OMNIEDITOR PROJECT

PHASE C: CLIENT UNIT TEST PLAN + JUNIT REPORTS

CODENAME: NIRVANA

CSC408F1
Group Members:

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

1 Functional 3
1.1 Scope 3
1.2 Strategy 3
1.2.1 JUnit Test 3
1.3 Unit Test Cases 4
1.3.1 createDocument 4
1.3.2 closeDocument 5
1.3.3 retrieveDocument 5
1.3.5 listDocuments 6
1.3.6 updateLine 7
1.3.9 lockLine 8
1.3.10 releaseLine 11
1.3.12 listLineLocks 13
1.3.14 addLine 15
1.3.15 deleteLine 18

1 Functional

1.1 Scope

The connectors are the blood of the client system. Therefore, in-depth testing is necessary. We prepared a suite of generic test cases that can be run against any connector. These test cases are located under the individual connector. For example, for Nirvana, the test cases are located under
edu.toronto.cdf.werewolf.c408h015.omni.client.core.connectors.nirvana\src\edu\toronto\cdf\were wolf\c408h015\omni\client\core\connectors\nirvana\junit. We did this so that each connector could have only the test cases that would work with that particular connector execute. For example, a connector that did not support line locking could not possibly ever be expected to pass the line locking test cases.

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 *lockRelease* 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

1.3 Unit Test Cases

1.3.1 createDocument

void String createDocument(String documentName, IDocument document)

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

Parameters:

- fileName – Name of the document.
- document – Contents of the document.

Return Value: Returns an exception, if needed.

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

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

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

Test case #: 4	
Input:	(<unique doc name>, <4MB Contents>)
Output:	no exception
Test:	Successful

Test case #: 5	
Input:	(<unique doc name>, <Blank Contents>)
Output:	no exception
Test:	Successful

Requirements to be Tested

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

1.3.2 closeDocument

void closeDocument(String filename)

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

Parameters:

- filename – filename of the document.

Return Value: Returns an exception, if needed.

Test case #: 1	
Input:	(null)
Output:	no exception
Test:	Successful

Test case #: 2	
Input:	(<non existing doc filename>)
Output:	no exception
Test:	Successful

Test case #: 3	
Input:	(<valid doc filename>)
Output:	no exception
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 filename does exist, function should just exit with an appropriate error code.

1.3.3 retrieveDocument

String retrieveDocument(String filename)

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

Parameters:

- filename – filename of the document.

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

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

Test case #: 2	
Input:	(<non existing doc filename>)
Output:	Null
Test:	Successful

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

Requirements to be Tested

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

1.3.5 listDocuments

String[] listDocuments()

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 .

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

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

Test case #: 2	
Procedure:	Loaded a document, browsed, then closed the same document and browsed.
Test:	Successful

Requirements to be Tested

The listDocuments function must return documents names 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

void int updateLine(String filename, int line, String updatedLine)

Purpose: Update line with the replacement line.

Parameters:
- filename – The filename desired for updating
- line – Line # where to make the update
- updatedLine – The line to be added

Return Value: Returns an exception, if needed.

Test case #: 1	
Input:	updateLine(<valid docFilename>, <valid line>, <valid updatedLine>)
Output:	No exception
Test:	Successful

Test case #: 2	
Input:	updateLine(<valid docFilename>, <inValid line>, <valid updatedLine>)
Output:	LineLockException
Test:	Successful

Test case #: 3	
Input:	updateLine(<valid docFilename>, <valid line>, <Invalid updatedLine>)
Output:	LineLockException
Test:	Successful

Test case #: 4	
Input:	updateLine(<Invalid docFilename>, <valid line>, <valid updatedLine>)

Output:	DocumentDoesNotExistException
Test:	Successful

Test case #: 5	
Input:	For testing lock with updateLine: Lock the line first : lockLine(<valid filename>, line 8) updateLine(<Valid docFilename>, line 8, <valid updatedLine>)
Output:	no exception
Test:	Successful

Test case #: 6	
Input:	For testing lock and unlock with updateLine: Lock the line first : lockLine(<valid filename>, line 6) Then, unlock line : releaseLine(<valid filename>, line 6) updateLine(<Invalid docFilename>, line 6, <valid updatedLine>)
Output:	no exception
Test:	Successful

1.3.9 lockLine

void lockLine(String filename, int line)

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

Parameters:
- filename – the filename representing the open document on the system
- line – the line number

Return Value: Returns an exception, if needed.

Test case #: 1	
Input:	(<documentNotLoaded filename>, <invalid line>)
Output: Test:	DocumentDoesNotExistException Successful

Test case #: 2	
Input:	For testing invalid lines: (<emptyDocument filename>, <invalid line>) For testing line locking conflicts: (<emptyDocument filename>, <line>)

Test case #: 2	
Input:	For testing invalid lines: (<emptyDocument filename>, <invalid line>) For testing line locking conflicts: (<emptyDocument filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 3	
Input:	For testing invalid lines: (<oneLineDocument filename>, <invalid line>) For testing line locking conflicts: (<oneLineDocument filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 4	
Input:	For testing invalid lines: (<twoLineDocument filename>, <invalid line>) For testing line locking conflicts: (<twoLineDocument filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 5	
Input:	For testing invalid lines: (<twoLineDocument_FirstLineEmpty filename>, <invalid line>) For testing line locking conflicts: (<twoLineDocument_FirstLineEmpty filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 6	
Input:	For testing invalid lines: (<twoLineDocument_BothEmpty filename>, <invalid line>) For testing line locking conflicts: (<twoLineDocument_BothEmpty filename>, <line>)

Test case #: 6	
Input:	For testing invalid lines: (<twoLineDocument_BothEmpty filename>, <invalid line>) For testing line locking conflicts: (<twoLineDocument_BothEmpty filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 7	
Input:	For testing invalid lines: (<threeLineDocument_ThirdLineEmpty filename>, <invalid line>) For testing line locking conflicts: (<threeLineDocument_ThirdLineEmpty filename >, <line>)
Output:	no exception
Test:	Successful

Test case #: 8	
Input:	For testing invalid lines: (<threeLineDocument_AllEmpty filename>, <invalid line>) For testing line locking conflicts: (<threeLineDocument_AllEmpty filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 9	
Input:	For testing invalid lines: (<largeDocument filename>, <invalid line>) For testing line locking conflicts: (<largeDocument filename>, <line>)
Output:	no exception
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 releaseLine

Void releaseLine(String filename, int line)

Purpose: Release a lock before it times out.

Parameters:
- filename – the filename representing the open document on the system
- line – the line number

Return Value: Returns an exception, if needed.

Test case #: 1	
Input:	(<documentNotLoaded filename>, <invalid line>)
Output:	DocumentDoesNotExistException
Test:	Successful

Test case #: 2	
Input:	For testing invalid lines: (<emptyDocument filename>, <invalid line>) For testing line unlocking conflicts: (<emptyDocument filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 3	
Input:	For testing invalid lines: (<oneLineDocument filename>, <invalid line>) For testing line unlocking conflicts: (<oneLineDocument filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 4	
Input:	For testing invalid lines: (<twoLineDocument filename>, <invalid line>) For testing line unlocking conflicts: (<twoLineDocument filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 5	
Input:	For testing invalid lines: (<twoLineDocument_FirstLineEmpty filename>, <invalid line>) For testing line unlocking conflicts: (<twoLineDocument_FirstLineEmpty filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 6	
Input:	For testing invalid lines: (<twoLineDocument_BothEmpty filename>, <invalid line>) For testing line unlocking conflicts: (<twoLineDocument_BothEmpty filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 7	
Input:	For testing invalid lines: (<threeLineDocument_ThirdLineEmpty filename>, <invalid line>) For testing line unlocking conflicts: (<threeLineDocument_ThirdLineEmpty filename >, <line>)
Output:	no exception
Test:	Successful

Test case #: 8	
Input:	For testing invalid lines: (<threeLineDocument_AllEmpty filename>, <invalid line>) For testing line unlocking conflicts: (<threeLineDocument_AllEmpty filename>, <line>)
Output:	no exception
Test:	Successful

Test case #: 9	
Input:	For testing invalid lines: (<largeDocument filename>, <invalid line>) For testing line unlocking conflicts: (<largeDocument filename>, <line>)
Output:	no exception

Test case #: 9	
Input:	For testing invalid lines: (<largeDocument filename>, <invalid line>) For testing line unlocking conflicts: (<largeDocument filename>, <line>)
Test:	Successful

Requirements to be tested

The releaseLine 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 unclassifiable conditions, the appropriate error code is returned.

1.3.12 listLineLocks

OmniLines[] String listLineLocks(String filename)

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

Parameters:

- filename - representing the filename desired for opening

Return value: An array of OmniLines.

Test case #: 1	
Input:	(<documentNotLoaded filename>)
Output:	empty string
Test:	Successful

Test case #: 2	
Input:	(<emptyDocument filename>)
Output:	An array of OmniLines with appropriate locks.
Test:	Successful

Test case #: 3	
Input:	(<oneLineDocument filename>)
Output:	An array of OmniLines with appropriate locks.
Test:	Successful

Test case #: 4	
Input:	(<twoLineDocument filename>)
Output:	An array of OmniLines with appropriate locks.
Test:	Successful

Test case #: 5	
Input:	(<twoLineDocument_FirstLineEmpty filename>)
Output:	An array of OmniLines with appropriate locks.
Test:	Successful

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

Test case #: 7	
Input:	(<threeLineDocument_ThirdLineEmpty filename>)
Output:	An array of OmniLines with appropriate locks.
Test:	Successful

Test case #: 8	
Input:	(<threeLineDocument_AllEmpty filename>)
Output:	An array of OmniLines with appropriate locks.
Test:	Successful

Test case #: 9	
Input:	(<largeDocument filename>)
Output:	An array of OmniLines with appropriate locks.
Test:	Successful

Requirements to be tested

The listLineLocks 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.

1.3.14 addLine

Test case #: 1		
Function: void addline(int filename, int line, string toad)		
Purpose: Test if addLine can be done when a document is loaded Test if one non-empty line can be added		
Input:	filename: line: toadd:	From retrieveDocument() 1 123 Adding 1st line 0
Expected Output:	1	
Actual Output:	1	

Test case #: 2		
Function: void addline(int filename, int line, string toadd)		
Purpose: Test if more than one line can be added		
Input:	filename: line: toadd:	From retrieveDocument() 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: void addline(int filename, int line, string toadd)		
Purpose: Test if a non-empty line can be added		
Input:	filename: line: toadd:	From retrieveDocument() 1 123 “” 2
Expected Output:	3	
Actual Output:	3	

Test case #: 4
Function: void addline(int filename, int line, string toadd)
Purpose: Test if a combination of various characters can be added in a line

Test case #: 4		
Input:	filename: line: toadd:	From retrieveDocument() 1 123 a b 1 @ ~ JUNK\n oh & > ? <{ |||__-- 3
Expected Output:	4	
Actual Output:	4	

Test case #: 5		
Function: void addline(int filename, int line, string toadd)		
Purpose: Boundary test if a combination of various characters can be added on the 2nd line		
Input:	filename: line: toadd:	From retrieveDocument() 2 123 a b 1 @ ~ JUNK\n oh & > ? <{ |||__-- 4
Expected Output:	5	
Actual Output:	5	

Test case #: 6		
Function: void addline(int filename, int line, string toadd)		
Purpose: Boundary test if an empty line can be added around the middle of the document		
Input:	filename: line: toadd:	From retrieveDocument() 6 123 “” 5
Expected Output:	6	
Actual Output:	6	

Test case #: 7		
Function: void addline(int filename, int line, string toadd)		
Purpose: Boundary test if more than 1 line can be added at the end of the document		
Input:	filename: line: toadd:	From retrieveDocument() 15 123 Adding more than 1 line\nThis is the 2nd line\n3rd line is here\n 6

Test case #: 7	
Expected Output:	7
Actual Output:	7

Test case #: 8	
Function: void addline(int filename, int line, string toadd)	
Purpose: Test if a line can be added when a document is not loaded	
Input:	filename: line: notExist 15 toadd: 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: void addline(int filename, int line, string toadd)	
Purpose: Test if the newly added lines are saved to the document	
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"

Test case #: 9	
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: int deleteLine(String filename, int line)		
Purpose: Boundary Test if the 1st line can be deleted		
Input:	filename: line:	From retrieveDocument() 1 123 0
Expected Output:	1	
Actual Output:	1	

Test case #: 2		
Function: int deleteLine(String filename, int line)		
Purpose: Boundary Test if a non-empty line can be deleted around the middle of the document		
Input:	filename: line:	From retrieveDocument() 6 123 1
Expected Output:	2	
Actual Output:	2	

Test case #: 3

Test case #: 3		
Function: int deleteLine(String filename, int line)		
Purpose: Boundary Test if a non-empty line can be deleted around the end of the document		
Input:	filename: line:	From retrieveDocument() 27 123 2
Expected Output:	3	
Actual Output:	3	

Test case #: 4		
Function: int deleteLine(String filename, int line,)		
Purpose: Boundary Test if an empty line can be deleted around the middle of the document		
Input:	filename: line:	From retrieveDocument() 6 123 3
Expected Output:	4	
Actual Output:	4	

Test case #: 5		
Function: int deleteLine(String filename, int line)		
Purpose: Boundary Test if an empty line can be deleted around the end of the document		
Input:	filename: line:	From retrieveDocument() 25 123 4
Expected Output:	5	
Actual Output:	5	

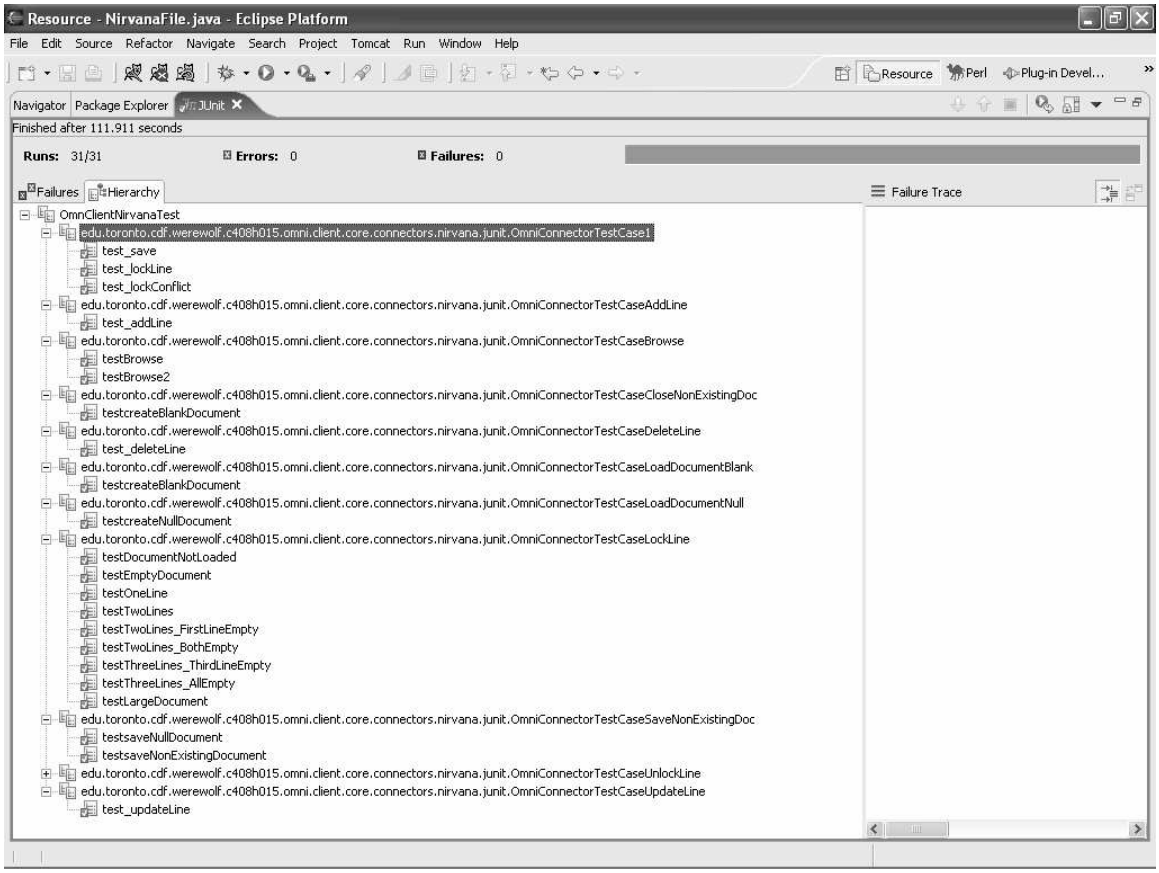
Test case #: 6		
Function: int deleteLine(String filename, int line)		
Purpose: Test if a non existing line can be deleted		
Input:	filename: line:	From retrieveDocument() 30 123 5
Expected Output:	-8	
Actual Output:	-8	

Test case #: 7

Test case #: 7	
Function: int deleteLine(String filename, int line)	
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 #$%, CAPS and 53487" + "\n" + "\n" + "I guess I can put something on this line, right?" + "\n" + "\n" + "10th Line" + "\n" + "\n" + "\n" + "\n" + "\n" + "\n" + "\n" + "Many Many lines" + "\n" + "\n" + "This is int" + "\n" + "This" + "\n" + "is" + "\n" + "for" + "\n" + "a" + "\n" + "greater" + "\n" + "good!" + "\n" + "\n" + "\n" + "\n" + "Last line after all ops" + "\n"
Actual Output:	testDeleteLine +"\n" + "Test cases for deleteLine" + "\n" + "\n" + "A bungch of chars like #$%, 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 int" + "\n" + "This" + "\n" + "is" + "\n" + "for" + "\n" + "a" + "\n" + "greater" + "\n" + "good!" + "\n" + "\n" + "\n" + "\n" + "Last line after all ops" + "\n"

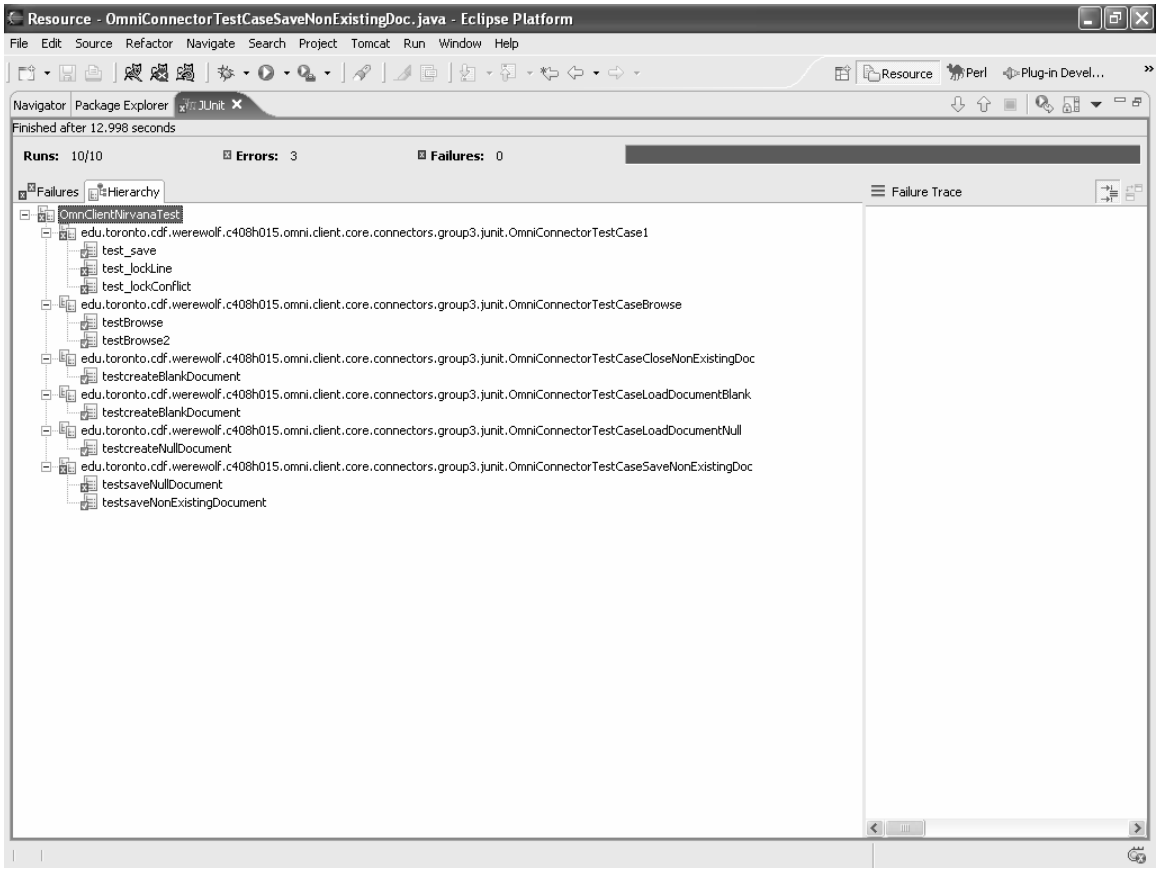
2 JUnit results against Nirvana Connector (implements 100% of connector)

Success=100% (31/31)



3 JUnit results against Nirvana Connector (implements 100% of connector)

Success=70% (7/10)



The TestCase1 fails because it needs line locking to function.

SaveNonExistingDoc fails due to a bug against the Group3 server.