PC	ddurdle@gmail.com	ASSI		Incomplete update()
2	major	P2	PC	ddurdle@gmail.com	RESO	FIXED	JUnit Tests Fail with message "java.sql.SQLException: ERR...
3	normal	P2	PC	ddurdle@gmail.com	RESO	FIXED	while running JUnit tests: org.postgresql.util.PSQLExcept...
4	normal	P2	PC	ddurdle@gmail.com	RESO	FIXED	AddLine: Wrong return value
5	normal	P2	PC	ddurdle@gmail.com	RESO	FIXED	Bugzilla not e-mailing users
6	normal	P2	PC	gvlvl@hotmail.com	RESO	INVALID	JUnit Testing Fails on test_insert
7	minimal	P2	PC	ddurdle@gmail.com	RESO	FIXED	Testing a Document that doesnot exist
8	critical	P1	PC	gvlvl@hotmail.com	RESO	FIXED	Document cannot end in many new lines
9	critical	P1	PC	gvlvl@hotmail.com	RESO	FIXED	Cannot consecutively delete the 1st line of document
10	critical	P1	PC	ddurdle@gmail.com	RESO	FIXED	Loading of document consisting of some empty lines
11	trivial	P2	PC	ddurdle@gmail.com	ASSI		Comment preconditions fell victim to copy/paste error
12	minimal	P2	PC	ddurdle@gmail.com	ASSI		Generic Exception Capture == Bad
13	minimal	P2	PC	ddurdle@gmail.com	RESO	FIXED	Calling LockLine twice
14	minimal	P2	PC	ddurdle@gmail.com	RESO	FIXED	Error msg for delete()
15	major	P1	PC	gvlvl@hotmail.com	RESO	FIXED	SQL Exception when 'getLine()' on a non-loaded document
16	normal	P2	PC	ddurdle@gmail.com	RESO	FIXED	insert() similar error msg to delete()
17	critical	P1	PC	gvlvl@hotmail.com	RESO	FIXED	getLine doesn't return everything that's on the line..may...
18	critical	P2	PC	ddurdle@gmail.com	RESO	FIXED	getHashCode does not handle nulls

19	normal	P2	PC	ddurdle@gmail.com	ASSI		File uploads incur false dependencies on existing docs / ...

1.4.2 Severity versus Resolution



1.4.3 Status versus Resolution

		Status		
		ASSIGNED	RESOLVED	Total
Resolution		4	.	4
	FIXED	.	14	14
	INVALID	.	1	1
	Total	4	15	**19**

1.5 Integration Test Plan

1.5.1 GENERAL INFORMATION

1.5.1.1 Purpose

The purpose of this test plan is to verify the runtime correctness of OnmiEditor server component along with the Web Service interface exposed to be used by OmniEditor client. This document highlights the integration test plan conducted to verify the above along with non-functional requirement checklist for clients.

1.5.1.2 Scope

The scope of the integration test plan includes the following components:

- OmniEditor Web-Service Component
- Postgres Backend Database
- Web-Service client interface

1.5.1.3 System Overview

The Omni-Editor web-service component is a central repository of shared document resources. Client editors interact with this system to enable collaboration between users using heterogeneous text editors on heterogeneous platforms. This document presents the integration test plan that was employed for the system.

- Operational status
  Operational / Under development

1.5.1.4 Project References

The integration test plan is inspired from the following documents.

- *Phase A User Requirement Analysis*
- *Phase A Project Test Plan*

1.5.1.5 Acronyms and Abbreviations

Nirvana – Project name for complete OmniEditor project.
Integration Test Plan – Integration test plan for server component of OmniEditor.

1.5.1.6 Points of Contact

Information

Following are the points of organizational contact (POCs) that may be needed by the document user for informational and troubleshooting purposes.

Gilles Lising
Project Manager
gvlvl@hotmail.com

Ashish Kaila
Test Team lead
416-247-4756
ashishkaila@hotmail.com

Sandy Lam
Quality Assurance Specialist – Project Nirvana
sanlwy@hotmail.com

Dylan Durdle
Maintenance Lead
99durdle@utsc.utoronto.ca

Mark Maglaque
Maintenance Assistant
mark_rampage@hotmail.com

Yani Ioannou
Maintenance Assistant / IT Lead
yani.ioannou@gmail.com

Coordination
For the purpose of integration test plan, team Nirvana has a coordination plan between the project manager, QA and maintenance personals. The plan consists of reviews of outstanding functionality bugs and their addressing. Moreover details about specific support functions (e.g., installation coordination, security, etc.) will be addressed in the coordination meetings. Following is the tentative schedule of the meetings (excluding online group conversations):

November 11, 2004 4:00 – 6:00 pm SF-1101
November 18, 2004 4:00 – 6:00 pm SF-1101
November 25, 2004 4:00 – 6:00 pm SF-1101

1.5.2 Test Definition

For the purpose of integration test following categories of test cases were performed:

- Requirements Validation – Functionality exposed by the OmniEditor fulfill the requirements of the users.
- Functional Testing – Tests to validate functions perform as required.
- Performance/Volume/Stress Testing – Tests to ensure service performs well under stress and high volume of requests.
- Security Testing – Tests to ensure security against buffer overflows and SQL injections that could prove to be crucial security holes.
- Ease of Use – Recommendations and viewpoints of third party taken into consideration while designing service interfaces and input/output to/from functions.
- Operational Testing – JUnit tests / thin client developed to simulate and test operational capabilities from clients.
- Interface Testing – heterogeneous clients from on a variety of platforms used to test web-service interface.

Integration testing stressed heavily on web-service interface definition to ensure data types used are interoperable across the platforms. More specifically we used both Java and .NET clients to test the interface functionality with native data types from both platforms.

1.5.2.1 Input Test Data Control

Input data are controlled to test the system with a minimum number of data types and values. The function was tested with a range of bona fide data types and values that test for overload, saturation, and other "worst case" effects and was tested with bogus data and values that test for rejection of irregular input. Also heavy sized inputs and string inputs with sql injections were tested on the interfaces.

1.5.2.2 Output Test Data Control

For the purpose of test, the backend data store was on a high performance test server and the interface test cases were tested from a variety of front ends (java / .net) imagining moderate client configuration. The output data are analyzed to detect whether an output is produced, output is evaluated as a basis for continuation of the test sequence and against the required output to assess system performance.

1.5.2.3 Data Recovery

The Postgres backend RDBMS ensures data integrity and reliability. Moreover RDBMS is configured to optimize data retrieval and CRUD operations. Thus Nirvana does not reinvent the wheel, rather delegates the data management responsibility to Postgres. Data retrieval during the tests were done both by the SaveDocument feature of the server and at times using a SQL Client to verify both versions.

1.5.2.4 Support Software

- Eclipse 3.0
- Microsoft Visual Studio.NET 2003
- Microsoft Internet Explorer 6.0

1.5.3 Test Execution

1.5.2.1 Test Schedule

JUnit Test suite is executed on daily basis and after every maintenance task. This test suite contains all the test cases mentioned above and also other primitive functions that form the execution basis for others. After test suite execution an automated report is generated that is then circulated among the team members and published on the team's website accessible to client groups.

1.5.2.2 Test Progression

JUnit tests have internal dependencies within each test case that simulates activity progression and real time request cycles Include an explanation concerning the manner in which progression is made from one test to another so the cycle or activity for each test is completely performed. This workflow based testing with interactive clients forms a perfect basis to ensure high performance and system durability.

1.5.2.3 Test Criteria

Tolerance

Maximum ranges of data types from various platforms used to test system tolerance. In case of network packet loss, tests were re-run to ensure system remains tolerant irrespective of network glitches.

Samples

Boundary conditions for data types included minimum and maximum values and special values (e.g. Double.NaN). Objects have boundary conditions as null values and also various states they can possibly acquire.

System Breaks

System was tested against interrupts, halts, and other system breaks which may occur because of non-test conditions and high network activity. Recovery mechanism was tested thoroughly via stress simulations. Also relative and absolute failure statistics were recorded via these simulations to determine relative uptime.

Test Control

Tests conducted were controlled by manual, semiautomatic, or automatic means. Randomized inputs were also derived from simulations for automated test purposes. Moreover an ASP.NET web application was used for performing interactive service tests to test asynchronous functionality.

2 Non Functional

2.1 Measuring Correctness and Reliability

Most of the real issues that need to be tackled, such as concurrency, caching and persistent data are all handled by the fact that our back end is a database. This avoid us writing faulty code trying to manage isolated documents in memory or on disc. Why reinvent the wheel, and possibly do a lousy job doing that, when someone else has already perfected the wheel? That is the premise behind our correctness and reliability.

Because it was not possible to do the analysis on the entire JUnit bucket due to a limitation in the software, so we used the following test scenario that touches upon all the functionality of the server.

We have an automated JUnit sanity check every 15 minutes. IBM does something like this too (there is a poster called "The build break hall of shame").

It has been setup on a machine (yani.wojjie.com) such that runs every hour:

* CVS on my machine is rsynced with seawolf.cdf.utoronto.ca (this is done already for viewcvs)
* Javadoc is run on the source code in CVS (this already happens also via an ant build)
* JUnit Tests are run and HTML reports generated
* If JUnit testing fails the sanity check the group is e-mailed

The correctness of operations is verified via a data type long that is defined as most function's return type?

1) The updated sequence number reflecting the latest transaction. Sequences numbers are only returned from functions that make a change to the document (insert,delete,add,etc). Locking/unlocking lines are not considered as changes in the document and, therefore, no sequence numbers are assignment to these methods.
2) If the operation failed, an error code is returned instead of the Sequence number. You can get these codes from ErrorCodes class. You can determine if an error occurred vs a new sequence number by the following conditional: if (response< ErrorCodes.SUCCESS).
3) Lock/unlock methods return the locked lineID (ID != number) or an appropriate error code.

Sequence numbers help the client to interact with the server to determine what transactions have occurred since the client's last transaction.

OmniWebServer.java in OmniWebServer is the Java Application version of our web service. It comes with a default test driver. It is considered the most up to date version.

OmniServerTest.java in OmniServerTest_dev is an example Java Application that talks to the Java Web Service. You might find this useful for developing a Eclipse plugin from or from writing some test cases.

OmniServerTestSuite.java in OmniWebServerJUnit is the JUnit test program for testing. It involves loading the OmniWebServer.java and not the Java Web Service.

As a group, we have spent a lot of time developing the OmniServer. As we proceeded at each state, we wrote a test driver that would run the methods under development. Once an operation was complete, the testers were given the method and it's documentation and asked to write JUnit scripts for them.

Faults and failures were found and isolated at the testing stage. When the tester wrote the test case and discovered behaviour differences, he or she opened a bug against the lead developer. The lead developer then investigated the method and test case. Most of the errors that were found, as shown by our bug report, are a result of improper usage of the function. When this occurred, we appraised the tester's intent with the goal of the method to determine if the precondition/postcondition of the method should be adapted to anticipate this type of request. This occurred several times for functions such as lockLine(), unLockLine(), and loadDocument(). Testers and developers worked together to refine the test cases and methods.

Some faults in logic were also later discovered in the database scheme. We had studied ACID and MVCC concepts (concurrency features) that Postgres offers. Mistakenly, they were not implemented correctly during the first round of testing. For example, we assume that when we create and populate a table in a database stored procedure transaction, we could avoid another user from performing the operation in parallel. We found that this behaviour is not the default behaviour of Postgres, and parallel transactions would become interleaved together. As a result, we researched into Postgres and discovered that locking the table with exclusive access would implement the desired behaviour. With this action, we also have to now be extra careful with the transaction processing. No exceptions can be thrown during a locked transaction, otherwise the table remains locked until the database is stopped. We discovered this was the case and found it difficult to stop and rollback the offending transaction. We have since fool-proofed the transactions that lock tables or rows to avoid this behaviour.

2.2 Measuring Performance/Scalability and Complexity

System performance is not affected by the number of documents open or the number of concurrent users. Each open document is isolated from each other and managed separately. Therefore, integrity and performance is not compromised. Expiring line locks accompany every action. Document locks are used for operations that need to occur in isolation (i.e. initial document upload).

SSL allows communication between the client to the Java Web Service and the Java Web Service to the Backend Service can all SSL enabled. A non-SSL solution is also available. Isolated backend service for each group will allow each group to receive their own running backend service. User IDs allow only valid assigned users will be able to use the service.

Because it was not possible to do the analysis on the entire JUnit bucket due to a limitation in the software, so we used the following test scenario that touches upon all the functionality of the server.

```
/**
 * Test Driver
 *
```

```
 */
public static  void main(String args[]) throws Exception{

	if (Configuration.DEBUG){

		//load a big document 400-500 lines
		String documentHash;
		OmniServer_dev service = new OmniWebService_dev();
		String document = readTXTFile("file.txt");
	System.out.println(document.length());

	//load document or close and reload
		documentHash = service.loadDocument("file.txt",document);
		if (documentHash == null){
			documentHash = service.getDocument(document);
			//close the document if it already exists
			System.out.println(service.closeDocument(documentHash));
			documentHash = service.loadDocument("file.txt",document);
			System.out.println("closed and reopened: " + documentHash);
		}
		System.out.println("hash: " + documentHash);

		//lock some lines
		System.out.println(service.lockLine(documentHash,22,22,1));
		System.out.println(service.lockLine(documentHash,22,22,1));

		//update
		service.updateLine(documentHash,4,22,"updated line\n",1);

		//insert
		service.insert(documentHash,4,22,3,5,"xxxxx",2);

		//delete
		service.delete(documentHash,14,22,0,2,3);

		//add
		service.addLine(documentHash,8,22,"updated liner",4);

		//deleteline
		service.deleteLine(documentHash,9,22,5);

		//get updates
		service.updates(documentHash,1);

		//get a list of locks
		System.out.println(service.getLocksAndUIDs(documentHash));

		//insert
		service.insert(documentHash,10,22,3,5,"xxxxx",6);

		//fetch document
		System.out.println(service.saveDocument(documentHash));

		//browse system
		System.out.println(service.browse());
```

```
        }
    }
```

There were 16 ***direct*** operations called. As the Structure of Server indicated, insert, delete, amongst other operations call other operations to get the task. This is a very good example of a brief but exhausting test.

Time

seawolf:/h/u3/c2/01/c2durdle/jakarta-tomcat-4.1.31/webapps/axis/c408h015% /usr/bin/time java -cp .:pg74.215.jdbc3.jar OmniServer_dev > ! time2

<output too long→ omitted, no errors>

1.09user 0.13system 0:02.81elapsed 43%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (2048major+1589minor)pagefaults 0swaps

Excellent! The responsiveness and smaller number of page faults indicate a very response service. The time is actually being spent making the call necessary to get the work done instead of attempting to manage itself.

Time --verbose

seawolf:/h/u3/c2/01/c2durdle/jakarta-tomcat-4.1.31/webapps/axis/c408h015% /usr/bin/time --verbose java -cp .:pg74.215.jdbc3.jar OmniServer_dev > ! time2

<output too long→ omitted, no errors>

```
Command being timed: "java -cp .:pg74.215.jdbc3.jar OmniServer_dev"
User time (seconds): 1.23
System time (seconds): 0.11
Percent of CPU this job got: 47%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:02.80
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 0
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 2153
Minor (reclaiming a frame) page faults: 1739
Voluntary context switches: 0
Involuntary context switches: 0
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
```

Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

Excellent! The responsiveness and smaller number of page faults indicate a very response service. The time is actually being spent making the call necessary to get the work done instead of attempting to manage itself.

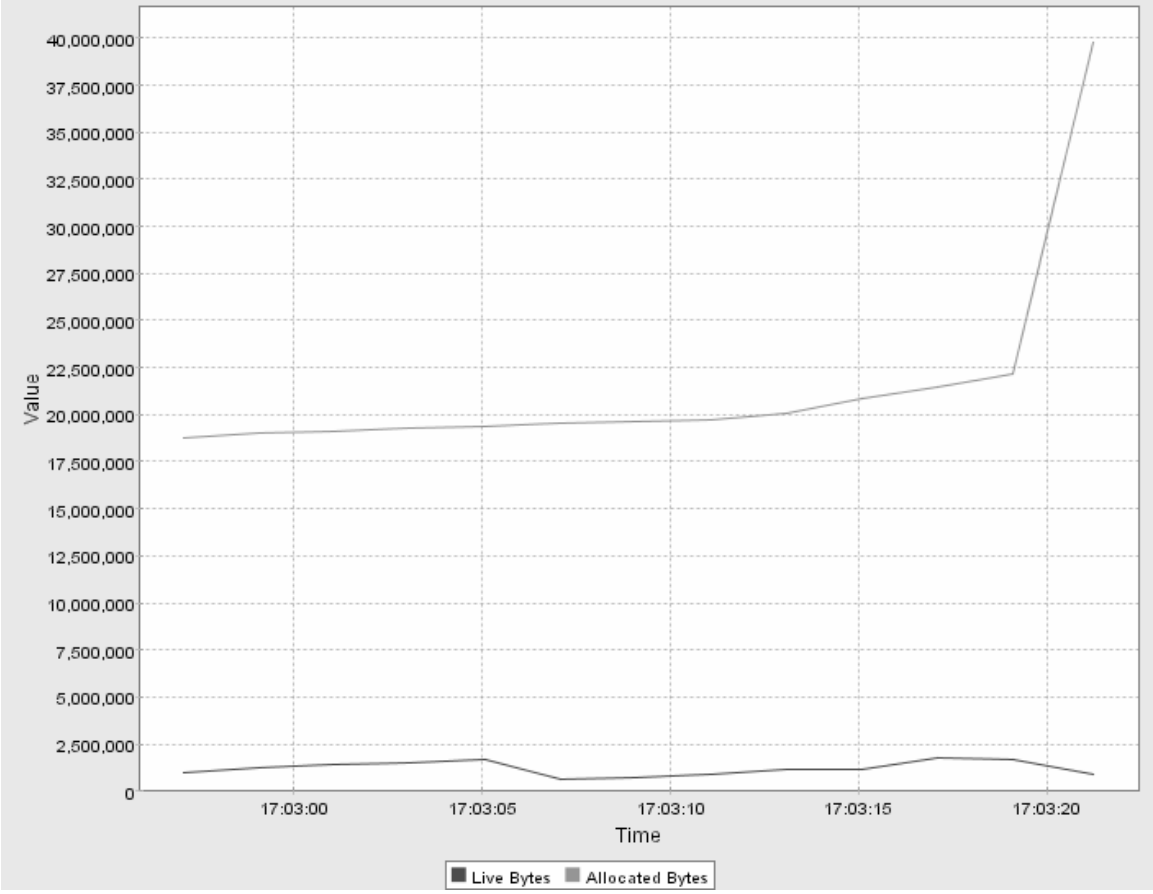
-Prof

count callee caller time
173090 java.lang.String.charAt(I)C
java.util.regex.Pattern$Single.match(Ljava/util/regex/Matcher;ILjava/lang/CharSequence;)Z 0
173090
java.util.regex.Pattern$Single.match(Ljava/util/regex/Matcher;ILjava/lang/CharSequence;)Z
java.util.regex.Pattern$Start.match(Ljava/util/regex/Matcher;ILjava/lang/CharSequence;)Z 1
44991 java.lang.StringBuffer.append(C)Ljava/lang/StringBuffer;
org.postgresql.jdbc1.AbstractJdbc1Statement.replaceProcessing(Ljava/lang/String;)Ljava/lang/Str
ing; 0
44991 java.lang.String.charAt(I)C
org.postgresql.jdbc1.AbstractJdbc1Statement.replaceProcessing(Ljava/lang/String;)Ljava/lang/Str
ing; 0
18149 java.io.BufferedInputStream.ensureOpen()V java.io.BufferedInputStream.read()I 0
14626 java.lang.System.arraycopy(Ljava/lang/Object;ILjava/lang/Object;II)V
java.lang.String.getChars(II[CI)V 0
14216 java.io.BufferedInputStream.read()I org.postgresql.core.PGStream.ReceiveIntegerR(I)I 0
13451 java.lang.String.getChars(II[CI)V
java.lang.StringBuffer.append(Ljava/lang/String;)Ljava/lang/StringBuffer; 0
9270 java.lang.String.charAt(I)C
java.util.Properties.loadConvert(Ljava/lang/String;)Ljava/lang/String; 0
9269 java.lang.StringBuffer.append(C)Ljava/lang/StringBuffer;
java.util.Properties.loadConvert(Ljava/lang/String;)Ljava/lang/String; 0
5852 java.lang.StringBuffer.setShared()V java.lang.String.<init>(Ljava/lang/StringBuffer;)V 0
5852 java.lang.StringBuffer.length()I java.lang.String.<init>(Ljava/lang/StringBuffer;)V 0
5852 java.lang.String.<init>(Ljava/lang/StringBuffer;)V
java.lang.StringBuffer.toString()Ljava/lang/String; 0
5775 java.lang.String.indexOf(II)I java.lang.String.indexOf(I)I 0
5064 java.lang.StringBuffer.<init>(I)V java.lang.StringBuffer.<init>()V 0
4954 java.lang.System.arraycopy(Ljava/lang/Object;ILjava/lang/Object;II)V
java.lang.StringBuffer.expandCapacity(I)V 0
4815 java.lang.StringBuffer.expandCapacity(I)V
java.lang.StringBuffer.append(Ljava/lang/String;)Ljava/lang/StringBuffer; 0
4744 java.lang.String.<init>(II[C)V java.lang.String.substring(II)Ljava/lang/String; 0
…
417 java.lang.ref.Reference.access$200()Ljava/lang/ref/Reference; <unknown caller> 0
417 java.util.ArrayList.get(I)Ljava/lang/Object; java.util.AbstractList$Itr.next()Ljava/lang/Object; 0
416 java.lang.StringBuffer.<init>()V
OmniServer_dev.saveDocument(Ljava/lang/String;)Ljava/lang/String; 0
416 java.lang.StringBuffer.toString()Ljava/lang/String;
OmniServer_dev.saveDocument(Ljava/lang/String;)Ljava/lang/String; 0
415 java.util.SubList$1.nextIndex()I java.util.SubList$1.hasNext()Z 0
415 java.util.AbstractList$Itr.next()Ljava/lang/Object; java.util.SubList$1.next()Ljava/lang/Object; 0

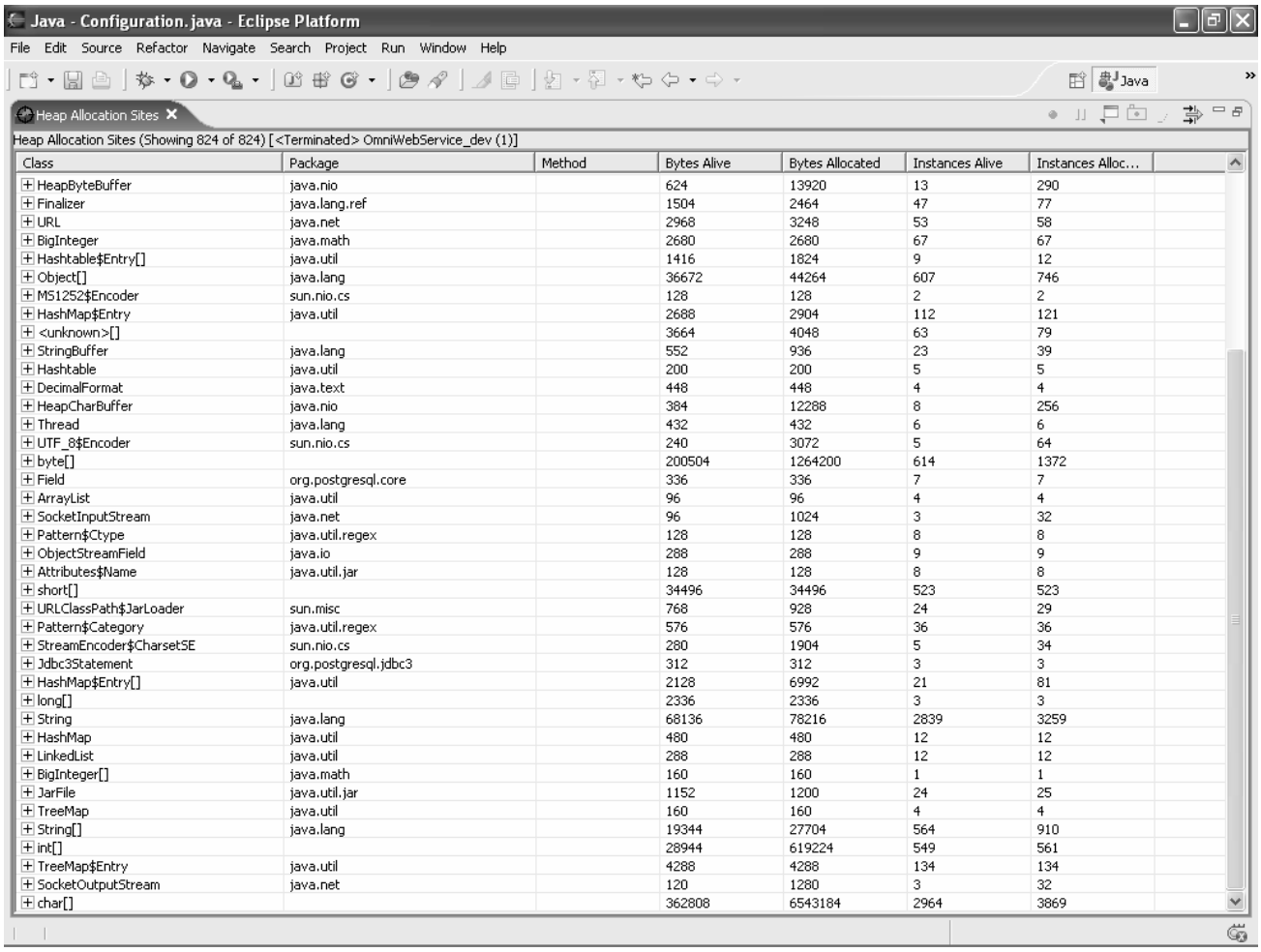
415 java.util.SubList$1.hasNext()Z java.util.SubList$1.next()Ljava/lang/Object; 0
415 org.postgresql.jdbc1.AbstractJdbc1ResultSet.next()Z
OmniServer_dev.saveDocument(Ljava/lang/String;)Ljava/lang/String; 0
415 java.util.regex.Matcher.start()I
java.util.regex.Pattern.split(Ljava/lang/CharSequence;I)[Ljava/lang/String; 0
415 java.util.regex.Matcher.end()I
java.util.regex.Pattern.split(Ljava/lang/CharSequence;I)[Ljava/lang/String; 0
415 java.util.SubList$1.next()Ljava/lang/Object;
java.util.AbstractCollection.toArray([Ljava/lang/Object;)[Ljava/lang/Object; 0
414 org.postgresql.jdbc1.AbstractJdbc1Statement.executeUpdate(Ljava/lang/String;)I
OmniServer_dev.loadDocument(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String; 2
413 java.lang.StringBuffer.append(I)Ljava/lang/StringBuffer;
OmniServer_dev.loadDocument(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String; 0
413 java.io.PrintStream.println(Ljava/lang/String;)V
OmniServer_dev.loadDocument(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String; 0
413 org.postgresql.jdbc1.AbstractJdbc1ResultSet.getString(I)Ljava/lang/String;
OmniServer_dev.saveDocument(Ljava/lang/String;)Ljava/lang/String; 0
413 java.util.zip.ZipFile.getEntry(JLjava/lang/String;)J
java.util.zip.ZipFile.getEntry(Ljava/lang/String;)Ljava/util/zip/ZipEntry; 0
413 java.io.BufferedReader.readLine()Ljava/lang/String;
OmniServer_dev.readTXTFile(Ljava/lang/String;)Ljava/lang/String; 0
412 java.lang.StringBuffer.<init>()V
OmniServer_dev.readTXTFile(Ljava/lang/String;)Ljava/lang/String; 0
412 java.lang.StringBuffer.toString()Ljava/lang/String;
OmniServer_dev.readTXTFile(Ljava/lang/String;)Ljava/lang/String; 0
405 java.lang.ThreadLocal.get()Ljava/lang/Object;
java.lang.Integer.appendTo(ILjava/lang/StringBuffer;)V 0
405 java.lang.StringBuffer.append([CII)Ljava/lang/StringBuffer;
java.lang.Integer.appendTo(ILjava/lang/StringBuffer;)V 0
405 java.lang.Integer.getChars(I[C)I java.lang.Integer.appendTo(ILjava/lang/StringBuffer;)V 0
402 java.lang.StringCoding.decode([BII)[C java.lang.String.<init>([BII)V 0
402 java.lang.String.checkBounds([BII)V java.lang.String.<init>([BII)V 0
…

Due to space constrains, the output has been condensed.

The Heap Summary

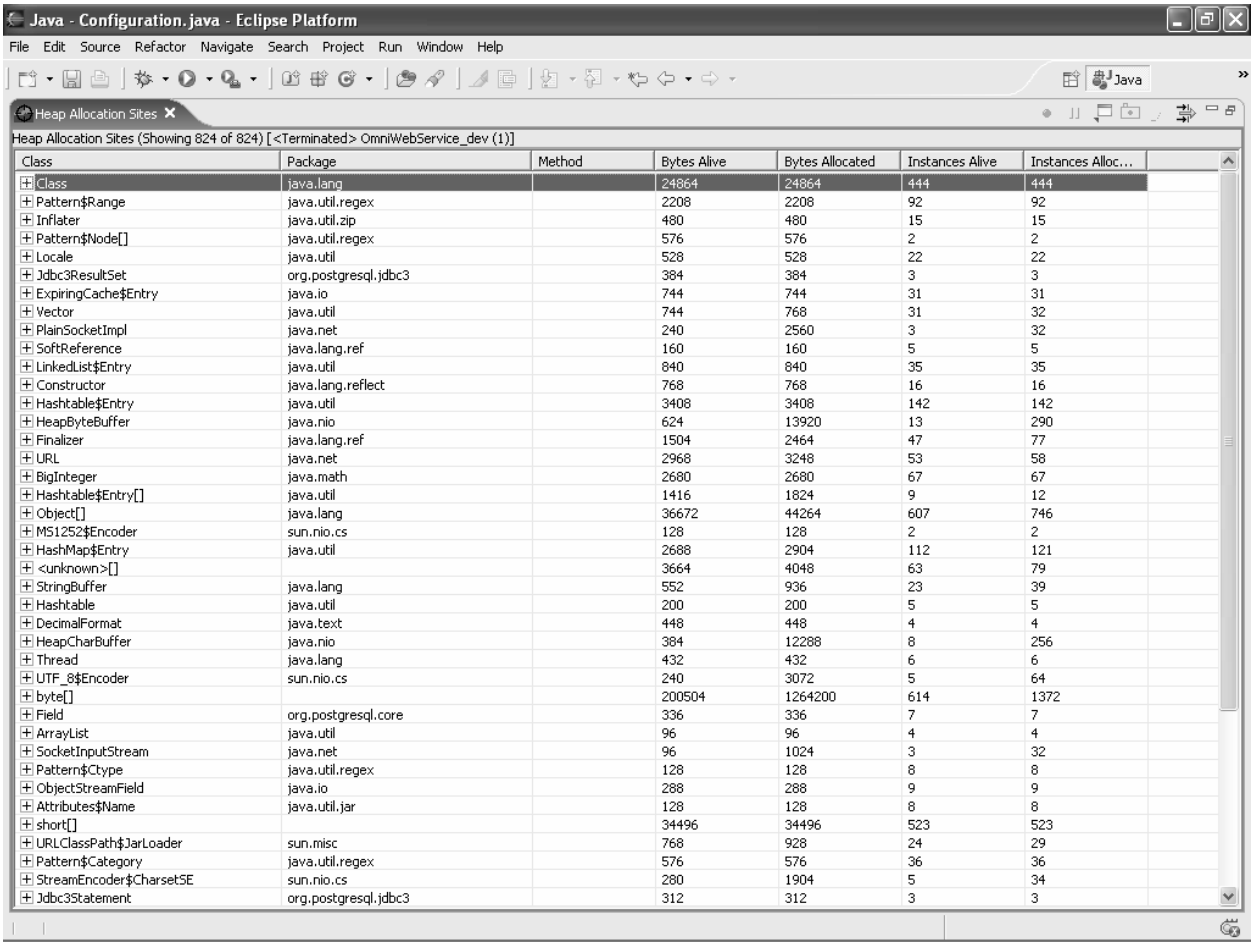

40,000,000
37,500,000
35,000,000
32,500,000
30,000,000
27,500,000
25,000,000
22,500,000
20,000,000
17,500,000
15,000,000
12,500,000
10,000,000
7,500,000
5,000,000
2,500,000
0
Value
17:03:00
17:03:05
17:03:10
17:03:15
17:03:20
Time
Live Bytes
Allocated Bytes

The Heap Allocation Sites



Class	Package	Method	Bytes Alive	Bytes Allocated	Instances Alive	Instances Alloc...
HeapByteBuffer	java.nio		624	13920	13	290
Finalizer	java.lang.ref		1504	2464	47	77
URL	java.net		2968	3248	53	58
BigInteger	java.math		2680	2680	67	67
Hashtable$Entry[]	java.util		1416	1824	9	12
Object[]	java.lang		36672	44264	607	746
MS1252$Encoder	sun.nio.cs		128	128	2	2
HashMap$Entry	java.util		2688	2904	112	121
<unknown>[]			3664	4048	63	79
StringBuffer	java.lang		552	936	23	39
Hashtable	java.util		200	200	5	5
DecimalFormat	java.text		448	448	4	4
HeapCharBuffer	java.nio		384	12288	8	256
Thread	java.lang		432	432	6	6
UTF_8$Encoder	sun.nio.cs		240	3072	5	64
byte[]			200504	1264200	614	1372
Field	org.postgresql.core		336	336	7	7
ArrayList	java.util		96	96	4	4
SocketInputStream	java.net		96	1024	3	32
Pattern$Ctype	java.util.regex		128	128	8	8
ObjectStreamField	java.io		288	288	9	9
Attributes$Name	java.util.jar		128	128	8	8
short[]			34496	34496	523	523
URLClassPath$JarLoader	sun.misc		768	928	24	29
Pattern$Category	java.util.regex		576	576	36	36
StreamEncoder$CharsetSE	sun.nio.cs		280	1904	5	34
Jdbc3Statement	org.postgresql.jdbc3		312	312	3	3
HashMap$Entry[]	java.util		2128	6992	21	81
long[]			2336	2336	3	3
String	java.lang		68136	78216	2839	3259
HashMap	java.util		480	480	12	12
LinkedList	java.util		288	288	12	12
BigInteger[]	java.math		160	160	1	1
JarFile	java.util.jar		1152	1200	24	25
TreeMap	java.util		160	160	4	4
String[]	java.lang		19344	27704	564	910
int[]			28944	619224	549	561
TreeMap$Entry	java.util		4288	4288	134	134
SocketOutputStream	java.net		120	1280	3	32
char[]			362808	6543184	2964	3869

The Heap Allocation Sites (continued)



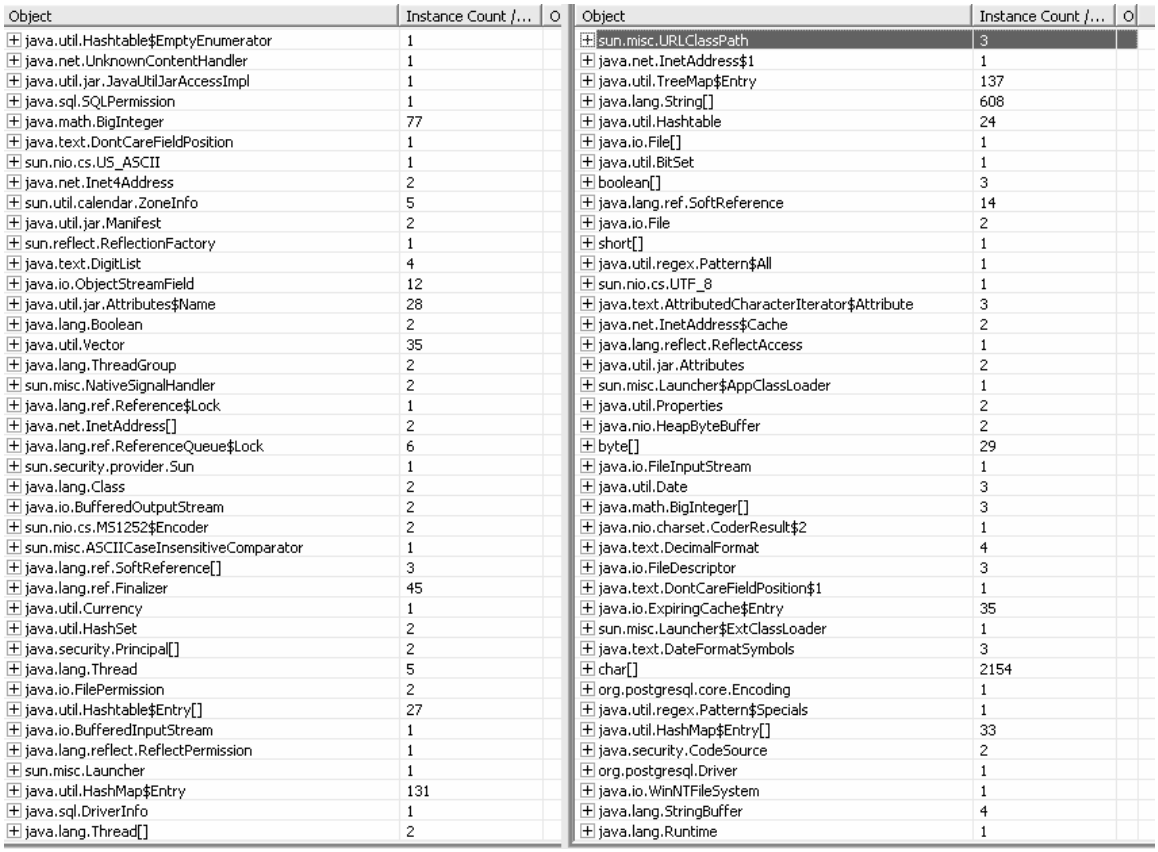
Class	Package	Method	Bytes Alive	Bytes Allocated	Instances Alive	Instances Alloc...
Class	java.lang		24864	24864	444	444
Pattern$Range	java.util.regex		2208	2208	92	92
Inflater	java.util.zip		480	480	15	15
Pattern$Node[]	java.util.regex		576	576	2	2
Locale	java.util		528	528	22	22
Jdbc3ResultSet	org.postgresql.jdbc3		384	384	3	3
ExpiringCache$Entry	java.io		744	744	31	31
Vector	java.util		744	768	31	32
PlainSocketImpl	java.net		240	2560	3	32
SoftReference	java.lang.ref		160	160	5	5
LinkedList$Entry	java.util		840	840	35	35
Constructor	java.lang.reflect		768	768	16	16
Hashtable$Entry	java.util		3408	3408	142	142
HeapByteBuffer	java.nio		624	13920	13	290
Finalizer	java.lang.ref		1504	2464	47	77
URL	java.net		2968	3248	53	58
BigInteger	java.math		2680	2680	67	67
Hashtable$Entry[]	java.util		1416	1824	9	12
Object[]	java.lang		36672	44264	607	746
MS1252$Encoder	sun.nio.cs		128	128	2	2
HashMap$Entry	java.util		2688	2904	112	121
<unknown>[]			3664	4048	63	79
StringBuffer	java.lang		552	936	23	39
Hashtable	java.util		200	200	5	5
DecimalFormat	java.text		448	448	4	4
HeapCharBuffer	java.nio		384	12288	8	256
Thread	java.lang		432	432	6	6
UTF_8$Encoder	sun.nio.cs		240	3072	5	64
byte[]			200504	1264200	614	1372
Field	org.postgresql.core		336	336	7	7
ArrayList	java.util		96	96	4	4
SocketInputStream	java.net		96	1024	3	32
Pattern$Ctype	java.util.regex		128	128	8	8
ObjectStreamField	java.io		288	288	9	9
Attributes$Name	java.util.jar		128	128	8	8
short[]			34496	34496	523	523
URLClassPath$JarLoader	sun.misc		768	928	24	29
Pattern$Category	java.util.regex		576	576	36	36
StreamEncoder$CharsetSE	sun.nio.cs		280	1904	5	34
Jdbc3Statement	org.postgresql.jdbc3		312	312	3	3

The Heap Dump

Object	Instance Count /...
java.security.Permissions	2
java.lang.ref.Finalizer$FinalizerThread	1
java.security.cert.Certificate[]	2
java.lang.Character	10
sun.net.www.protocol.jar.Handler	3
long[]	4
java.util.Hashtable$EmptyIterator	1
java.util.LinkedList	13
sun.net.www.protocol.file.Handler	1
java.lang.String$CaseInsensitiveComparator	1
java.util.LinkedHashMap$Entry	4
java.net.Inet4AddressImpl	1
java.lang.ref.ReferenceQueue$Null	2
java.lang.Long$1	1
java.security.AccessControlContext	4
java.util.GregorianCalendar	3
java.io.PrintStream	2
java.util.ResourceBundle$ResourceCacheKey	7
java.util.regex.Pattern$LastNode	1
sun.misc.SoftCache$ValueCell	6
sun.misc.JarIndex	1
java.util.jar.JarVerifier	1
java.util.TreeMap	4
<unknown>[]	6
sun.nio.cs.StreamEncoder$CharsetSE	2
sun.misc.Unsafe	1
java.nio.charset.CoderResult	2
java.lang.ref.ReferenceQueue	4
sun.misc.SoftCache	3
java.util.Hashtable$Entry	184
java.text.SimpleDateFormat	3
java.util.LinkedHashMap	2
java.lang.ThreadLocal	3
java.net.URL	55
org.postgresql.core.v3.SimpleParameterList	1
java.text.DateFormat$Field[]	2
java.util.zip.Inflater	15
java.lang.ref.WeakReference	2
java.lang.Integer	3
java.util.ArrayList	9

Object	Instance Cou
java.util.regex.Pattern$Node	1
java.util.regex.Pattern$Node[]	2
java.text.FieldPosition[]	1
java.lang.Integer$1	1
java.util.regex.Pattern$Category	36
java.util.LinkedList$Entry	37
sun.nio.cs.StandardCharsets	1
java.io.FilePermissionCollection	2
java.io.ExpiringCache	2
java.nio.ByteOrder	2
org.postgresql.core.v3.ConnectionFactoryImpl	1
java.lang.Package	9
java.security.ProtectionDomain	2
java.util.regex.Pattern$Ctype	8
org.postgresql.core.v3.SimpleQuery	1
java.util.Stack	4
java.nio.charset.CodingErrorAction	3
sun.misc.URLClassPath$JarLoader	25
java.lang.RuntimePermission	2
org.postgresql.core.v2.ConnectionFactoryImpl	1
java.util.regex.Pattern$Range	92
java.lang.Object[]	75
java.lang.String	2222
sun.nio.cs.Surrogate$Parser	2
java.io.OutputStreamWriter	2
java.io.ObjectStreamField[]	6
java.text.DateFormat$Field	18
java.io.BufferedWriter	2
java.util.jar.JarFile$JarFileEntry	2
java.util.HashMap	31
sun.misc.URLClassPath$FileLoader	1
java.text.DecimalFormatSymbols	4
sun.misc.Launcher$Factory	1
java.lang.ThreadGroup[]	1
java.io.ByteArrayOutputStream	1
sun.nio.cs.MS1252	1
java.lang.Terminator$1	1
java.util.Locale	22
java.util.jar.JarFile	25
java.net.InetAddress$CacheEntry	2

The Heap Dump (continued)



Object	Instance Count /...	O
java.util.Hashtable$EmptyEnumerator	1	
java.net.UnknownContentHandler	1	
java.util.jar.JavaUtilJarAccessImpl	1	
java.sql.SQLPermission	1	
java.math.BigInteger	77	
java.text.DontCareFieldPosition	1	
sun.nio.cs.US_ASCII	1	
java.net.Inet4Address	2	
sun.util.calendar.ZoneInfo	5	
java.util.jar.Manifest	2	
sun.reflect.ReflectionFactory	1	
java.text.DigitList	4	
java.io.ObjectStreamField	12	
java.util.jar.Attributes$Name	28	
java.lang.Boolean	2	
java.util.Vector	35	
java.lang.ThreadGroup	2	
sun.misc.NativeSignalHandler	2	
java.lang.ref.Reference$Lock	1	
java.net.InetAddress[]	2	
java.lang.ref.ReferenceQueue$Lock	6	
sun.security.provider.Sun	1	
java.lang.Class	2	
java.io.BufferedOutputStream	2	
sun.nio.cs.MS1252$Encoder	2	
sun.misc.ASCIICaseInsensitiveComparator	1	
java.lang.ref.SoftReference[]	3	
java.lang.ref.Finalizer	45	
java.util.Currency	1	
java.util.HashSet	2	
java.security.Principal[]	2	
java.lang.Thread	5	
java.io.FilePermission	2	
java.util.Hashtable$Entry[]	27	
java.io.BufferedInputStream	1	
java.lang.reflect.ReflectPermission	1	
sun.misc.Launcher	1	
java.util.HashMap$Entry	131	
java.sql.DriverInfo	1	
java.lang.Thread[]	2	

Object	Instance Count /...	O
sun.misc.URLClassPath	3	
java.net.InetAddress$1	1	
java.util.TreeMap$Entry	137	
java.lang.String[]	608	
java.util.Hashtable	24	
java.io.File[]	1	
java.util.BitSet	1	
boolean[]	3	
java.lang.ref.SoftReference	14	
java.io.File	2	
short[]	1	
java.util.regex.Pattern$All	1	
sun.nio.cs.UTF_8	1	
java.text.AttributedCharacterIterator$Attribute	3	
java.net.InetAddress$Cache	2	
java.lang.reflect.ReflectAccess	1	
java.util.jar.Attributes	2	
sun.misc.Launcher$AppClassLoader	1	
java.util.Properties	2	
java.nio.HeapByteBuffer	2	
byte[]	29	
java.io.FileInputStream	1	
java.util.Date	3	
java.math.BigInteger[]	3	
java.nio.charset.CoderResult$2	1	
java.text.DecimalFormat	4	
java.io.FileDescriptor	3	
java.text.DontCareFieldPosition$1	1	
java.io.ExpiringCache$Entry	35	
sun.misc.Launcher$ExtClassLoader	1	
java.text.DateFormatSymbols	3	
char[]	2154	
org.postgresql.core.Encoding	1	
java.util.regex.Pattern$Specials	1	
java.util.HashMap$Entry[]	33	
java.security.CodeSource	2	
org.postgresql.Driver	1	
java.io.WinNTFileSystem	1	
java.lang.StringBuffer	4	
java.lang.Runtime	1	

Halstead Complexity

File, length, volume, level, effort, time
/h/u3/c2/01/c2durdle/jakarta-tomcat-4.1.31/webapps/axis/c408h015/EchoHeaders.java
336 2073 0.042416 48876 0.3
/h/u3/c2/01/c2durdle/jakarta-tomcat-4.1.31/webapps/axis/c408h015/ErrorCodes.java
38 155 0.405797 383 0.0
/h/u3/c2/01/c2durdle/jakarta-tomcat-4.1.31/webapps/axis/c408h015/OmniServer_dev.java
6139 54661 0.008242 6632335 36.8

This computes to the following:

N1: operators, unique n1
N2: operands, unique n2

Length = N1 + N2 = 6513

Volume = (N1 + N2) $\log_2$ (n1 + n2) = 56889
Level = (2/n1)*(n2/N2) = 0.456455
Efforts = Volume / Level = 6681594
Time = Efforts / 180000 = 37.1 person months

This indicates fairly-not-too-complicated code.

Mccabe Complexity

File Name Complexity No. of returns
--- --- --- ---
/h/u3/c2/01/c2durdle/jakarta-tomcat-4.1.31/webapps/axis/c408h015/EchoHeaders.java ***
3 4
/h/u3/c2/01/c2durdle/jakarta-tomcat-4.1.31/webapps/axis/c408h015/ErrorCodes.java ***
1 0
/h/u3/c2/01/c2durdle/jakarta-tomcat-4.1.31/webapps/axis/c408h015/OmniServer_dev.java ***
107 52

Control flow from the above:

|V| + |E| - 2 = (3+1+107) + (4 + 0 + 52) - 2 = 165

This is a high number indicating good test coverage.