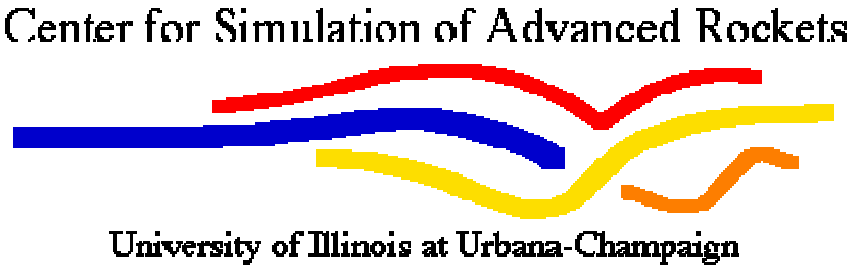

Roctest User's Guide

Center for Simulation of Advanced Rockets
University of Illinois at Urbana-Champaign
2260 Digital Computer Laboratory
Urbana, IL

Title:	Roctest User's Guide
Author:	Mark Brandyberry
Subject:	Documents use of the Roctest code.
Revision:	Initial (0) 12/2002 Update (1) 3/2005
Revision History	Revision 0: Initial Release Revision 1: Substantial Rewrite of Roctest
Effective Date:	3/17/2005

1.0 Introduction

Roctest is a script-based program (Perl) designed to run test problems using supplied executables and preinstalled test setups. It takes a list of test problems, the location of an executable program, and runs the test problems using the executable. It is installed on each platform on which it must be run. It is designed to be called by the Rocbuild program (see the Rocbuild User's Guide), but does not have to be. It uses pre-installed run description properties files named "JobRunData.conf" that must be included with each test problem. Test problems must be pre-installed on each platform and the test problem directories entered into Roctest's configuration file (described in section 4). Roctest is currently very much under development, and is likely to have further capabilities added to it in the near future. It is also only currently installed and running on the Turing platform, and for the rocstar3 code.

2.0 Purpose and Methods

Roctest's purpose is to take a newly built executable file (or an old one for that matter), one or more regression test cases, and to run the executable on each of the test cases. It then checks the screen dump file for a user-specified string to see if the run completed or not. It can then perform difference comparisons between some or all of the run output files compared to a set of "gold" standard results to assess whether the current results are the same as previous results. This regression test comparison assures that changes made in the code being tested have not changed the answers expected from a set of test problems.

Currently, Roctest is called at a command line in one of two ways:

1) with two arguments: the full path to the executable to run, and the symbolic name of one or more test cases to run (the test cases having been preconfigured in the RegisterTests.conf configuration file).

2) with the –f switch and a filename. The file referenced should contain one or more lines. Each line should have the full path to an executable to run, and a set of test cases following on the same line, separated by spaces. In this way, Roctest may be instructed to run multiple tests with multiple executables.

Roctest has been substantially rewritten as compared to its initial version. In particular, it has been changed to depend on batch submission of test jobs instead of interactive submission, since all platforms that it will target are now batch-oriented. Further, it is a multi-threaded application, so that all test job submissions are submitted (essentially) simultaneously. This means that each executable to be tested must have its own set of installed test cases, since two executables will not be able to share a single test case, since they would be expecting to both run at the same time.

3.0 Building and Running

Since Roctest is written in Perl, it is an interpreted code, and does not have to be "built". Currently, all files required for Roctest are in the Roctest/source directory, except for the

Properties.pm class, which is in the Common directory. The code is checked into CVS in the Rocstar/RocBT/Roctest directory and in the Rocstar/RocBT/Common directory. The structure is given below:

Common/Properties.pm
Roctest/source/ProblemProperties.pm
Roctest/source/Roctest.pm
Roctest/source/RoctestProperties.pm
Roctest/source/TestRunner.pm
Roctest/source/TuringRunner.pm

The Common and Roctest directories should be in the same directory (Roctest.pm references Properties.pm as ../../Common/Properties.pm).

The file Roctest.pm is the only one of the files that is executable. It is run as ./Roctest.pm, always with two or more command-line arguments: the full path and filename to the executable to run, and the symbolic name of one or more test cases to be run (see section 4.1.2 for discussion of registering test cases with Roctest), or the –f flag with a file containing multiple executable runs. Thus, two example runs would be:

```
./Roctest.pm /turing/projects/csar/autobuild/rocstar3/build4/bin/rocstar3 testcase1 testcase2

./Roctest.pm –f /turing/projects/csar/autobuild/rocstar3/build4/RoctestInput.txt
```

There are several directories that usually exist in the Roctest directory:

Rocbuild/conf -> where all configuration files are kept (see section 4.1)
Rocbuild/logs -> all log files are placed in this directory (not currently used)
Rocbuild/source -> all source code file (except Common files) are kept here.

Note that these paths and directory names may be changed by using parameters in the Global Configuration File described in section 4.1.1. To check that the source files at least "compile" (i.e., run through the interpreter) without error, the command "perl –c filename.pm" may be issued at the command line (where "filename.pm" is one of the source files listed above). The response "filename.pm syntax OK" should be the response.

The input file structure needed to run the code is described in section 4.1. Since Roctest runs completely on a single machine, it does not need access configuration. It is explicitly run by a calling program (usually Rocbuild) or from the command line, and is not managed by Cron or any other system program. Note however, that all test problems to be run must have a JobRunData.conf file (see section 4.1), and all test cases to be used must be preconfigured and registered with Roctest (also see section 4.1).

4.0 Input and Output (User Interface)

Roctest uses five types of input files: the Global Configuration File, the Registered Tests file, JobRunData.conf properties files for each test case, a difference file list for each test case, and

(optionally) a file containing multiple executables, each with test cases. These input files are described in section 4.1.1 through 4.1.6.

4.1 Input File Structure

The global configuration, registered tests, and JobRunData.conf files are examples of "properties" files, and they all have the same structure. A properties file is a series of key->value pairs, where the keys are pre-defined keywords, and the values are the values to be assigned to those keywords in the program. The separator "->" between the key->value pairs must be exactly "->", and there can be no spaces on either side of it. Each of the different properties file types contains different keys, and controls a different aspect of the program's operation. Note that in all configuration files, including parameters that are not needed will not hurt anything, but leaving out required parameters will abort the run. The parameters may be arranged in the file in any order, one per line. The global configuration and registered tests files should be in the "conf" directory described in section 3. The JobRunData.conf files will be stored with each of their respective problem datasets.

4.1.1 Global Configuration File

The global configuration file controls global aspects of the program, and must be named Roctest.conf, and be stored in the conf directory. No optional command-line definition of this file is allowed for Roctest (as it is allowed in Rocbuild). An example global configuration file is shown below. Each parameter is discussed subsequently.

```
#->This configuration file consists of param name=param value pairs.
#->Comment lines must start with '#->'.  Parameter name/value pairs

#->must have an (->) between the name and value, with
#->NO SPACES AROUND THE -> CHARACTERS!
#->Example:
#->      paramname->/usr/bin/perl

RocHome->/csar/Roctest/
confPath->/csar/Rocstar/RocBT/Roctest/conf
logsPath->/csar/Roctest/logs/
testHomePath->/csar/autotest/
regFile->/csar/Rocstar/RocBT/Roctest/conf/RegisterTests.conf
hostName->turing
submitFile->/home/mdbrandy/bin/pj_all
diffTool->/turing/projects/csar/Rocstar/Rocdiff/rocdiff
```

- RocHome is the full path to the base Roctest directory. REQUIRED
- confPath is the full path to the conf directory where configuration files are stored. Does not HAVE to be in Rocbuild directory. REQUIRED
- logsPath is the full path to the logs directory where all generated log files will be stored. Does not HAVE to be in Rocbuild directory. This parameter is not currently used. OPTIONAL

- testHomePath is the full path to the directory where the directories for all the test runs that will be made are stored. See section 4.1.2 for more information on test file configuration. REQUIRED

- regFile is the full path and name of the file that contains the list of registered tests. See section 4.1.2 for more information on test file configuration. REQUIRED

- hostName is the name of the system that Roctest is currently running on. It should be part or all of the text string returned by issuing the hostname command on the system. It is used to allow Roctest to know which system it is running on, so it can use any platform-specific functionality needed to submit or analyze jobs. REQUIRED

- submitFile is the full path and name of the script used to submit batch jobs on the system. Currently, Roctest does not contain platform-specific information on how to submit jobs on the supported systems, and relies on the pj_all script that has been written to submit rocstar3 jobs on all systems that rocstar3 is supported on. In the future, this parameter may allow substituting another script for job submission, but for now, pj_all is all that is supported. Any other script that takes the same command-line input parameters in the same order could be substituted for pj_all using this parameter. REQUIRED

- diffTool is the full path and name of the program used to compare test output files to the gold standard results. Any tool which takes the same input parameters as the Rocdiff tool (see the Rocdiff User's Guide) may be substituted here. Rocdiff input and output format are all that are currently supported in Roctest. Any tool that takes the same command-line input and writes the same output file format could be substituted using this parameter. REQUIRED

4.1.2 Registered Tests File

The registered tests file is just a list of symbolic test names associated with full paths to the root directory of the test archives to be run. The example file below lists six registered tests. Note that this file uses the properties file format discussed above, but the "keys" on the left side of the "->" are user-defined symbolic names for each test archive. These symbolic names are used by Rocbuild when it writes build scripts that drive Roctest. It passes the symbolic name of the requested test through to Roctest on the command line, and Roctest looks up the path in the Registered Tests file. That way, the same symbolic name can be used for the same tests on two different platforms, without Rocbuild needing to know the detailed path information.

```
#->This configuration file consists of param name=param value pairs.
#->Comment lines must start with '#->'.  Parameter name/value pairs
#->must have an (->) between the name and value, with
#->NO SPACES AROUND THE -> CHARACTERS!
#->Example:
#->      paramname->/usr/bin/perl

LS16FloFracPlain->/turing/projects/csar/autotest/gen3-data/labscale/LS16FloFracPlain
LS16FloFracCharm->/turing/projects/csar/autotest/gen3-data/labscale/LS16FloFracCharm
LS16FluFracCharm->/turing/projects/csar/autotest/gen3-data/labscale/LS16FluFracCharm
SB4ndCCPlain->/turing/projects/csar/autotest/gen3-data/spongebar/SB4ndCCPlain
```

```
SB4ndCCCharm->/turing/projects/csar/autotest/gen3-data/spongebar/SB4ndCCCharm
Arienti4Plain->/turing/projects/csar/autotest/gen3-data/Arienti/Arienti4Plain
```

The archives actually referred to should be *.tgz files containing the gzipped tar archive of a directory with the name shown above. Thus, the line:

```
LS16FloFracPlain->/turing/projects/csar/autotest/gen3-data/labscale/LS16FloFracPlain
```

relates the LS16FloFracPlain symbol with a .tgz file located at :

```
/turing/projects/csar/autotest/gen3-data/labscale
```

and called LS16FloFracPlain.tgz. This .tgz file should be a tar/gzip of a directory called LS16FloFracPlain. Roctest will find the .tgz file, ungzip and untar it, and will then run the test located under the directory LS16FloFracPlain. If that directory already exists, it will be completely deleted before unzipping/untaring the archive for the current run. See section 4.1.5 for more detailed information on setting up test cases for Roctest.

4.1.3 JobRunData.conf Files

The JobRunData.conf file that must be included in the root directory of each problem set describes the overall test run to be made. It contains the parameters needed to pass to the pj_all script so that the script can submit the batch job to the system. An example of this file is:

```
# pj_all Non-interactive mode (all command line arguments always required):
#

virtualComputeCPUs->16
problemName->LSFloFracPlain
pandaServers->0
totalPhysicalCPUs->16
wallClockMinutes->20
fluidSolver->Rocflo
fluidAlone->n
solidSolver->Rocfrac
solidAlone->n
burnSolver->RocburnAPN
systemTimeStep->5.0E-06
zoomFactor->0.0
maxPCIterations->1
endTime->0.0005
outputInterval->0.0001
jobName->LSFloFracPlain
restartFlag->0
jobCompleteString->GENX System Time Step :  100
diffFileList->LS16FloFracCharmDiff.txt
```

All but the last two of these parameters are passed to the pj_all script in order to submit the job. They may be included in this file in any order. The last two in the example above allow Roctest to verify run completion and to regression compare the results. The parameters will be discussed below. It is not necessary to know how to run pj_all to use Roctest. You just need to supply the path to pj_all, and to supply this JobRunData.conf file with its parameters. pj_all To learn more

about the pj_all script, and interactive job submission with it, see the Rocstar3 User's Guide. Most of these parameters are actually used to change values in the RocstarControl.txt file, which is also described in the Rocstar3 User's Guide.

- virtualComputeCPUs is used to indicate how many virtual processors to use during a charm-based run. This is generally set to be the same as totalPhysicalCPUs. REQUIRED

- problemName is a string label that can be used by pj_all to find the problem directory if the problem set is in a very specific directory. Not really used in Roctest, and is usually set the same as the jobName parameter. REQUIRED

- pandaServers indicates how many panda servers to use if the run is using Rocpanda for output. Indicate zero (0) if panda is not being used. REQUIRED

- totalPhysicalCPUs is the total actual number of physical CPUs needed for the job. REQUIRED

- wallClockMinutes is how many wall clock minutes to ask for in the batch job. Note that 15 minutes will be subtracted from this number by pj_all to account for final output time, so this parameter should be the amount of runtime you actually want plus 15. REQUIRED

- fluidSolver is a string indicating which fluid solver to use (Rocflo or Rocflu). REQUIRED

- fluidAlone is a single character that indicates whether this is a fluid-only job or not ('y' or 'n'). REQUIRED

- solidSolver is a string indicating which solid solver to use (Rocfrac or Rocsolid). REQUIRED

- solidAlone is a single character that indicates whether this is a solid-only job or not ('y' or 'n'). REQUIRED

- burnSolver is a string indicating which burn model to use (RocburnAPN or RocburnPY). is a single character that indicates whether this is a fluid-only job or not ('y' or 'n'). REQUIRED

- systemTimeStep should be a floating point number describing the system timestep to use for this run. REQUIRED

- zoomFactor is a parameter setting the time zooming factor for the run. If it is set to 0.0, no time zooming and no Rocprop surface propagation will occur. If set to 1.0, no time zooming will occur, but in a fluids run, Rocprop will propagate the surface. If greater than 1.0, then time will be 'zoomed' by the factor chosen. REQUIRED

- maxPCIterations indicates the maximum number of predictor-corrector iterations to use in the time stepping algorithm. REQUIRED

- endTime is a floating point number describing the computational ending time desired for the simulation. The number of timesteps run will be this value divided by the system timestep, unless the wallclocktime is exceeded. REQUIRED

- outputInterval is a floating point number for how often rocstar should output the solution. REQUIRED

- jobName is a string used to uniquely identify the problem in the batch system. pj_all appends the number of processors to the end of it, so you don't have to include this. Also, restrict the number of characters to 14 or fewer, or the jobname may overrun the available space in the job listing, and the system will truncate the job name, and make it impossible for Roctest to find it. REQUIRED

- restartFlag is an integer flag that tells pj_all whether this run is a restart of a previous job or not. If the flag is set to 0, this will be assumed to be a new run starting from time zero. If it is set to 1, the run will be assumed to be a restart, and the start time will be read from the restart.txt file from the previous run. REQUIRED

- jobCompleteString is a string that is used by Roctest to see if the job ran to completion. It should be some string near the end of the screen dump file that will always be there if the run completed. In the example, 100 system timesteps will be run, so the string "GENX System Time Step : 100" will be searched for by Roctest, and if found, Roctest will assume that the run completed.

- diffFileList is the name of a file that contains pairs of files or directories to compare using the difftool setup in using the Roctest.conf file. In the example, the file name is "LS16FloFracCharmDiff.txt", and this file is assumed to be in the "conf" directory, the path for which is given in Roctest.conf. See section 4.1.4 for the format for this file.

4.1.4 Difference File List Format

Roctest uses files that provide sets of three input values for submission to Rocdiff:

<filename1>, <filename2>, <outputfilename>

or

<directory1>, <directory2>, <outputfilename>

As of the writing of this manual, Rocdiff only understands the first format, but will soon be upgraded to understand the second.

Format 1: compare two specific files, and write the results into a file. The files must be .hdf files output by Rocstar3, and the output file will be a text table of the results of comparing all

data in each of these files with all available comparison measures (see the Rocdiff manual for available comparison measures). Roctest will examine the <outputfilename> file to assess whether the regression was successful or not.

Format 2: compare the files in two specific directories, and write the results into a file. A listing of files in directory 1 will be made, and all files with the same names in directory 2 will be compared with the files in directory 1. Any extra files in directory 2 will be ignored. Missing files in directory 2 will cause errors. The output file will be a text table of the results of comparing all data in each of the compared files with all available comparison measures (see the Rocdiff manual for available comparison measures). Roctest will examine the <outputfilename> file to assess whether the regression was successful or not. An example file using the directory comparison method might look like:

```
/Gold/Rocflo/, /Rocflo/Rocout/, /RocfloDiff.txt
/Gold/Rocfrac/, /Rocfrac/Rocout/, /RocfracDiff.txt
/Gold/RocburnAPN/, /RocburnAPN/Rocout/, /RocburnDiff.txt
```

This file directs Roctest to use Rocdiff to compare three sets of directories, and to write the difference results into the three files indicated as the last parameter on each line. All paths are assumed to be relative to the problem directory of the dataset being run. For example, files in /Gold/Rocflo/ will be compared with newly created files from the run that are created in /Rocflo/Rocout/. The files in Gold don't change (unless changed by the user directly), but the files in Rocflo/Rocout change every night.

4.1.5 Setting up Roctest Test Cases

Setting up a Rocstar3 test case for Roctest starts out in exactly the same way as building a Rocstar3 dataset for a standalone run. This document will not attempt to describe how to set up a Rocstar3 dataset. For that information, see the Rocprep User's Guide, and the Rocstar3 User's Guide. Assuming you have already setup a Rocstar3 dataset, in order to make it runnable with Roctest, you add two items to it: a JobRunData.conf file, and optionally, a "Gold" directory that will hold the regression comparison files. Thus, a complete Roctest-compatible dataset starts out looking like the following, with the items added for Roctest in Bold:

LS16FloFracPlain
- **Gold**
- **JobRunData.conf**
- Rocflo
- Rocman
- RocburnAPN
- RocfloRocfrac.log
- RocstarControl.txt
- RocburnPY
- Rocfrac

The JobRunData.conf file is described in section 4.1.3. "Gold" is a directory, and is often set up as:

Gold
    Rocflo
    Rocfrac
    Rocburn

where under each of the Roc* subdirectories are the files that are considered to be the "gold standard" for that run to be compared with Rocdiff. See section 4.1.4 for the format of the difference list file needed to direct comparison of the files in these Gold directories with files that will be produced by Rocstar in the main problem directories.

Once this problem directory is set up correctly, and all files are present, the entire directory should be tar'd and gzipped into a .tgz archive, from just above the top level directory. So when done, you might have a structure similar to:

labscale
    LS16FloFracPlain
    LS16FloFracPlain.tgz

Once the .tgz file is available, the LS16FloFracPlain directory may be removed. When Roctest is run, and unzip/untars the LS16FloFracPlain.tgz file, the LS16FloFracPlain directory and it's files will be recreated.

Note: currently Roctest demands a .tgz file, and not other typical extensions such as .tar.gz. This may be relaxed in the future, but currently the file extension MUST be .tgz.

4.1.6 RoctestInput.txt Test Driver File

When Rocbuild drives Roctest on a platform, it uses the –f flag and a file called RoctestInput.txt to list the executables and test cases per executable. each line in the file contains the fully qualified path to an executable to test, and then a list of test case labels following on the same line. For two executables on the turing apple cluster, with three test cases per executable, the file looks like:

```
/turing/projects/csar/autobuild/rocstar3/build_3.0.0-54/gen3charm/bin/rocstar  LS16FloFracCharm
SB4ndCCCharm LS16FluFracCharm
/turing/projects/csar/autobuild/rocstar3/build_3.0.0-54/gen3plain/bin/rocstar  LS16FloFracPlain
SB4ndCCPlain Arienti4Plain
```

Note that this is actually only two lines in the file. The first executable to test is:

```
/turing/projects/csar/autobuild/rocstar3/build_3.0.0-54/gen3charm/bin/rocstar
```

and three test case runs are requested:

```
LS16FloFracCharm SB4ndCCCharm LS16FluFracCharm
```

The second line defines a similar set of tests for a second executable.

Since the –f flag is a command line parameter for Roctest, there is no reason that a user of Roctest could not assemble their own file of executables/tests and run Roctest from the command line using the –f flag. The file name can be any valid filename, and does not have to be RoctestInput.txt.

4.2 Output File Structure

Currently, Roctest has no direct output other than output to the screen, which actually gets piped into the Rocbuild log file for the system (unless Roctest is run standalone from the command line). A significant amount of screen information is output, but an important line for each test case will look like:

```
Job LSFloFracCharm RAN TO COMPLETION with
/turing/projects/csar/autobuild/rocstar3/build_3.0.0-
50/gen3charm/bin/rocstar; Rocdiff regression comparison  NOT YET IMPLEMENTED
for /RocfloDiff.txt.
```

This line records the test name, whether it ran, the exact executable used, and the Rocdiff results (this message is under construction at the present time). This line is extracted from the log for each test run and used in e-mails and on the web page for Roctest.

Other output of interest, especially in the case of failures will be the screen dump files and Rocdiff comparison files, which will be left in the test case directories after the runs. These files will remain available until the next Roctest run, at which time their entire directories will be deleted, and the results destroyed. Since this happens every 24 hours currently, there is ample time to examine and/or archive the results if needed.

5.0 Examples and Test Problems

Roctest is currently being developed and run on the CSE turing apple cluster. Roctest is installed in:

```
/turing/projects/csar/RocBT/Roctest
```

and the test cases are set up in:

```
/turing/projects/csar/autotest
```

Available test cases are:

16 processor Labscale

Rocflo/Rocfrac fully coupled case

Rocflu/Rocfrac fully coupled case

2 processor "Spongebar"

Rocflo/Rocfrac fully coupled case

Rocflu/Rocfrac fully coupled case

32 processor Coupled Shock (Arienti) problem

Rocflo/Rocfrac fully coupled case

These test cases are installed under:

```
/turing/projects/csar/autotest/gen3-data/labscale

   LS16FloFracCharm.tgz
   LS16FloFracPlain.tgz
   LS16FluFracCharm.tgz

/turing/projects/csar/autotest/gen3-data/spongebar

   SB4ndCCCharm.tgz
   SB4ndCCPlain.tgz

/turing/projects/csar/autotest/gen3-data/Arienti

   Arienti4Plain.tgz
```

Note that one test case CAN NOT be run with multiple executables. Since all test cases are submitted as batch jobs all at once, each combination of executable and test case needs to be unique. Thus, the `LS16FloFracCharm.tgz and LS16FloFracPlain.tgz` cases are essentially identical datasets except for their JobRunData.conf files and their gold results. One will be run with a "plain" rocstar build, one with a build with charm, hence the names.