

Using the ooTest Framework to Write ooRexx Test Cases

Quick Start

Version 1.0.0 Edition

Draft - SVN Rev 3060

August 22, 2008



Mark Miesfeld

Using the ooTest Framework to Write ooRexx Test Cases: Quick Start

by

Mark Miesfeld

Version 1.0.0 Edition

Published August 22, 2008

Copyright © 2008 Rexx Language Association. All rights reserved.

This program and the accompanying materials are made available under the terms of the [Common Public License Version 1.0](#).

Before using this information and the product it supports, be sure to read the general information under [Notices](#).

Thanks to Julian Choy for the ooRexx logo design.

Table of Contents

About This Book	v
1. Related Information	v
2. Getting Help	v
2.1. The Rexx Language Association Mailing List	v
2.2. The Open Object Rexx SourceForge Site	v
2.3. comp.lang.rexx Newsgroup	vii
1. First Steps	1
1.1. ooRexxUnit Snapshots	1
1.2. Download	1
1.3. Extract the Files	2
1.4. Test Your Install	2
1.5. Understanding What you See	5
2. Starting to Write a Test Case	7
2.1. A First Test Case	7
2.2. Contributing to the ooRexx Project	7
3. Starting a Test Group from Scratch	11
3.1. Starting a Test Group	11
3.2. Editing STREAM.testGroup	11
3.3. STREAM.testGroup Continued	13
3.4. Finally, the Interesting Part	14
3.5. The Core, Validating Results	15
3.6. The Finale, Executing our Test	15
3.7. One More Quick Test	16
4. Some Reference Documentation	19
4.1. The assertXXX() Methods	19
5. How to Become a Committer	23
6. Negative Tests	25
6.1. Example One	25
6.2. Example Two	26
A. Notices	31
A.1. Trademarks	31
A.2. Source Code For This Document	32
B. Common Public License Version 1.0	33
B.1. Definitions	33
B.2. Grant of Rights	33
B.3. Requirements	34
B.4. Commercial Distribution	34
B.5. No Warranty	35
B.6. Disclaimer of Liability	35
B.7. General	36
Index	37

About This Book

ooTest is a testing framework that facilitates the testing of the ooRexx interpreter. It sits on top of the ooRexxUnit framework which is a generic testing framework that anyone can use to test their Rexx applications.

This PDF attempts to give anyone, even those with only a small understanding of Rexx, a quick start to writing test cases that test the ooRexx interpreter. In particular, it is geared towards the person who says: I would like to help the ooRexx project by writing test cases, but I don't understand that object stuff.

The Quick Start comes from a series of e-mail messages I posted on the ooRexx Developer's list. As such it has a rather informal tone. The basic idea driving this book is that by using a few mechanical techniques along with some boilerplate code, anyone who can write a small piece of Rexx code can write a working test case. Although the ooTest framework is object orientated, you do not need to know objects, or even understand objects, to write test cases.

It is not my intention to fully explain every detail here. My intention is to get some people started writing test cases. I believe that if you use the boilerplate code to start writing test cases, some of the object stuff will just seep in. Because of this, writing test cases is a good way to slowly get used to using objects in your own ooRexx code.

1. Related Information

See also: *Open Object Rexx: Reference*

2. Getting Help

The Open Object Rexx Project has a number of methods to obtain help for ooRexx. These methods, in no particular order of preference, are listed below.

2.1. The Rexx Language Association Mailing List

The *Rexx Language Association* (<http://www.rexxla.org/>) maintains a mailing list for its members. This mailing list is only available to RexxLA members thus you will need to join RexxLA in order to get on the list. The dues for RexxLA membership are small and are charged on a yearly basis. For details on joining RexxLA please refer to the *RexxLA Home Page* (<http://rexxla.org/>) or the *RexxLA Membership Application* (http://rexxla.org/About_RexxLA/member.html) page.

2.2. The Open Object Rexx SourceForge Site

The Open Object Rexx Project (<http://www.oorexx.org/>) utilizes *SourceForge* (<http://sourceforge.net/>) to house the *ooRexx Project* (<http://sourceforge.net/projects/oorexx>) source repositories, mailing lists and other project features. Here is a list of some of the most useful facilities.

The ooRexx Forums

The ooRexx project maintains a set of forums that anyone may contribute to or monitor. They are located on the *ooRexx Forums* (http://sourceforge.net/forum/?group_id=119701) page. There are currently three forums available: Help, Developers and Open Discussion. In addition, you can monitor the forums via email.

The Developer Mailing List

You can subscribe to the oorexx-devel mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is for discussing ooRexx project development activities. It also supports a historical archive of past messages.

The Users Mailing List

You can subscribe to the oorexx-users mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is for discussing using ooRexx. It also supports a historical archive of past messages.

The Announcements Mailing List

You can subscribe to the oorexx-announce mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is only used to announce significant ooRexx project events.

The Bug Mailing List

You can subscribe to the oorexx-bugs mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is only used for monitoring changes to the ooRexx bug tracking system.

Support Requests

You can create a support request at *ooRexx Support Request* (http://sourceforge.net/tracker/?group_id=119701&atid=684731) page. Please be sure to log in to Sourceforge before creating the request so that it will record your e-mail address. This will allow SourceForge (and the ooRexx developers) a way to contact you when updates are made to your request. Otherwise you will need to manually check back on this page to track any updates to the request.

Also, please try to provide as much information in the support request as possible so that the developers can determine the problem as quickly as possible.

Bug Reports

You can create a bug report at *ooRexx Bug Report* (http://sourceforge.net/tracker/?group_id=119701&atid=684730) page. Please be sure to log in to Sourceforge before creating the report so that it will record your e-mail address. This will allow SourceForge (and the ooRexx developers) a way to contact you when updates are made to your report. Otherwise you will need to manually check back on this page to track any updates to the report.

Also, please try to provide as much information in the bug report as possible so that the developers can determine the problem as quickly as possible.

Patch Reports

If you create an enhancement patch for ooRexx please post the patch using the *ooRexx Patch Report* (http://sourceforge.net/tracker/?group_id=119701&atid=684732) page. Please be sure to log in to Sourceforge before creating the report so that it will record your e-mail address. This will allow SourceForge (and the ooRexx developers) a way to contact you when updates are made to your report. Otherwise you will need to manually check back on this page to track any updates to the report.

Also, please try to provide as much information in the patch report as possible so that the developers can evaluate the enhancement as quickly as possible.

Please do not post bug patches here, instead you should open a bug report and attach the patch to it.

2.3. comp.lang.rexx Newsgroup

The comp.lang.rexx (news:comp.lang.rexx) newsgroup is a good place to obtain help from many individuals within the Rexx community. You can obtain help on Open Object Rexx or on any number of other Rexx interpreters and tools.

Chapter 1. First Steps

We'll start with a quick explanation of how to get started and a little bit of an overview.

1.1. ooRexxUnit Snapshots

Download one of the ooRexxUnit snapshots from SourceForge. Through out this discussion I use the term ooRexxUnit and ooTest interchangeably. ooRexxUnit is a generic testing framework and the term first used when talking about testing ooRexx. Some things retain that name. What I am really concerned with is ooTest which is a framework whose sole purpose is to test the ooRexx interpreter.

As the ooRexx interpreter evolves, so does the ooTest framework. For that reason you need to grab a snapshot that matches the version of ooRexx that you will be writing and executing your tests with.

It can not be stressed enough how valuable a contribution to the ooRexx project test cases are. You do not need to write and execute test cases using the most current non-released version of ooRexx. Most people should write and execute test cases using the most recent released version of ooRexx. This provides a solid base of good test cases with known results. These test cases then make it far easier for the developers to improve and enhance ooRexx. After any code changes the suite of known working test cases can be executed to ensure the new code did not break anything.

An ooRexxUnit X.X.X Snapshot is a package released by the ooRexx team that contains both the testing framework and the set of tests used in testing the ooRexx interpreter at that X.X.X level. As the version name implies, it is a snapshot of ooRexxUnit at that point in time. The package contains all the currently available test groups written to test the X.X.X interpreter. These test groups are an excellent source of techniques for using ooRexxUnit.

The primary purpose of releasing snapshot versions of ooRexxUnit is to encourage individuals and organizations to contribute test cases for the interpreter.

Writing test groups to test the interpreter requires nothing other than an installed X.X.X interpreter and a framework package. If an individual has limited knowledge of ooRexx programming, then writing test cases will be a excellent way to learn the language.

Individuals wishing to contribute to ooRexx through writing test case can get advice and help by joining the oorexx-devel list. The list is free and open to anyone. Go to: *Subscribe oorexx-devel list* (https://sourceforge.net/mail/?group_id=119701) to join the list.

In addition the RexxLA mailing list is also a good place to seek advice or help in writing test units. If not already a member of RexxLA, more information on the group can be found at: *RexxLA Home Page* (<http://rexsla.org/>)

1.2. Download

Go to the file download section of the ooRexx project on SourceForge: *SourceForge.net Files* (https://sourceforge.net/project/showfiles.php?group_id=119701) and download the snapshot for your version of ooRexx.

Most of the time the snapshots are packaged in both zip and tar format. However, the ooTest framework and files are completely platform independent. Either package will work on any system that ooRexx is working on. Pick the packaging type that is most convenient for your needs.

1.3. Extract the Files

Open a console window and unzip or untar the snapshot in a convenient spot. After unpackaging you will end up with a directory structure similar to this:

```
ooRexxUnit.X.X.X
|
*-----*framework
|       |
|       *-----<subdirectories>
|
*-----*misc
|       |
|       *-----<subdirectories>
|
*-----*ooRexx
|
|       *-----<subdirectories>
```

The framework directory and subdirectories contain additional documentation and examples.

The ooRexx directory and subdirectories contain all the tests implemented using the ooTest framework. These sub-directories contain the tests used in testing the ooRexx interpreter.

The misc directory has, well miscellaneous stuff. The most important of which for this discussion is a template file that can be used to start your test group files and several examples of test group files.

1.4. Test Your Install

After unpackaging the snapshot, cd into the top level directory and read the ReadMe.first file. The top level directory will look similar to below:

```
E:\work.ooRexx\ooRexxUnit\3.2.0>dir
Volume in drive E is Blackfoot
Volume Serial Number is 9C3D-6D2A

Directory of E:\work.ooRexx\ooRexxUnit\3.2.0

07/03/2008  08:00 PM    <DIR>          .
07/03/2008  08:00 PM    <DIR>          ..
07/03/2008  08:00 PM    <DIR>          framework
07/03/2008  08:06 PM    <DIR>          misc
07/03/2008  08:06 PM    <DIR>          ooRexx
11/29/2007  05:47 PM                11,839 CPLv1.0.txt
```

```

12/19/2007 09:15 PM          2,822 directory.structure.tests
06/30/2008 05:47 PM          1,737 Expected.Results
01/18/2008 10:39 AM        69,323 ooTest.frm
06/30/2008 05:47 PM          5,918 ReadMe.first
01/02/2008 08:03 PM          9,139 runTestGroups.rex
12/04/2007 08:07 PM          2,437 setTestEnv.bat
12/04/2007 08:06 PM          3,062 setTestEnv.sh
01/13/2008 01:15 PM          4,441 test00Rexx.rex
01/12/2008 08:44 PM        18,190 worker.rex
          10 File(s)        128,908 bytes
          6 Dir(s)  14,359,519,232 bytes free

```

Note: For the sake of this document I am going to show examples on Windows. But, the same general thing applies to Linux. Just translate the slashes. In addition the examples are from the 3.2.0 version of ooTest. The same general principles apply to whatever snapshot you have.

The program file test00Rexx.rex is what drives the automated execution of the test cases. Provided that you have a standard ooRexx install, you can execute the entire test suite as follows. This command will execute the entire test suite. Depending on your system it will take several minutes to finish.

```

E:\work.ooRexx\ooRexxUnit\3.2.0>rexx test00Rexx.rex -V 5
Searching for test containers.....
Executing automated test suite.....
.....
.....

```

ooTest Framework - Automated Test of the ooRexx Interpreter

```

Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit:  2.0.0_3.2.0          ooTest: 1.0.0_3.2.0

```

```

Tests ran:          16600
Assertions:         537835
Failures:           2
  (Known failures:) 42
Errors:             2
Exceptions:         0
Skipped files:      0
Messages:           0

```

```

[failure] [20080819 17:37:57.809000]
  Test:  TEST_MULTIPLE_INHERITANCE_WITH_MULTIPLE_METACLASSES
  Class:  Class.testGroup
  File:   E:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\base\class\Class.testGroup
  Line:   576
  Failed: assertEquals
         Expected: [['123.'], identityHash="495954478"]

```

```

    Actual:  [['231.'], identityHash="421451928"]

[failure] [20080819 17:37:57.825000]
  Test:  TEST_SUBCLASSES
  Class:  Class.testGroup
  File:  E:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\base\class\Class.testGroup
  Line:  684
  Failed: assertTrue
    Expected: [1]
    Actual:  [[0], identityHash="535806184"]

[error] [20080819 17:37:57.809000]
  Test:  TEST_MIXINCLASS_01
  Class:  Class.testGroup
  File:  E:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\base\class\Class.testGroup
  Event: [SYNTAX 97.1] raised unexpectedly.
    Object "NOT_AN_EXISTING_CLASS" does not understand message "NEW"
    Line:  509
    509 ** cl=.object~mixinclass("subTest_01", not_an_existing_class)

[error] [20080819 17:37:57.825000]
  Test:  TEST_SUBCLASS_01
  Class:  Class.testGroup
  File:  E:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\base\class\Class.testGroup
  Event: [SYNTAX 97.1] raised unexpectedly.
    Object "NOT_AN_EXISTING_CLASS" does not understand message "NEW"
    Line:  616
    616 ** cl=.object~subclass("subtest_01", not_an_existing_class)

Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit:  2.0.0_3.2.0          ooTest: 1.0.0_3.2.0

Tests ran:          16600
Assertions:         537835
Failures:           2
  (Known failures:) 42
Errors:             2
Exceptions:         0
Skipped files:     0
Messages:           0

File search:        00:00:51.456000
Suite construction: 00:00:02.000000
Test execution:     00:02:17.632000
Total time:         00:03:11.620000

E:\work.ooRexx\ooRexxUnit\3.2.0>

```

You should get results similar to the above. Included with the snapshot is a file called `Expected.results`. That file should show you results similar to what you actually get.

1.5. Understanding What you See

We looked at the entire output from running the automated test suite. Let's examine some of the sections in more detail. The below stats show:

```
Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit: 2.0.0_3.2.0          ooTest: 1.0.0_3.2.0
```

```
Tests ran:          16600
Assertions:         537835
Failures:           2
  (Known failures:) 42
Errors:             2
Exceptions:         0
Skipped files:      0
Messages:           0
```

```
File search:        00:00:51.456000
Suite construction: 00:00:02.000000
Test execution:     00:02:17.632000
Total time:         00:03:11.620000
```

that 16,600 tests ran, comprising 537,835 assertions, taking about 3 minutes total to finish. There were 2 failures, 42 known failures, and 2 errors.

For a little terminology. When you write a test case there are 2 expected outcomes. It is expected that the test case either passes or fails. So in ooTest, a failure is a test case that did not pass. An error is an **unexpected** event. We are going to ignore the errors for now. In general they indicate something is wrong, maybe with the framework, maybe with the interpreter.

I'm going to just explain one of the failures. You have this output:

```
[failure] [20080819 17:37:57.809000]
  Test:  TEST_MULTIPLE_INHERITANCE_WITH_MULTIPLE_METACLASSES
  Class:  Class.testGroup
  File:   E:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\base\class\Class.testGroup
  Line:   576
  Failed: assertEquals
    Expected: [['123.'], identityHash="495954478"]
    Actual:   [['231.'], identityHash="421451928"]
```

The above shows that the name of the failed test was

TEST_MULTIPLE_INHERITANCE_WITH_MULTIPLE_METACLASSES and it can be found on line 576 in the file `Class.testGroup`.

What failed was an assertion that two things were equal. The thing was expected to be equal to '123' but it actually was '231'

Unfortunately these failing two test are a little complex, maybe not the best for an intro, but to continue with what we have. If we look at the failing test file on line 576 we see:

```
self~assertEquals("'123.'", .class123~info)
```

Okay, it has a method, but just bear with me for a second. This is the heart of writing a test case.

```
assertEquals("123", .class123~info)
```

Think of the method just as a routine for now with some prefix on it that you do not need to worry about. The routine name is assertEquals. The routine has 2 args and what we are doing is asserting that the 2 args are equal.

If they are not equal, the test case fails and the ooTest framework takes care of all the details of reporting the failure. That is the print out you see above. For now, don't worry about how the framework does this, just accept that it does it.

Next, let's look at the lines around 567, and you see this:

```
::method "test_MULTIPLE_INHERITANCE_WITH_MULTIPLE_METACLASSES"  
  
    self~assertEquals("'123.'", .class123~info)
```

Think of the method as a routine, for now, that is named:

test_MULTIPLE_INHERITANCE_WITH_MULTIPLE_METACLASSES. And that is the test name, the name reported in the print out. Next is the assertion, assertEquals(). The test writer is asserting that "123" is equal to .class123~info. The assertion failed, the test failed.

You and I don't need to figure out what .class123~info is right now.

Enough for an introduction.

Chapter 2. Starting to Write a Test Case

In the last section we looked at this, which I said was the heart of a test case:

```
::method "test_MULTIPLE_INHERITANCE_WITH_MULTIPLE_METACLASSES"

  self~assertEquals("'123.'", .class123~info)
```

In this section we will start to write our own test case.

2.1. A First Test Case

Say we want to test that the interpreter is adding $2 + 3$ correctly. We expect our test case either to show that the interpreter is adding $2 + 3$ correctly or that it is not.

To write a test case, it has to be in a method, which always starts as follows:

```
::method
```

That is boilerplate, just type it. Then the next key thing is that the framework executes each method whose name begins with 'test', case not significant, as a test case. We need to name the test case method, so we do this:

```
::method test_simpleAddition
```

That's also boilerplate, make up a name, start it with test. I usually add the underscore, but it is not needed.

What's the test? We know that $2 + 3$ is 5, so if the interpreter adds $2 + 3$ we would expect the result to be 5. We write some code that adds $2 + 3$ and then assert that the result is 5:

```
::method test_simpleAddition
  val = 2 + 3
  self~assertEquals(5, val)
```

That is all there is to it. The above 3 lines of code will test that the interpreter adds 2 plus 3 correctly. The framework takes care of all the other details.

Well there is a little more to it, we need to add the test case methods to a file, I'll show that later.

To summarize, to get started: Think of `::method` as the beginning of a function definition, where the function name comes right after the `::method`. Think of `assertEquals()` as a function call. And just note that the function call has to start with a prefix of `'self~'`. You do not need to really understand the `'self~'` part at first. Just know that you have to add it. We'll worry about learning about classes some other time. This is enough to write test cases.

2.2. Contributing to the ooRexx Project

At this point, I'm going to interject a little info on contributing to the ooRexx project / contributing to the ooRexx test suite.

The test suite is enormously helpful to the project. Everyone and any one can contribute to this effort. Even though we have a large number of tests, we still have a lot of holes in the coverage.

There are 4 main ways someone can contribute:

- You could write a whole new *.testGroup file.
- You can add tests to an existing *.testGroup file.
- You can examine existing test case code for correctness and correct or improve wrong or weak test cases.
- You can help organize or promote the contribution of test cases into the project.

It is my strong opinion that all the .testGroup files in the test suite are works in progress. In addition, we could really use more pairs of eye examining the existing .testGroup files with a critical eye.

Any one and every one is encouraged to either add to an existing file or to say hey this test is wrong, it should be written this way. Or to say, hey the Lines.testGroup does not have a test for this option, here's a method that does test it.

The other thing that is really important, is that just because something is working now, that doesn't mean there shouldn't be a test for it. There is no guarantee that anything that is working today will be working after I make my next commit. If I break something with my next commit, and we have a test case for it, it will show up right away and get fixed.

For example, you may think that there is no way the say instruction could be broken. But what about if you have this in a Rexx file:

```
/* Simple.rex */  
say 'Hello World'
```

and then run it like this:

```
E:\>rexx Simple.rex > myTest.file
```

You would expect myTest.File to contain 1 line consisting of:

```
Hello World<endOfLineCharacter>
```

Well, a real life bug we just had was that instead of the normal Windows end of line which is 0x0d0a we were actually getting 0x0d0d0a.

Now, running the test suite did catch this, but not because we had a test for it. The test suite caught it because some other tests were failing with what appeared to be no reason. It took quite awhile to locate what was really wrong.

We did not, and still don't, have a simple test case that asserts that a say statement redirected to a file produces what it should. The simple test should assert that the bytes in the file are exactly 13 and then assert that byte 1 is 0x48, byte 2 is 0x45, ... byte 12 is 0x0d, byte 13 is 0x0a.

Next chapter, create a testGroup file from the template and start a test group.

Chapter 3. Starting a Test Group from Scratch

In the ooTest framework, tests are organized as follows. An assertion is a single test. A single method represents a test case. Each test case would contain at least one assertion, but often a test case will contain several assertions. Then a number of test cases for a similar area are gathered together in a file, which we call a test group.

3.1. Starting a Test Group

I'm going to assume that the reader has been following along in the thread and try not to repeat myself. The idea behind this thread is to show how to start writing test cases even if you do not understand classes by using boilerplate code.

If you downloaded a snapshot and unzipped it, it will have created a directory tree. We start in the root of the tree which will be named ooRexxUnit.X.X.X.

There is the misc/ subdirectory. It has a template file. Pick a name for the test group and figure out roughly where it should go. Copy the template to the subdirectory renaming it in the copy. I am going to work with the stream BIF, since we do not even have a test group started for that important BIF. This example is on Linux

```
Raven:/ooRexxUnit.3.2.0 # cp misc/template.testGroup ooRexx/base/bif/STREAM.testGroup
```

You don't even have to put it in the proper subdirectory. To get started you could just put it anywhere under the ooRexx subdirectory.

```
Raven:/ooRexxUnit.3.2.0 # cp misc/template.testGroup ooRexx/STREAM.testGroup
```

3.2. Editing STREAM.testGroup

Open the file in an editor. The first thing I do is a search and replace of `template.testGroup` with `STREAM.testGroup`. I'm not going to give editor lessons, but if you do not understand classes, be sure you don't skip this first step.

Then starting at the top of the editor we have the first 41 lines that you just ignore and leave alone. The first few are:

```
#!/usr/bin/rexx
/*
  SVN Revision: $Rev: 2267 $
  Change Date:  $Date: 2008-01-18 09:41:04 -0800 (Fri, 18 Jan 2008) $
*/
```

Which are just book keeping. The `#!/usr/bin/rexx` causes the file to execute as a script on Linux, etc. The next is just svn book keeping. Then there is the license text.

Starting on line 42 through 51 we have what I think of as the entry point to an ooRexx program that has directives in it. You do not need to understand this to start off with, it is the code that lets the framework automate the execution of tests. You just need to be sure you changed `template.testGroup` to `STREAM.testGroup`. The lines look like:

```
parse source . . s

group = .TestGroup~new(s)
group~add(.STREAM.testGroup)

if group~isAutomatedTest then return group

testResult = group~suite~execute~~print

return testResult
-- End of entry point.
```

Briefly what this does is:

- Create a new `TestGroup` object using the full path name of the file.
- Add the `STREAM.testGroup` class to the test group
- Magically test if the invocation of the program is part of an automated test. If so the group object is returned and the code execution is done.
- If it is not an automated test, it is a stand alone test. The test group executes all the tests it contains and prints out the results, then returns the test result object. This allows you to execute the tests by just invoking the file as an ooRexx program. I am not going to go into details about that now.

When working on just your own test group file, it is best to just run the single test group by itself through the frame work. Right now you have a complete test group that will execute. You can run it using this command:

```
Raven:/ooRexxUnit.3.2.0 # ./test00Rexx.rex -R ooRexx -f STREAM
```

(The above should work, but I can not try it right now, so I am going to show it on Windows.)

```
E:\ooRexxUnit.3.2.0>test00Rexx.rex -R ooRexx\ -f STREAM
Searching for test containers..
Executing automated test suite..
```

ooTest Framework - Automated Test of the ooRexx Interpreter

```
Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit:  2.0.0_3.2.0          ooTest: 1.0.0_3.2.0
```

```
Tests ran:          4
Assertions:         2
Failures:           0
```

```

Errors:                0
Skipped files:         0

File search:           00:00:00.047000
Suite construction:    00:00:00.000000
Test execution:        00:00:00.000000
Total time:            00:00:01.000000

E:\ooRexxUnit.3.2.0>

```

Okay, that's it. You created and executed your first group of tests. 4 tests ran using 2 assertions with no failures or errors. On my system the total execution time was 1 second.

That takes care of all the initial steps. You have a working test group.

The next installment will show adding some test cases to the new test group.

3.3. STREAM.testGroup Continued

Picking up with writing the new test group `STREAM.testGroup`.

The last section showed the top boilerplate from the the template file. This picks up with the actual test case class. I'm not to going to explain too much about the class part of this, I'll save that for later. This is about just editing the boilerplate to quickly get started.

This is the rest of the file, leaving out some comment lines. Format your code however you like.

```

-- End of entry point.

::requires 'ooTest.frm'

::class "STREAM.testGroup" public subclass ooTestCase

::method test_YYY
    self~assertTrue(.true)

::method test_XXX
    self~assertSame('dog', 'dog')

```

The `::requires` line pulls in the `ooTest` framework. That is what provides the backing code for our test cases.

Then we have our `STREAM.testGroup` class which is a subclass of `ooTestCase`. Once you do the replace of `template.testGroup` with `STREAM.testGroup` you are done with those lines. You can just leave the boilerplate alone and figure out what this means later.

Then, to restate what I've said earlier. Each individual test is a method of the test case class where the method names starts with 'test' Here our test case class is: `STREAM.testGroup` and we see that we have 2 methods: `test_YYY` and `test_XXX`.

That is why executing the `STREAM.testGroup` actually runs some tests, even though we didn't add any tests to the file yet.

If you don't understand classes yet, just think of the two methods as two routines, or functions, or procedures. Whichever terminology you are comfortable with. I'm going to call them methods, because that's what they are.

The 2 methods are in the template to jog your memory and get you started. I start out by editing the first method and coding an actual test. We are doing a group of tests for the stream BIF, so an easy way to get started is to look at the documentation for the stream BIF.

If you are following along and don't understand stream at all, the advice is that the principles here apply to writing any test cases, pick some area of REXX you do understand. Plus, I have to add, if you don't understand stream, then digging in to it enough to write some test cases is a great way to learn about stream.

Looking at the doc, we see that it says stream returns a string, which string is dependent on the args to stream, and that the first arg names the stream to be worked with. The second arg can be one of: State, Command, or Description.

For the Description arg it says that it returns the same string as State, but with a colon at the end, maybe followed by some text describing an error or not ready condition. That suggests a test. If we use the Description arg, the returned string must have a colon in it.

3.4. Finally, the Interesting Part

This is the interesting part. If you like to program, then it is usually fun to think of a way to code a test for this. Here is one way.

The mechanics of this are to change the name of the first method. The restriction is that every method in the class has to have a unique name. If there is a failure, the name of the method gets printed out. If the name of the method reflects the test, it is a little easier to discern what the test is about. But, you could just name every method test_001, test_002, test_003 and so on.

Here goes, rename the method:

```
::method test_description_arg
```

One of the most common types of streams is file input or output. The name of the stream is the file name. One approach is to use a file name for the stream name, use the Description arg, and validate the return:

```
::method test_description_arg
  streamName = ???
  retString = stream(streamName, "Description")
  <check retString>
```

Remember this is going to be an automated test that should run on any system. We don't want to use a file name that is on your system, but is not on Rick's system. What file name are we absolutely sure exists?

The `STREAM.testGroup` file, for sure, because that is the file we are using. We can get that name from `parse source`. If you don't know `parse source`, look it up in the doc.

```
::method test_description_arg
  parse source junk notUsed streamName
  retString = stream(streamName, "Description")
  <check retString>
```

Now we just need to code the check of `retString`.

3.5. The Core, Validating Results

Here we are at the core of the `ooTest` framework. We validate expected results by using one of the `assertXXX()` methods. I will list the different `assertXXX` a little later.

In the template code we already had:

```
self~assertTrue(.true)
```

The meaning of that should be easy enough to discern. We are asserting that `.true` is true.

Okay, our test is about the stated fact that the `Description` arg must return a string with a colon in it. We need to write some code that we can get true out of. There are a number of ways to do this. I'll use a BIF since this is directed toward people who may not know classes too well.

We know that if the returned string has a colon in it, then the `pos` bif will return a non-zero position for the colon. Here is the complete test case:

```
::method test_description_arg
  parse source junk notUsed streamName
  retString = stream(streamName, "Description")
  p = pos(retString, ":")
  self~assertTrue(p > 0)
```

3.6. The Finale, Executing our Test

And here is the output from executing this test group (stripped of a little bit):

```
C:\ooRexxUnit.3.2.0>rexx test00Rexx.rex -R ooRexx -f stream
...

Tests ran:          4
Assertions:         1
Failures:           1
Errors:             0
Skipped files:      0

[failure] [20080701 07:48:09.882000]
```

```
Test:    TEST_DESCRIPTION_ARG
Class:   STREAM.testGroup
File:    C:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\STREAM.testGroup
Line:    68
Failed:  assertTrue
        Expected: [1]
        Actual:    [[0], identityHash="535806184"]
```

This is perfect! Okay, the output shows that the test named TEST_DESCRIPTION_ARG failed. Whoa, that's the one I just wrote. It shows that what failed was assertTrue and that it was on line 68.

What was expected was 1 (true) but the actual was 0 (false) Let's look at the test case again:

```
p = pos(retSting, ":")
self~assertTrue(p > 0)
```

I always forget the args to pos. They are needle, haystack. I coded haystack, needle. Well the astute observer may have noticed that I also coded retSting rather than retString. The fixed test case:

```
::method test_description_arg
  parse source junk notUsed streamName
  retString = stream(streamName, "Description")
  p = pos(":", retString)
  self~assertTrue(p > 0)
```

The output:

```
Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit:  2.0.0_3.2.0          ooTest: 1.0.0_3.2.0

Tests ran:           4
Assertions:          2
Failures:            0
Errors:              0
Skipped files:       0

File search:         00:00:00.160000
Suite construction:  00:00:00.000000
Test execution:      00:00:00.000000
Total time:          00:00:00.160000
```

Alright. Now, if someone rewrote the stream libraries, a lot of work, and had some trivial error that forgot to tack on the ":" for the Description arg, we have a test case that would catch it.

3.7. One More Quick Test

Let's do one more quickly.

The doc also says that the Description arg returns the same string as the State arg, with a colon and some other possible text added after the colon. That suggests a test.

Here is the code. Notice the use of `assertSame()` which really fits in with the semantics of the test. We are testing that the Description and State args return the same string:

```
::method test_description_state_same
  parse source junk notUsed streamName

  retDiscript = stream(streamName, "Description")
  retState    = stream(streamName, "State")

  retDiscript = left(retDiscript, pos(":", retDiscript) - 1)

  self~assertSame(retDiscript, retState)
```

Here is the output:

```
C:\ooRexxUnit.3.2.0>rexx test00Rexx.rex -R ooRexx -f stream
...
Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit:  2.0.0_3.2.0          ooTest: 1.0.0_3.2.0

Tests ran:           4
Assertions:          2
Failures:            0
Errors:              0
Skipped files:       0

File search:         00:00:00.160000
Suite construction:  00:00:00.000000
Test execution:      00:00:00.000000
Total time:          00:00:00.160000
```

To code the 2 tests, we really just used classic REXX. The 'object' stuff is confined to some boilerplate code that you should be able to use without really understanding it. The process of just writing this, will allow the 'object' stuff to start seeping in.

Next installment: I am always threatening to write the documentation for the ooTest framework, but it is an *empty* threat. Or - what *are* the assertXXX methods?

Chapter 4. Some Reference Documentation

This chapter will provide some basic documentation for the ooTest framework. In the spirit of the rest of this document, it is minimalistic. Just enough to write simple tests.

The best way to get more information if you are stuck is to join the ooRexx-Devel list and ask questions. Asking questions on that list will help in several ways. You'll get your question answered. Other people that have the same question will benefit from the answer. The question and answer become part of the permanent archive. And, hopefully, that information will migrate from the list into the ooTest documentation in a similar manner to the way the information in this document was migrated.

4.1. The assertXXX() Methods

The main thing that is left is to list the different assertXXX() methods. They are:

- `assertEquals(expected, actual, [msg])`
- `assertNotEquals(expected, actual, [msg])`
- `assertNull(actual, [msg])`
- `assertNotNull(actual, [msg])`
- `assertSame(expected, actual, [msg])`
- `assertNotSame(expected, actual, [msg])`
- `assertTrue(actual, [msg])`
- `assertFalse(actual, [msg])`

Some key points.

- 1.) All of these are methods, so to use them in your tests you do, for example:

```
self~assertTrue(.true)
```

If you are hazy on objects, just think of that as a routine with a mandatory prefix of 'self~'

- 2.) Where you have expected and actual args, expected always comes first.

```
val = 2 + 3
self~assertSame(5, val)
```

5 is expected, val is the actual.

- 3.) Where you just have the actual arg, it is the actual thing.

```
val = .true
self~assertTrue(val)
```

or

```
val = 2 + 3
self~assertTrue(val == 5)
```

4.) As the above shows, there are usually several ways to write the same assertion. Pick what you are comfortable with.

5.) All the assertions have an optional 'message' argument that is the last arg. The message is printed out if the assertion fails.

Here is an example:

```
::method test_addition
  val = 2 + 3
  self~assertSame(6, val, 'Special test, 2 plus 3 must equal 6')
```

and the output:

```
[failure] [20080701 09:19:45.455000]
Test:    TEST_ADDITION
Class:   STREAM.testGroup
File:    C:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\STREAM.testGroup
Line:    82
Failed:  assertSame
Expected: [[6], identityHash="535894080"]
Actual:   [[5], identityHash="535906054"]
Message:  Special test 2 plus 3 must equal 6
```

The intention of the message is to give some idea about what the test is testing. Here we see that the test writer has some special purpose in mind for this test.

Without the message and a non-descriptive test name we would see:

```
[failure] [20080701 09:22:45.367000]
Test:    TEST_001
Class:   STREAM.testGroup
File:    C:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\STREAM.testGroup
Line:    82
Failed:  assertSame
Expected: [[6], identityHash="535894080"]
Actual:   [[5], identityHash="535906054"]
```

which, if you were not the one that wrote the test, might leave you clueless. Then if you look at the test you would see:

```
::method test_001
  val = 2 + 3
  self~assertSame(6, val)
```

And you might really wonder what the test writer had in mind for this test in the STREAM test group. (Of course even with the message, we still don't know why 2 + 3 should equal 6.)

In general you don't need a message. But, it can help clarify what the test is about by using a message that states what is expected.

6.) The assertXXX() methods are pretty self explanatory, with maybe the exception of `assertSame()` and `assertEquals()`.

`assertSame()` (and `assertNotSame()`) assert that 2 things are strictly equal.

```
5 == 5
```

`assertEquals()` (and `assertNotEquals()`) assert that 2 things are loosely equal.

```
'dog ' = 'dog'
```

`Equals` is also used to test if 2 collections are loosely equal.

```
a1 = .array~of(1, 2, 3)
a2 = .array~of(1, 2, 3)
self~assertEquals(a1, a2)
```

`assertNull()` and `assertNotNull()` might also need some explaining. They test if the actual is `.nil` (or not `.nil`).

7.) The assertXXX() methods are located in the OOREXXUNIT.CLS file. Since this is open source you can always browse the file to see what assertXXX() methods are available to you. Which is what I do since I can never remember them and no one has produced easy to find documentation for the framework.

OOREXXUNIT.CLS is located in the framework subdirectory:

```
C:\ooRexxUnit.3.2.0>dir framework\OOREXXUNIT.CLS
Volume in drive C has no label.
Volume Serial Number is B4C0-DCBA

Directory of C:\ooRexxUnit.3.2.0\framework

05/12/2008  01:15 PM                65,321 OOREXXUNIT.CLS

-----
```

Speaking of doc etc., in the misc subdirectory there are some sample test group files. Looking at those should be helpful. I believe I tried to comment them well.

```
C:\ooRexxUnit.3.2.0>dir misc
Volume in drive C has no label.
Volume Serial Number is B4C0-DCBA

Directory of C:\ooRexxUnit.3.2.0\misc

01/18/2008  10:41 AM                5,755 SampleOLEObject.testGroup
01/16/2008  01:40 PM                5,638 Simplest.testGroup
01/16/2008  04:17 PM                5,775 SimpleWithOneTimeSetup.testGroup
01/16/2008  04:28 PM                5,609 SimpleWithSomeSetup.testGroup
```

There is also doc and more examples in the framework directory. However, those docs and examples are from the original ooRexxUnit framework.

The framework we are using here is the ooTest framework, which sits on top of ooRexxUnit. The ooRexxUnit doc does not necessarily apply to ooTest. And, I have made changes to ooRexxUnit, but have not been rigorous about maintaining the ooRexxUnit doc and examples.

Chapter 5. How to Become a Committer

Armed with this quick start, I think that anyone who can program in Rexx and is using ooRexx to some degree can begin writing test cases for the ooRexx interpreter.

Ask questions when you get stuck. Contribute test cases to the project. Earn yourself the status of a committer.

About that last sentence. Want to have the exalted rank of a committer on an Open Source project? But you don't know C or C++. Contribute test cases to the ooRexx project and I guarantee I will push to make you a committer. And, Rick will vote in favor of making you a committer.

Chapter 6. Negative Tests

In my original series of e-mail posts, Rick pointed out that I forgot to include how to do negative tests. This is really an important part of testing, and since I spent a lot of time in test, I'm kicking myself for overlooking it. <grin>

6.1. Example One

We'll continue with the `STREAM.testGroup` and look at what the doc says about the second arg. Which is exactly: "The second argument can be one of the following strings ..." and it lists exactly 3 strings: State, Description, and Command.

So a negative test is what happens when you do something that is incorrect. In this case what happens if you don't follow a requirement. (Well there are other types of negative tests, but let's not get too complicated here.)

The requirement is that the second arg be one of only 3 different things. What we want to test, is what happens if we don't use one of the 3 specified args. Say if we use `FILESTATE` ?

Staying with the same `STREAM.testGroup`. How do we test this? Maybe something along those lines?

```
::method test_wrong_2nd_arg
  parse source junk notUsed streamName

  retDiscript = stream(streamName, "FILESTATE")
```

Output:

```
[error] [20080701 12:06:53.206000]
Test:  TEST_WRONG_2ND_ARG
Class: STREAM.testGroup
File:  C:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\STREAM.testGroup
Event: [SYNTAX 40.904] raised unexpectedly.
      STREAM argument 2 must be one of SDC; found "FILESTATE"
Line:    83
      83 ** retDiscript = stream(streamName, "FILESTATE")
```

Now if you look closely at that, it was not a failure, it was an error.

If you remember back to what I said about terminology, I said with a test case you expect one of 2 outcomes. You either expect the test to pass, or you expect the test to fail.

Errors are something totally unexpected. We can see from the output that the interpreter is handling this the way it should. But, what we want is a test case for this that *passes*.

In this case we **expect** the error condition. So our test case will pass when we get an error. The framework provides for these types of test with the `expectSyntax()` method.

We code our test like this:

```
::method test_wrong_2nd_arg
    parse source junk notUsed streamName

    self~expectSyntax(40.904)
    retDiscript = stream(streamName, "FILESTATE")
```

And the output we get is:

```
C:\ooRexxUnit.3.2.0>rexx test00Rexx.rex -R ooRexx -f stream
..
Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit:  2.0.0_3.2.0          ooTest: 1.0.0_3.2.0

Tests ran:           5
Assertions:          3
Failures:            0
Errors:              0
Skipped files:       0

File search:         00:00:00.160000
Suite construction:  00:00:00.000000
Test execution:      00:00:00.000000
Total time:          00:00:00.160000
```

Great, we now have 3 tests coded and they all pass.

6.2. Example Two

What else can we test in this way? We see from the doc that the first arg to `stream` is required. What happens if we leave it out? We should get some type of error. And the interpreter should not blow up.

Now how to code it? Similar to what we just did:

```
::method test_no_args

--self~expectSyntax(40.904)
retDiscript = stream()
```

But, what syntax error do we expect? The best thing to do, is to look up the error numbers and descriptions and decide for yourself what you think it *should* be.

The error message list is in the back of the doc, in an appendix. I don't want to drag this out, so I'm going to say, since the first error was in the 40. range, that is a good place to start. Looking there we see:

```
001
  External routine "routine" failed
```

Okay, that seems reasonable. Though, there is also:

```
005
  Missing argument in invocation of routine; argument argument_number is required
```

Aha, that is it. Argument 1 is required, says the doc. So the message: "Missing argument in invocation of STREAM; argument 1 is required" seems perfect.

Our test looks like:

```
::method test_no_args

  self~expectSyntax(40.005)
  retDiscript = stream()
```

and our output looks like this:

```
[error] [20080701 12:27:15.323000]
Test:  TEST_NO_ARGS
Class: STREAM.testGroup
File:  C:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\STREAM.testGroup
Event: [SYNTAX 40.3] raised unexpectedly.
      Not enough arguments in invocation of STREAM; minimum expected is 1
Line:    90
      90 ** retDiscript = stream()
```

Eureka! We found a bug! This is what it is all about, finding bugs. <grin> But, let's investigate a little. What is 40.3?

```
003
Not enough arguments in invocation of routine; minimum expected is number
```

Making the message: Not enough arguments in STREAM; minimum expected is 1 Well, okay, that message also seems to fit. What to do? If you find something like this, that you think is a bug, then the thing to do is bring it up on the developer's list and lay out your case for why you think this is a bug.

In this instance, you would not prevail with your case. The reason being that 40.3 is applicable and it is what has been used in the past. Therefore, changing it because you think 40.5 is better has the potential of breaking existing code that is checking for this specific error code.

So, the test looks like this:

```
::method test_no_args

    self~expectSyntax(40.003)
    retDiscript = stream()
```

output:

```
[error] [20080701 12:39:59.063000]
Test:  TEST_NO_ARGS
Class:  STREAM.testGroup
File:   C:\work.ooRexx\ooRexxUnit\3.2.0\ooRexx\STREAM.testGroup
Event:  [SYNTAX 40.3] raised unexpectedly.
       Not enough arguments in invocation of STREAM; minimum expected is 1
       Line:      90
       90 ** retDiscript = stream()
```

Great!! Now we REALLY found a bug! It is supposed to be error 40.003. Well, again, do a little research. What the doc says is:

```
Some errors have associated subcodes. A subcode is a one- to
three-digit decimal extension to the error number, for example, 115 in
40.115
```

A subcode is a *one* to three digit number.

Okay, no bug. Our test looks like this:

```
::method test_no_args

    self~expectSyntax(40.3)
    retDiscript = stream()
```

output:

```
Interpreter: REXX-ooRexx_3.2.0(MT) 6.02 30 Oct 2007
ooRexxUnit:  2.0.0_3.2.0          ooTest: 1.0.0_3.2.0
```

```
Tests ran:          6
Assertions:         4
Failures:           0
Errors:              0
Skipped files:      0
```

```
File search:        00:00:00.160000
Suite construction: 00:00:00.000000
Test execution:     00:00:00.000000
Total time:         00:00:00.160000
```

Now we have 4 tests of an area that hadn't been touched. The 4 additional tests add 0 seconds to the overall test execution time. 2 of the tests are positive tests, 2 are negative.

Appendix A. Notices

Any reference to a non-open source product, program, or service is not intended to state or imply that only non-open source product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Rexx Language Association (RexxLA) intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-open source product, program, or service.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-open source products was obtained from the suppliers of those products, their published announcements or other publicly available sources. RexxLA has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-RexxLA packages. Questions on the capabilities of non-RexxLA packages should be addressed to the suppliers of those products.

All statements regarding RexxLA's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

A.1. Trademarks

Open Object Rexx™ and ooRexx™ are trademarks of the Rexx Language Association.

The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

1-2-3
AIX
IBM
Lotus
OS/2
S/390
VisualAge

AMD is a trademark of Advance Micro Devices, Inc.

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in

the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

A.2. Source Code For This Document

The source code for this document is available under the terms of the Common Public License v1.0 which accompanies this distribution and is available in the appendix [Common Public License Version 1.0](#). The source code itself is available at

http://sourceforge.net/project/showfiles.php?group_id=119701.

The source code for this document is maintained in DocBook SGML/XML format.



Appendix B. Common Public License Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

B.1. Definitions

"Contribution" means:

1. in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
2. in the case of each subsequent Contributor:
 - a. changes to the Program, and
 - b. additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

B.2. Grant of Rights

1. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.
2. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such

combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

3. Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.
4. Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

B.3. Requirements

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

1. it complies with the terms and conditions of this Agreement; and
2. its license agreement:
 - a. effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
 - b. effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
 - c. states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
 - d. states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

1. it must be made available under this Agreement; and
2. a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

B.4. Commercial Distribution

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

B.5. No Warranty

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

B.6. Disclaimer of Liability

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE

PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

B.7. General

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Index

C

Common Public License, [33](#)
CPL, [33](#)

L

License, Common Public, [33](#)
License, Open Object Rexx, [33](#)

N

Notices, [31](#)

O

ooRexx License, [33](#)
Open Object Rexx License, [33](#)

