Center for Simulation of Advanced Rockets

University of Illinois at Urbana-Champaign

Rocbuild User's Guide

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

Title:	Rocbuild User's Guide
Author:	Mark Brandyberry
Subject:	This document describes the use of the Rocbuild automated build program.
Revision:	Initial (0) Update (1)
Revision History	Revision 0: Initial Release Revision 1: Updated for multithreading and Roctest integration
Effective Date:	03/17/2005

1.0 Introduction

Rocbuild is a script-based program (Perl) designed to check out source code from a CVS repository, tar the source code into a single file, and scp the source code to one or more remote platforms. It then constructs a dynamic Perl script that is executed on each remote platform, which configures build directories on the remote machine, un-tars the source code, builds the source code, and optionally runs the Roctest program to regression test the new build. Rocbuild also sends e-mails to configured developers after each build with short messages about the results of the build. Rocbuild uses the Linux cron facility to run periodically. The Perl code is fairly platform-neutral, but has only been tested on Linux.

2.0 Purpose and Methods

Rocbuild's purpose is to allow automated building of one or more code modules on a periodic basis, on one or more platforms. It is highly configurable, and can access multiple modules from CVS, build them on multiple platforms (each module may be built on different platforms if desired), and then notify a list of e-mail addresses (different for each module) about the results of the build(s). Rocbuild runs on a Linux platform, and uses the native CVS client in Linux to check out code modules from the CVS repository. Multiple repositories may be accessed to obtain the code modules (one repository per code module). It then uses scp to transfer the code modules to the remote platform(s), and ssh to run the build script on each platform. No passwords are used, so the remote accounts must be configured with an SSH key for the build machine with no passphrase (see section 3 for more information). Rocbuild uses linux-specific "mail" program syntax to e-mail developers the results of each build. In the future this could easily be extended if needed to run on other platforms. After all builds and tests are run, Rocbuild has the capability to transfer the log files from the builds to a web-server directory, and produces an HTML file that allows developers to access the results and the log files for each build from a web browser each day. A separately linked HTML page is produced to list the results of the Roctest runs. Rocbuild constructs this Roctest web page, so it will be described in this manual.

3.0 Building and Running

Since Rocbuild is written in Perl, it is an interpreted code, and does not have to be "built". Currently, all files required for Rocbuild are in the Rocbuild/source directory, except for the properties class, which is in the Common directory. The code is checked into CVS in the Rocstar/RocBT/Rocbuild directory and in the Rocstar/RocBT/Common directory. Within the Rocbuild directory, the conf directory is also checked in with all the configuration files currently in use. These configuration files are only useful as-is on the lovell.csar.uiuc.edu autobuild server, since they contain path information for that machine, but they could easily be changed to run on another machine. The structure of the source code is given below:

Common/Properties.pm

Rocbuild/source/Rocbuild.pm
Rocbuild/source/RocbuildProperties.pm
Rocbuild/source/TargetProperties.pm
Rocbuild/source/JobProperties.pm
Rocbuild/source/BuildScript.pm
Rocbuild/source/LinuxMailer.pm
Rocbuild/source/LogAnalyzer.pm
Rocbuild/source/WebFileBuilder.pm

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

The file Rocbuild.pm is the only one of the files that is executable. It is run as ./Rocbuild.pm, usually with no command-line arguments. (There is actually a small shell script that runs Rocbuild – see section 3.3.) A configuration filename may be given on the command-line if the default file named "../conf/Rocbuild.conf" is not to be used (see section 4.1 for the contents of Rocbuild.conf). The filename to use is given after the '-c' command line switch. If a conf filename is given on the command line, it may have any name, and should either be in the same directory with Rocbuild.pm, or the full path must be given, such as:

Rocbuild.pm –c <theConfFile.conf>

The only other command line switch available with Rocbuild is '-test'. This switch truncates some of the functionality inside Rocbuild to facilitate testing. It is usually used in conjunction with –c to specify a test version of the Rocbuild configuration file, and may be either the first or third parameter on the command line, as:

Rocbuild.pm –c <theConfFile.conf> -test

Rocbuild.pm –test –c <theConfFile.conf>

See section 3.4 for further information on running Rocbuild in test mode.

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

Rocbuild/checkout ->where code modules are checked out to.
Rocbuild/conf -> where all configuration files are kept (see section 4.1)
Rocbuild/logs -> all log files are placed in this directory
Rocbuild/scripts -> copies of all auto-generated scripts are placed here
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 Rocbuild accesses one or more remote computers using SCP and SSH, it is necessary to configure the remote machines to accept connections from the machine running Rocbuild without a password or SSH key passphrase. This configuration , and security of the autobuild server is discussed in section 3.1. Section 3.2 discussed setting up the Autobuild server to enable access to the CVS repository without entering a password. Rocbuild is run at the command line or by using the cron daemon. Setting up the cron daemon is discussed in section 3.3. Finally, using the Rocbuild test mode is described in section 3.4.

3.1 Autobuild Server Build Account and Remote Target Machine(s) SSH Setup

The account on the build machine (currently lovell.csar.uiuc.edu) is called "csarbld". The csarbld account has been configured with an SSH key in it's .ssh directory, stored in the identity.pub file. This public key has been generated (using ssh-keygen) without a passphrase. This is normally poor security, but is acceptable in this case to facilitate unattended logons from the automated csarbld account. To allow these unattended logons, on each remote machine, the key in the identity.pub file must be copied into the authorized_keys file in the remote machine's .ssh directory, in the account that will be used for the logon. The account used for logon is specified in the targetaccount parameter in the Module - Target Platform Configuration File that is described in section 4.1.4. Once the identity.pub key is placed in the authorized_keys file on the correct account, then the csarbld user will be able to log into that machine to perform the builds.

3.2 Autobuild Server CVS Account Setup

The csarbld CVS account should be set up in the same way that any other cvs user would be set up. Make an empty .cvspass account in the home directory for csarbld. Then run cvs login as: cvs –d :pserver:cvsaccount@galileo.csar.uiuc.edu:/cvsroot login. Then enter the password for the cvs account that is being used (replace "cvsaccount" with a valid cvs user name in the command above). Do this for each CVS user/machine configuration that will be used. (i.e, if there is ever a second CVS server to be accessed, the account/server login would have to be run again to add the account/server password to the .cvspass file). The account to use is specified in the "checkoutaccount" parameter in the Code Module Configuration file discussed in section 4.1.3.

3.3 Autobuild Server Cron Daemon Setup

The Cron daemon runs the Rocbuild.pm script nightly (currently at 1:00 am). This is controlled by entries in the csarbld crontab file which is located in the /var/spool/cron directory. The relevant entries in the crontab file are the MAILTO=   line, and the line:

```
0 1 * * * /home/csarbld/nightlybuild
```

The values here represent the zeroth minute of the first hour of *any day of *any month of *any year. The command `/home/csarbld/nightlybuild` is then executed. Nightlybuild is a short shell script:

```
#!/bin/tcsh
mail -s BUILDSTART mdbrandy@uiuc.edu < test.txt
cd /home/csarbld/Rocstar/RocBT/Rocbuild/source/
./Rocbuild.pm
```

which just emails the (current) Rocbuild author of the start of the build cycle, changes to the source directory for Rocbuild and runs it. Note: DO NOT EDIT THE buildgenx crontab file by hand! There is a program called "crontab" used to set the files up. Read the man page on crontab to use it.

3.4 Running Rocbuild in Test Mode

Rocbuild's test mode is designed to do a few things differently, such as not writing test logs into the real webserver log directory, and to not checkin updated version files to CVS. By supplying the –test switch, and using the –c switch to supply an alternate Rocbuild.conf file, a completely separate test environment may be set up that will use virtually the same code as a normal Rocbuild run, but can be configured to write logs and scripts to directories different that the real logs and scripts, as well as check out different modules, run different build commands, etc. See section 4 for what files are required by Rocbuild. Essentially, the test mode just helps by not writing log files where you don't want them on the web server, and by not checking unwanted build number updates into CVS. Having a separate set of configuration files for testing is important to couple with this switch.

3.5 Threading in Rocbuild

Rocbuild has evolved to use Perl Threads to enable reasonable turnaround times. Rocbuild can be configured to checkout multiple codes, and take each code, transfer it to multiple platforms, and build it multiple ways on each platform. Since building the CSAR rocstar code once can take from 15 minutes to two hours (depending upon platform and options), running these builds in serial is not feasible. Thus, the checkout of each code occurs in a separate thread, and the transfer and building of each code module on each platform occurs in a separate thread. Thus, if there are three codes, and four platforms for each code, twelve threads will be spawned and all modules on all platforms will be checked out and built simultaneously. Note however, that for a single module on a single platform, the builds for that module occur in series.

4.0 Input and Output (User Interface)

Rocbuild uses four different types of input files: Global configuration (usually Rocbuild.conf, but overridable with the –c switch), a build list ("Tobuild.conf") file that has the list of modules to build, and pairs of files: "modulename.conf" and "module_targetname.conf" for each module and module/target platform pairs to be built. These files are described in section 4.1. In this context, a "modulename" is a code module such as "rocstar3". A targetname is a computing platform to build on, such as "turing". Output files from each run of Rocbuild are log files for each module/platform pair, script files for each module/platform pair, and a set of modulename_lastbuildnumber files maintained in the logs directory.

4.1 Input File Structure

The global configuration, modulename.conf, and module_targetname.conf files are all examples of a Rocbuild "properties" file, and they all have the same structure. A Rocbuild 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 conf 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. All of the configuration files should be in the "conf" directory described in section 3.0.

4.1.1 Global Configuration File

The global configuration file controls global aspects of the program, and must be named Rocbuild.conf, and be stored in the conf directory if it is to be found automatically upon startup of Rocbuild. If this is not desired, then the global configuration file name and path should be given on the command-line to Rocbuild.pm. In either event, the format of the file is the same. 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->/home/csarbld/Rocstar/RocBT/Rocbuild/
confPath->/home/csarbld/Rocstar/RocBT/Rocbuild/conf/
logsPath->/home/csarbld/Rocstar/RocBT/Rocbuild/logs/
scriptsPath->/home/csarbld/Rocstar/RocBT/Rocbuild/scripts/
checkoutPath->/home/csarbld/Rocstar/RocBT/Rocbuild/checkout/
buildConfFile->Tobuild.conf
saveLogsToWeb->TRUE
webServer->copernicus.csar.uiuc.edu
webRoot->/servers/www/csar/Rocbuild
logWebPath->/servers/www/csar/Rocbuild/logs
webAccount->mdbrandy
webLogTitle->CSAR Rocbuild Automated Build Status
webLogFileName->index.html
```

- RocHome is the full path to the base Rocbuild 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. REQUIRED

- scriptsPath is the full path to the scripts directory where all generated build scripts will be stored. Does not HAVE to be in Rocbuild directory. REQUIRED

- checkoutPath is the full path to the checkout directory where modules will be checked out from CVS. Does not HAVE to be in Rocbuild directory. REQUIRED

- buildConfFile is the file name of the configuration file that holds the names of the code modules to build (see section 4.1.2). Must be in the conf directory referenced by the confPath parameter above. REQUIRED

- saveLogsToWeb is a boolean switch (either "TRUE" or "FALSE") which controls whether the log files are transferred to a web server or not, and whether the HTML is generated to access the build results and logfiles. REQUIRED

- webServer is the name of the web server machine to transfer the logs to. REQUIRED ONLY IF saveLogsToWeb is TRUE.

- webRoot is the full path to the root directory for Rocbuild – it is where the "webLogFileName" file will be transferred. REQUIRED ONLY IF saveLogsToWeb is TRUE.

- logWebPath is the full path on the web server where log files are to be stored. Note that this is assumed to be a subdirectory below the directory in which the HTML to access the build results and logfiles is kept. REQUIRED ONLY IF saveLogsToWeb is TRUE.

- webAccount is the account name to use to transfer the log and HTML files to the web server under. Note that the web server machine must be set up to accept SCP connections from the autobuild machine as described in section 3.1, and the account used must have write access to the web directory. REQUIRED ONLY IF saveLogsToWeb is TRUE.

- webLogTitle is the text title to be used for the main Rocbuild Web status page. It can be any single line of text. REQUIRED ONLY IF saveLogsToWeb is TRUE.

- webLogFileName is the name of the file on the build server to be transferred to the web server. Note that this file will be created in the logs directory, and is assumed to be there to transfer to the webserver. REQUIRED ONLY IF saveLogsToWeb is TRUE.

4.1.2 Module Build List Input File

The name of the Module Build List Input File may be any valid filename, but it must be listed as the buildConfFile parameter in the Global Parameters File. The suggested name is "ToBuild.conf". This file is not in "Properties" format. An example file is given below.

```
gen3.conf
gen2_5.conf
rocfluidmp.conf
```

Each line in the file refers to a different code module to build. Each module to build is actually defined in its own conf file, and thus this file lists the filenames of the conf files that are to be processed by Rocbuild. These names may be anything, but it is suggested that they be modulename.conf for clarity. See section 4.1.3 for description of the contents of the modulename.conf files. Note that these file references may be in any order, but they will be processed in order, so if there are any dependencies between the modules, they should be placed in the appropriate sequence.

4.1.3 Code Module Configuration File

The code module configuration file contains all the parameters needed to check out the code from CVS, update the build number file, define the platforms on which to build, and define who to e-mail results of the build to. An example of a code module configuration file is provided below, and each parameter in the file is described following the example.

```
#->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

buildmodule->genx/Codes
altModuleName->rocstar3
pserver->galileo.csar.uiuc.edu
checkoutaccount->csarbld
repository->/cvsroot
cvsopts->-d rocstar3
version->3.0.0
versionFile->rocstar3/Rocman/include/configf90.h
numPlatforms->4
system1->gen3_turing2.conf
system2->gen3_cu.conf
system3->gen3_tungsten.conf
system4->gen3_copernicus.conf
mailTo->mdbrandy@uiuc.edu, jiao@uiuc.edu, rfiedler@uiuc.edu, cmclay@uiuc.edu
```

- buildmodule is the CVS name of the module to check out and build. It must be the name of a module CVS recognizes with a command such as: cvs co genx/Codes. See the cvsopts parameter for options that can be added to the command. Note that you cannot use two modules that have the same module name with Rocbuild. Due to its multithreaded nature, the checkouts would conflict with each other, overwriting files and CVS directories, and unpredictable results would occur. The altModuleName parameter helps in this situation. REQUIRED

- altModuleName is a string (no spaces) that will be used as an alternate name for this module after checkout. As an example, the rocstar3 code is still checked out using genx/Codes, but using the cvsOpts parameter, is renamed rocstar3 during checkout. The altModuleName parameter allows this renaming within Rocbuild, even though the actual module being checked out is called genx/Codes. OPTIONAL

- pserver is the name of the machine where the CVS pserver is located for accessing the CVS instance. REQUIRED

- checkoutaccount is the account name of the CVS account to be used for checking out the code module. REQUIRED

- repository is the path on the pserver machine where the CVS repository is located. REQUIRED

- cvsopts is a list of options to be placed in the checkout command. It is used to build a command of the form: "cvs co –d rocstar3 genx/Codes" given the values in the file above. The value given for cvsopts will be placed between "co" and the module name in the checkout command. OPTIONAL

- version is the base version string to which the build number will be added for each build. It is used in several directory names. REQUIRED

- versionFile is the relative path below the base directory for the module being built where the version file for the module is kept. In this file should be a line of the form:

    - build = '0' ! to be edited by Rocbuild

  Rocbuild will replace the value between the single quotes with the current build number, and then check the file back into CVS. This is the general case. However, for the rocstar codes, another file format was constructed that has three lines:

  ```
  CHARACTER(*), PARAMETER :: CI_DATE ="$Date$ GMT"
  CHARACTER(*), PARAMETER :: VERSION = "3.0.0"
  INTEGER, PARAMETER      :: BUILD_NUM = 51
  ```

  Rocbuild will fill in the version and the build number, and the versionFile parameter is the relative path to the file. Note that in Rocbuild, if the buildmodule is either genx or rocstar3, this new config file format is hardcoded to be assumed. This should be fixed in a future release. REQUIRED

- numPlatforms is an integer telling Rocbuild how many remote computers this module will be built on (i.e., how many systemX parameter will be used). REQUIRED

- systemX is a set of parameters listing the target platform (remote computer) configuration files controlling the building of this module. If numPlatforms is "2", then there should be two entries: system1 and system2 (no space between "system" and the number). If numplatforms is 5, then it should go up through system5. There is currently no limit to the number of platforms supported (other than build time). There must be at least one. The module_targetname.conf files that are referenced here can have any name, but it is suggested that they be named as "modulename_targetname.conf" for clarity. These file should be in the conf directory described in section 4.1.1. system1 is REQUIRED. Others are REQUIRED only if numPlatforms > 1.

- mailTo is a comma-delimited list of e-mail addresses that will receive an e-mail for each platform on which this module is built, with a subject of the form:

```
Autobuild run for rocstar3 on turing-new.cse.uiuc.edu: All builds SUCCEEDED!
```

The body of the message will contain messages of the form:

```
 ******gen3_PLAIN BUILD SUCCESSFUL(0)
 ******gen3_REMESH_CHARM_LIBSUF BUILD SUCCESSFUL(0)
 ******gen3_DEBUG BUILD SUCCESSFUL(0)
 ******gen3_PEUL_PLAG BUILD SUCCESSFUL(0)
 ******gen3_LIBSUF BUILD SUCCESSFUL(0)
******gen3_charm BUILD SUCCESSFUL(0)
******gen3_REMESH_CHARM BUILD FAILED(512)
```

There will be one of these success or failure lines for each target executable checked during the build on each platform. See section 4.1.4 for how different executable results are checked while building a module. The number in parentheses after each build is the return code from the make utility. Zero means no errors. OPTIONAL

4.1.4 Module - Target Platform Configuration File

The module-target platform configuration files control, for each module, how it is built on each platform. An example of a code module configuration file is provided below, and each parameter in the file is described following the example. The example shown is the file "rocstar3_turing.conf", which shows the building of the rocstar3 module, being built on the turing computer platform. This example shows building rostar3 with seven different sets of parameters, demonstrating the make flexibility of Rocbuild.

```
#->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 = SIGN!
#->Example:
#->      paramname->/usr/bin/perl

targetaccount->mdbrandy
targetsystem->turing-new.cse.uiuc.edu
rootdir->/turing/home/mdbrandy/
targetdir->/turing/projects/csar/autobuild/rocstar3/
makefile->rocstar3/
makeline1->make REMESH=1 CHARM=1 PREFIX=../gen3remcharm
makeline2->make REMESH=1 CHARM=1 clean
makeline3->make PREFIX=../gen3plain
makeline4->make clean
makeline5->make REMESH=1 CHARM=1 LIBSUF=a PREFIX=../gen3staticremcharm
makeline6->make REMESH=1 CHARM=1 LIBSUF=a clean
makeline7->make DEBUG=1 PREFIX=../gen3debug
makeline8->make DEBUG=1 clean
makeline9->make PEUL=1 PLAG=1 PREFIX=../gen3part
makeline10->make PEUL=1 PLAG=1 clean
makeline11->make LIBSUF=a PREFIX=../gen3plainstatic
makeline12->make LIBSUF=a clean
makeline13->make CHARM=1 PREFIX=../gen3charm
makeline14->make CHARM=1 clean
buildname1->gen3_REMESH_CHARM
buildname3->gen3_PLAIN
buildname5->gen3_REMESH_CHARM_LIBSUF
```

```
buildname7->gen3_DEBUG
buildname9->gen3_PEUL_PLAGbuildname11->gen3_LIBSUF
buildname13->gen3_charm
rocTestDir->/turing/projects/csar/RocBT/Roctest/source/
test1_1->LS16FloFracCharm
test1_2->SB4ndCCCharm
test1_3->LS16FluFracCharm
targetexe1->gen3charm/bin/rocstar
targetexe2->gen3plain/bin/rocstar
test2_1->LS16FloFracPlain
test2_2->SB4ndCCPlain
test2_3->Arienti4Plain
deleteAfterBuild->TRUE
```

- targetaccount is the computer account that should be used to scp the code to and to ssh into to run the build script. This account must be set up as described in section 3.1 for logging in without a password. REQUIRED

- targetsystem is the computer name of the computer where the code module will be built. REQUIRED

- rootdir is the full path where the source code will initially be SCP'd to on the target system. REQUIRED

- targetdir is the full path where the build will be set up and run. Under this directory, a build directory of the form: build_version-buildnumber will be made (ex: build_3.0.0-57). The source archive will be copied here, and unarchived. REQUIRED

- makefile is the relative path under the build_version-buildnumber directory where the main makefile for the build is located. Essentially, it is the place in the build directory structure where commands will be issued from the buildscript. REQUIRED

- makelineX is a series (one or more) of commands to be issued within the directory specified on the "makefile" line. These commands can actually be any valid shell command (they are just passed through to the system), but are usually make-oriented. In the example file above, makelines 1 through 14 show building rocstar3 seven different ways, cleaning the build directory between each build. If all is successful, in the end there will be 7 executables in the directories referenced in the PREFIX parameters on each build. Note that the PREFIX parameter is functionality available for rocstar3 builds only. For other modules that produce their executables in-place, multiple builds may still be made, but the clean commands will delete them after each build. This restricts the use of Roctest later to only the last executable built. There is no such restriction with rocstar3 due to the available PREFIX option. There is currently no limit to the number of makelines that may be used. The makelineX commands may be entered in any order, but they will be executed in numerical order by the integer in the "X" position. makeline1 is REQUIRED. All others are OPTIONAL.

- buildnameX is a series of labels to be used for builds. Since there is no way for Rocbuild to know from the makeline commands which commands are actually builds, the buildnameX lines are used to associate a label with a makeline, and to flag it as being a build. Thus, the line buildname3->gen3_PLAIN means that the makeline3 command is to be interpreted as a build line, and that the label used for it for reporting purposes will be gen3_PLAIN. This way, reporting on commands for directory cleaning or other utility operations will not be reported as builds and clutter the output. There should be one buildnameX parameter for each makelineX line that it is desired to label and interpret as a build. The X part should be the same for the buildname and makeline parameters to be associated. OPTIONAL

There are three other optional parameters that may be used in this file to interface with the Roctest program (see the Roctest User's Guide). These are:

- rocTestDir is the full path to the source directory for the Roctest code on the platform where the build is being made. An example is:

  ```
  rocTestDir->/turing/projects/csar/RocBT/Roctest/source/
  ```

- targetexeX is a series of parameters that list the executables that should be run with Roctest. The values of the parameters are the relative paths (relative to the makefile directory) to the actual executable files to be tested.

- testX_Y are the symbolic names of one or more tests to be run with the new builds. If rocTestDir is defined, then at least "test1_1" should be defined. The "X" portion refers to the executable number listed in the targetexeX parameter that contains the executable name to run with this test, and the Y portion lists the test number. An example is:

  ```
  test1_1->LS16FloFracCharm
  ```

  The symbolic name on the right side of this definition is a symbol defined in the Registered Tests file for the Roctest installation on the platform being run on. There can be as many testX_Y parameters as desired (test1_1, test1_2, test1_3, ….). All of these example problems that are labeled test1_Y will be run with the executable labeled targetexe1. See the Roctest User's guide for more information on setting up test problems.

- deleteAfterBuild is a parameter that tells Rocbuild whether to clean out the build directory after completing all it's builds and tests. On some systems with limited disk space, a single Rocstar3 set of builds can take up hundreds of megabytes, and thus it is prudent to delete the build when finished for the night. Unfortunately, that leaves only the log file as evidence for the build if there are problems. Set the parameter to TRUE if you want all the build files deleted each night. Set to FALSE (or don't include this parameter) if you want the builds left after Rocbuild completion. OPTIONAL

4.2 Output File Structure

There are five types of output files from Rocbuild: screen output, log files, script files, version files, and HTML files. These files will be described below.

4.2.1 Screen Output

Rocbuild writes a significant amount of status information to the screen during its run. Since it runs without an interactive login, the Cron daemon captures this screen output, and sends it to the e-mail account registered with Cron. See section 3.3 for Cron setup for Rocbuild. An example of a portion of the screen output is presented as Appendix A to this document. The screen output echos the input parameters for each job, as well as some other status messages.

4.2.2 Log Files

Log files have no set internal format – they are just a log of the screen dump from both standard output and standard error that come from the build. Each log is named for the module and target platform as: modulename_targetname_build_build_version-#.log, as in:

rocfluidmp_turing.cse.uiuc.edu_build_0.1.0-54.log

The log files are saved in the "logs" directory as specified in the Rocbuild.conf file. If requested (using the saveLogsToWeb = TRUE parameter in the Rocbuild file), they are also scp'd to a webserver directory for access over the web.

4.2.3 Script Files

Script files are generated for each module for each platform on which the module will be built. The scripts are all stored in the "scripts" directory as defined in the Rocbuild.conf file. The names of the script files are the same as the corresponding log file, but without the .log extension (see section 4.2.2). The scripts look similar to:

```
#!/usr/bin/perl
if (-e '/csar/autobuild/gen2_5/')
    {chdir('/csar/autobuild/gen2_5/')}
else
    {mkdir('/csar/autobuild/gen2_5/',0750);
     system("chmod 750 /csar/autobuild/gen2_5/");
     chdir('/csar/autobuild/gen2_5/');}
if (-e "build_2.0.0-77")
    {chdir("build_2.0.0-77");
     system("rm -rf *");}
else
    {mkdir("build_2.0.0-77",0750);
     system("chmod 750 build_2.0.0-77");
     chdir("build_2.0.0-77");}
system("mv /home/mdbrandy/gen2_5.tar .");
system("tar -xf gen2_5.tar");
system("gzip gen2_5.tar");
chdir("gen2_5/Solver/");
system("gmake -j 8 OSTYPE=Linux");
if (-e '/csar/autobuild/gen2_5/build_2.0.0-77/gen2_5/Solver/genx.x')
   {print "******/csar/autobuild/gen2_5/build_2.0.0-77/gen2_5/Solver/genx.x BUILD SUCCESSFUL\n";}
else
   {print "******/csar/autobuild/gen2_5/build_2.0.0-77/gen2_5/Solver/genx.x BUILD FAILED\n";}
chdir('/csar/autobuild/gen2_5/');
system("chmod -R 750 *");
```

Each file will have different paths, filenames, buildnumbers, etc., but will be of this form. For information on this file, see the Rocbuild Developer's Guide. Normally, a user of Rocbuild will never actually see this file.

4.2.4 Version Files

Each module that has ever been built will have a version file that is stored in the logs directory (as defined in Rocbuild.conf) which as a name of the form: modulename_lastbuildnumber. Each of these files has a single integer stored in it that is the number of the last build number for that module. Thus, each module may have a different build number. If this file is deleted, then Rocbuild will make it again the next time a module is built, and will start from build number 1. If it is desired to change the cycle of the build numbers, it is easy to edit this file, placing whatever integer is desired into the file. The next time Rocbuild builds this module, it will increment that integer by 1 and use that number as the current build number.

4.2.5 HTML Files

There are two HTML files currently written by Rocbuild: the build result page, and the Roctest result page. The name of the build results page is listed in the main Rocbuild Configuration file, and is currently called index.html so that it performs as the default page on the web server. The build result page lists the target platform, code module, and builds for each combination built during one full run of Rocbuild. Log files that are captured from the builds are copied to the web server, and the each target/module build is linked from its web page to that log file. Thus, the web page supplies quick information on which builds were successful or failed on each platform, and by clicking on the log link, a developer has quick access to the log file for the builds on any platform for a code module. Note that this log file will contain the results for all builds for that module on that platform, so there will, in general, be multiple build results per log file. This makes it somewhat difficult to search the logfiles. The easiest way to find the end of a failed build, in order to look through the error messages, is to search for the capitalized word "FAILED" in the log. This will generally take you directly to the line just after each failed build in a file, since the builds are separated by a status line.

On some platforms, the build separator lines are all buffered at the end of the file, so that they are useless for separating the builds in the log. Currently, this happens only on the CSAR Sun systems. It is not currently known how to fix this problem.

The second HTML page constructed is Roctest.html. This file logs the results of any Roctest runs requested. It currently has no parameter to set its name or title, but this will likely change. The Roctest.html page lists the target platform, executable, and results of the tests run with that executable. The actual log results of the tests are available in the Rocbuild log for the specific platform and module. At the top of the Rocbuild page, the Roctest.html page is linked so that the two types of results are linked, but not interspersed.

Currently, the URLs for the build and test pages are shown below, but these may change at any time.

Rocbuild:

http://www.csar.uiuc.edu/Rocbuild

Roctest:

http://www.csar.uiuc.edu/Roctest.html

5.0 Examples and Test Problems

The Rocbuild code does not have examples or test problems, since it is not a calculational code. Its input files are all stored in CVS in the conf directory under RocBT/Rocbuild. These conf files would serve as examples, but the data in them would change significantly (mostly path information) if Rocbuild were used to build the codes on another system.