JCSE Users Guide

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

Title:	JCSE Users Guide
Author:	Mark Brandyberry
Subject:	Documentation of the use of the JCSE automated build and test system
Revision:	Rev. 0
Revision History	Revision 0: Initial Release for JCSE 1.0
Effective Date:	12/15/2006

1.0 Introduction

The *JCSE* application is an automated build and test driver designed to operate on a single clustered machine to automate the build and regression testing needs of the *Rocstar* code. There are three software modules that comprise the *JCSE* dedicated build and test system. The java-based modules are included in the *JCSE* package.

JBuild: Automated build driver and build result analyzer. This software is written in Java.

JTest: Automated regression test driver and result analyzer. This software is written in Java.

Rocdiff: Utility to compare variables in *Rocstar* HDF files for "equivalence" using several criteria. This module is a utility written by C. McLay [McClay, 2005], and is used by the *JTest* system. This software is written in C++, and uses file I/O routines from the *Rocstar* software base in order to read the *Rocstar* output files.

The build and test program as currently used involves a dedicated build and test cluster configured specifically to build *Rocstar* in several different configurations, and then to regression test those builds. The Hilbert computational cluster is a small Linux-based computing cluster formed from cast-off hardware from the old Linux-based Turing cluster. There are 32 total processors in 16 nodes, and a login node with 4 processors and 4 GB of RAM. In addition, a Myrinet High Speed Interconnect (HSI) and a small (800GB) RAID array are provided.

This cluster is dedicated to a small user base, and will almost exclusively be used as a build and regression testing platform for the *Rocstar* computer code. The cluster uses the TORQUE job scheduling system in conjunction with the MOAB job submission optimizer. Reference [Brandyberry, 2006] describes the hardware and software environment for the Hilbert cluster in detail.

2 Purpose and Methods

The purpose of the *JCSE* application is to provide an integrated build and regression testing system that operates in a single environment in a comprehensive manner. It is currently configured to run on Linux, although, since it is written in Java, it could run on any platform on which the Java 5 jdk is available. The *JBuild* module checks out *Rocstar*, builds it per information in configuration files, analyzes the build logs, checks for required output files, and drives testing by starting the *JTest* module. *JTest* uses a set of preconfigured test archives and runs the executable(s) specified by *JBuild* with sets of test archives. *JTest* verifies that each test completes (by searching for a user-defined string in the job dump), and then calls *Rocdiff* to compare the numerical results for each test against a set of gold-standard output files for that test.

3 Building and Running

JCSE is distributed as a java .jar file called JCSE.jar, and requires no further building on any platform where a Java 5 runtime environment is available. The *Rocdiff* code should be built on

the target platform. See [McLay, 2005] for directions on building *Rocdiff*. If it should be necessary to modify applications in *JCSE*, see the Developer's guide for information.

3.1 *JBuild*

The *JBuild* program is written in Java 5, and is completely command-line and file driven. No GUI elements are used, so that the program will run and perform the same on any platform on which Java 5 is installed. However, the system issues Unix-specific commands to interact with external systems such as CVS, SSH, MOAB, and TORQUE. This makes the system Unix-specific at this time, as well as tied-in to the MOAB and TORQUE job control systems (*JTest* actually uses the MOAB and TORQUE systems for job submission). The *JCSE* system is installed on the Hilbert cluster at the following location: /`csar/software32/JBuild .`

The basic flow of control for the *JBuild* program is shown in figure 3.1. The Linux Cron utility is used to start a small script that then starts the *JBuild* program at the specified time. The script changes directories to the *JBuild* directory, and then uses the Java runtime interpreter to start the *JBuild* program with the following commandline:

```
java -classpath JCSE.jar:mail.jar:activation.jar:jdom.jar cse.JBuild.JBuild -v -c
conf/paths.xml
```

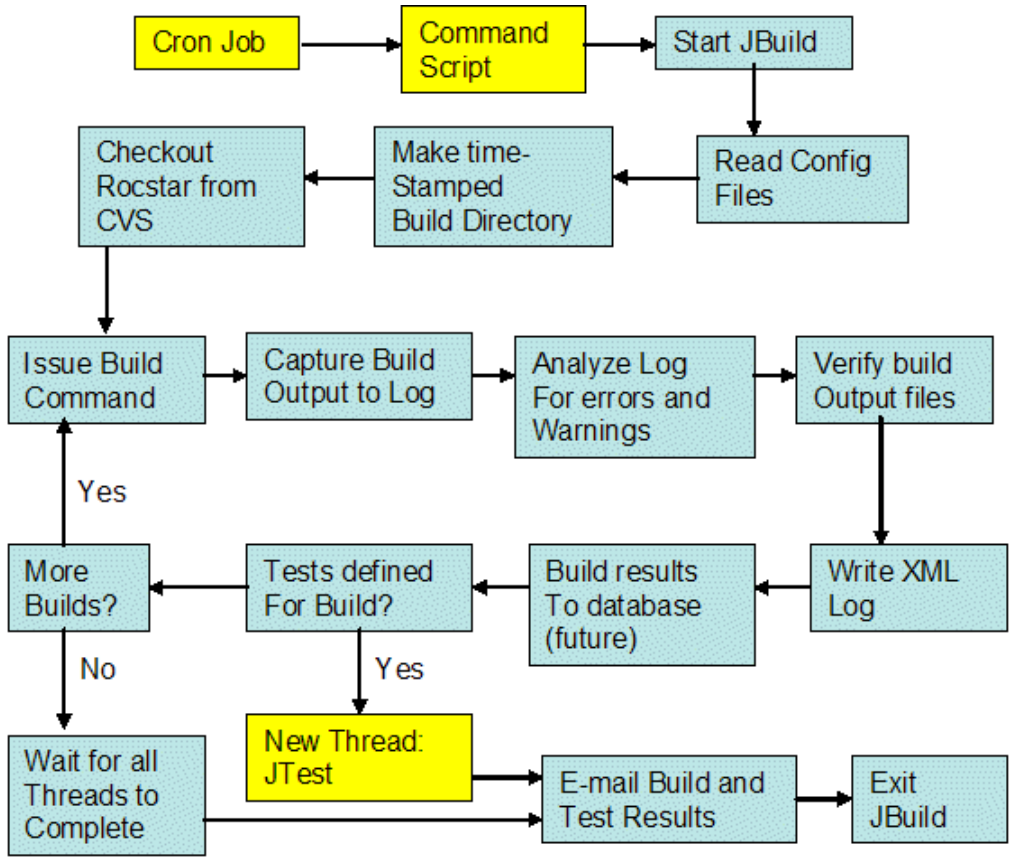

Figure 3.1 *JBuild* System Flow of Control

The first portion of this command simply starts the java interpreter with the –classpath switch to set the paths to the required libraries that are not part of the normal java runtime environment. These files are included in the *JCSE* distribution. The *JBuild* program is located in the *JCSE.JBuild* package inside the JCSE.jar file, and thus to load it into the Java interpreter, the path JCSE.JBuild.JBuild is required. The –v flag indicates verbose mode, where more text and error information is output. The –c flag is used with *JBuild* to provide the path to the main `paths.xml` configuration file (see section 3.2.1.1). This file contains the paths to all other required configuration files for *JBuild*.

Once all of the configuration information is available to *JBuild*, the main build loop can be started.

The build loop uses the same *Rocstar* source code base for each build, and thus depends on a makefile clean command to work effectively to clear out the previous build from the source code directories, as well as the PREFIX flag to place each executable in a unique place for later analysis and testing. Thus, the actual builds, as well as the makefile and the makefile clean commands are tested. The builds are performed in serial, and the source code directory cleaned between builds. The clean step is not shown in figure 3.1. The build command to be used for each build is provided in an XML file of commands that control the builds (`RocstarBuild.xml`). The path to and name of that build control file are provided in the `paths.xml` file that is provided to *JBuild* on the command line.

The basic build loop goes through the following steps:

- Changes directories to the source code directory, and issues the first build command from the `RocstarBuild.xml` file.

- As the build progresses, collect all of the screen output from the build into a log file.

- When the build has completed, analyze the log file for errors or warnings. A list of "error" or "warning" keywords is used to scan the file for lines that are likely to contain errors or warnings. *JBuild* then writes an XML file containing records for each error or warning found. Note that all lines back to the previous compiler execution statement are captured to give context for each error or warning. Also, any error or warning within a user-specified number of lines of another error or warning are grouped together into one record.

- Verify that all the required output files listed in the build verification file are actually present in the build output. The build fails if required files are not present. Verify that all optional build files listed in the build verification file are actually present in the build output. A warning is issued if optional files are missing, but the build is considered complete. These are normally utility codes that are not required for *Rocstar* to run but are normally built along with *Rocstar*.

- Write the XML log file for the build (actually occurs throughout the build process, but is finalized now).
- Using the log file(s), transfer the build results to the remote database (this functionality is planned, but not currently implemented).
- If testing is specified for this build, spawn a new thread of *JTest*, providing it the new executable path to test.
- Issue a clean command in the build directory.
- If there are further builds, issue the next build command.
- When there are no further builds, wait for all *JTest* threads (if any) to complete, and exit.

Note that for the above loop to work correctly for all but the first build, the clean command must operate correctly. Also, the *Rocstar* makefile allows specifying a "PREFIX" flag, so that that the build results may be placed in a directory away from the source directory. This flag is used so that the build results (i.e., the executable programs) are not "cleaned" between builds, so that each executable is available for testing as further builds are completed. Thus, while the builds are serial, the tests for the builds execute in parallel as each build is completed.

3.2 *JTest*

The *JTest* module is also a pure-java program that is located in the *JCSE.JTest* package. It can be used in two modes: as a separate, command-line driven program, distinct from running *JBuild*, or as an object constructed and used by *JBuild*. The main method in the *JTest* class either reads and then processes the command line as a standalone program, or *JBuild* constructs a *JTest* instance, and then calls the processCommandLine method on that *JTest* instance, sending it the same command line as would have been used in the standalone program call.

JTest is designed to test a single executable instance, running multiple tests on that executable. The command line takes the fully-qualified path to the executable to be tested. Several other flags, including the name of the *test group* to use for that executable may be (and sometimes must be) specified. A test group corresponds to a sub-directory in the main test directory (as specified in the `paths.xml` file discussed in section 3.2.1, and documented in appendix section A.1). That subdirectory must have test archives (discussed below) or links to test archives in it.

A valid *JTest* command line to execute *JTest* as a standalone program will look like the following:

```
java –classpath JCSE.jar cse.JTest.JTest [-v] [-c configfile]
<-l label> <-r executable> <testgroup1> [testgroup2]…
```

The portions of this command line are:

- `java` : the Java interpreter.

- –classpath `JCSE.jar` : classpath flag to tell the interpreter where to look for the *JTest* program.
- `cse.JTest.JTest` : package-specified program invocation.
- `-v` : optional verbose flag.
- `-c configfile` : optional path to configuration file. Must be the absolute path, or relative to where *JTest* is run. It is assumed to be relative to the executable and located in `./conf/paths.xml` if not provided.
- `<-l label>` : A required text label for the testing set. Used as part of the test directory naming convention. “label” can be any string with no spaces.
- `<-r executable>` : Required absolute path to the executable to be tested.
- `<testgroup1> [testgroup2]…` : list of test groups to run. At least one test group is required. Others are optional. Separate by spaces, with no spaces in test group names. See section 3.2.2.1 for a discussion of a test group, and section 3.2.2.2 for a discussion of a test archive.

4 Input/Output

4.1 JBuild

Once the `paths.xml` file is loaded, several other configuration files are read. These files are:

- `paths.xml`: full paths to all the main directories needed to run *JBuild*. Also used by *JTest*.
- `tools.xml`: Full paths to tools used by *JBuild* and *JTest*, such as the CVS client, and the pj_all job submission script. Also used by *JTest*.
- `params.xml`: miscellaneous parameters, for the most part having to do with timeout settings for both *JBuild* and *JTest*.
- `mailParams.xml`: parameters used to specify mail server settings for sending developer notification messages.
- `RocstarBuild.xml`: This is the main file for defining what builds will be performed, whether they will be tested, how they will be checked, etc.
- Build Verification Files: There are several files of this type that define sets of output files from different builds that should be verified to exist after a build completes. Different build flags can produce different sets of files, so a unique verification specification is required for each build that has a unique file set. There are currently 3 files of this type.
- `webConnections.xml`: A prototype file for defining parameters to be used to submit build information to a centralized database. Currently read, but not functional.

- `webServerProperties.xml`: A prototype file for defining parameters to be used to define the server properties in order to submit build information to a centralized database. Currently read, but not functional.

See Appendix A for details on the contents of these files.

4.2 JTest

4.2.1 Test Groups

A test group is a set of test archives (see section 4.2.2 for details on test archives) contained in a specific, named directory. That directory must be located under the directory defined by the `datapath` key defined in the `paths.xml` file (see section A.1). The name of the directory must be the same as the testgroup name specified on the *JTest* command line. Each testgroup must have a unique name and a corresponding unique directory. Test groups are defined to group sets of tests that can be run using specific sets of *Rocstar* build flags, so that special testing setups are easy to develop. Generally, a test group directory will actually have links pointing to test archives in a test archive library. In this manner, a single test (located in the test library) may be associated with multiple test groups without having to duplicate it (they can be large) and without having to maintain the configuration of separate copies.

As will be discussed in section 4.2.2, a test archive is a tar'd, gzip'd archive containing a complete test dataset. Thus, a test group is basically a directory containing either a set of these .tgz files, or links to .tgz files. *JTest* then expands these archives into a specific test directory (constructed and time-stamped at runtime), and runs them with the specified executable.

4.2.2 Test Archives

A test archive is simply a tar'd, gzip'd archive containing a runnable *Rocstar* dataset. It must either have the extension `.tgz` or `.tar.gz`. No other extensions will be processed (e.g., you can't use .zip). See [Fiedler, 2006] for the actual contents of a *Rocstar* dataset. There must be several other items in a test archive, however, that are not in a normal *Rocstar* dataset:

- `JobProperties.xml` file: Controls the job submission by providing parameters required by the pj_all script used to submit *Rocstar* jobs. See section B.1 for a description of the JobProperties.xml file. *JTest* reads this file to format the command line to pj_all to submit each test job.

- `diffs.xml` file : definitions of comparisons to be made by the Rocdiff utility of "Gold standard" results against the test results produced by this test run. See section B.2 for a definition of this file.

- AttributeMetrics XML file(s): One or more metrics files to define which variables and what comparisons are to be made by the program. It allows defining less than, greater than, or equal to comparisons, and a success criteria for each comparison. See section B.3 for a description of this file.

- Gold directories: Usually called “Gold”, and containing subdirectories for each module for which gold standard results are supplied. Each subdirectory (often called “Rocflo”, “RocburnAPN”, “Rocfrac”, etc.) should contain .hdf result files for Rocdiff to compare with the corresponding results from the current test. Directory names are actually contained in the `diffs.xml` file described in section B.2.

4.2.3 Rocdiff

Use and architecture of the *Rocdiff* utility code is described in [McLay, 2005]. The utility is designed to do one thing: Take two HDF files (or two directories of HDF files) that are outputs from the *Rocstar* code and compare the data in them. The two files must have identical number of variables and identical mesh connectivity information, since *Rocdiff* will perform a point-by-point (spatial points) comparison of the data.

Rocdiff can currently calculate two difference metrics for variables in HDF files:

- Maximum difference
- Mean Square Error

The maximum difference is an absolute difference metric across all points for each variable, not scaled to anything. The mean square error is also not scaled, but is less sensitive to outlier values, since it is calculating an average error.

JTest uses *Rocdiff* as an external program, calling it with a command line set of parameters in order to compare the results of regression tests to the results stored in the Gold standard directories within each test archive (see section 4.2.2 for a description of a test archive). The output from Rocdiff is captured by *JTest* and analyzed based on the contents of the AttributeMetrics file discussed in section 4.2.2 and Appendix section B.3.

5.0 Examples and Test Problems

5.1 Rocbuild

On Hilbert, JCSE is currently configured in `/csar/software32/JBuild` as:

```
JBuild
     activation.jar
     conf
          attributeMetricsRocburn.xml
          attributeMetricsRocfrac.xml
          attributeMetricsRocflo.xml
          attributeMetricsRocsolid.xml
          attributeMetricsRocflu.xml
          paths.xml
```

```
            rocstarPlain.xml
            rocstarStatic.xml
            rocstarRocman3.xml
            rocstarStaticRocman3.xml
            webServerProperties.xml
            mailParams.xml
            RocstarBuild.xml
            tools.xml
            diffs.xml
            params.xml
            webConnections.xml

      JCSE.jar
      jdom.jar
      mail.jar
      runBuild.sh
```

All builds occur under a directory located at:

```
/csar1/autobuild/rocstar3
```

That directory contains all the date-stamped builds for *Rocstar*, both for JBuild runs, and for the mulit-platform Rocbuild code.

5.2 JTest

The JTest runtime directory on Hilbert is:

```
/csar1/autotest
```

In that directory is the structure:

```
autotest
      particleTests
      plainTests
            ACMFloPropRst.tgz -> ../testBank/ACMFloPropRst.tgz
            Arienti10nd.tgz -> ../testBank/Arienti10nd.tgz
            EPistonFluFrac.tgz -> ../testBank/EPistonFluFrac.tgz
            LSFloFrac.tgz -> ../testBank/LSFloFrac.tgz
            LSFluFrac.tgz -> ../testBank/LSFluFrac.tgz
            scaleFloSolid2.tgz -> ../testBank/scaleFloSolid2.tgz
      RM2Tests
            EPistonFluFrac.tgz
            LSFloFrac.tgz
            LSFluFrac.tgz
            scaleFloSolid2.tgz
      RUNDIR
           [Hundreds of test runs]
      staticTests
            .
```

```
      ACMFloPropRst.tgz -> ../testBank/ACMFloPropRst.tgz
      Arienti10nd.tgz -> ../testBank/Arienti10nd.tgz
      LSFloFrac.tgz -> ../testBank/LSFloFrac.tgz
      scaleFloSolid2.tgz -> ../testBank/scaleFloSolid2.tgz
testBank
      ACMFloPropRst.tgz
      Arienti10nd.tgz
      EPistonFluFrac.tgz
      LSFloFrac.tgz
      LSFluFrac.tgz
      scaleFloSolid2.tgz
turbulenceTests
      ACMFloPropRst.tgz -> ../testBank/ACMFloPropRst.tgz
      Arienti10nd.tgz -> ../testBank/Arienti10nd.tgz
      EPistonFluFrac.tgz -> ../testBank/EPistonFluFrac.tgz
      LSFloFrac.tgz -> ../testBank/LSFloFrac.tgz
      LSFluFrac.tgz -> ../testBank/LSFluFrac.tgz
      scaleFloSolid2.tgz -> ../testBank/scaleFloSolid2.tgz
  weeklyTests
      <empty>
```

As you can see from this list, the actual test archives are contained in the directories “RM2Tests” and “testBank”. Other test groups, such as staticTests or turbulenceTests pick tests from the testBank through links to the archive files. In this manner, duplication of test archives is avoided.

References

Brandyberry, M., *Hilbert Build and Test Cluster Operational Configuratio*n, University of Illinois, Revision 0, December 2006.

Fiedler, R., *Rocstar 3 Solid Propellant Rocket Simulation Software User’s Guide*, Center for Simulation of Advanced Rockets, University of Illinois, Revision 1, December 15, 2006.

McLay, C., *Rocdiff User’s Guide*, Center for Simulation of Advanced Rockets, University of Illinois, Revision 1, December 9, 2005.

APPENDIX A JBUILD INPUT FILE DETAILS

A.1 paths.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : RoctestPaths.xml
    Created on : January 17, 2006, 10:05 PM
    Author     : mdbrandy
    Description: Location of program and data
-->

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>Need base installation path and paths for various
directories</comment>
    <entry key="home">/csar/software32/JBuild/</entry>
    <entry key="confPath">/csar/software32/JBuild/conf/</entry>
    <entry key="logsPath">/csar/software32/JBuild/logs/</entry>
    <entry key="dataPath">/csar1/autotest/</entry>
    <entry key="testRunPath">/csar1/autotest/RUNDIR/</entry>
    <entry key="buildRootPath">/csar1/autobuild/</entry>
</properties>
```

A.2 tools.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : tools.xml
    Created on : January 17, 2006, 10:05 PM
    Author     : Mark Brandyberry
    Description: system tools to be used for various jobs
-->

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>Need base installation path and base dataset path</comment>
    <entry key="versionControl">cvs</entry>
    <entry key="jobScript">/home/mdbrandy/bin/pj_all</entry>
    <entry key="shell">/bin/tcsh</entry>
    <entry key="diffTool">/csar/software32/Rocdiff/Codes/rocdiff</entry>
    <entry key="SCCtool">/usr/bin/cvs</entry>
    <entry key="checkOutCommand">co</entry>
</properties>
```

A.3 params.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : params.xml
    Created on : January 17, 2006, 10:05 PM
    Author     : Mark Brandyberry
```

```
    Description: Misc JTest-wide parameters
-->

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>General parameters</comment>
    <entry key="groupWaitTime">43200000</entry>
    <entry key="testWaitTime">7200000</entry>
    <entry key="jobStartInterval">10000</entry>
    <entry key="jobPollTime">60000</entry>
</properties>
```

A.4 mailParams.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : mailParams.xml
    Created on : April 7, 2006, 3:15 PM
    Author     : Mark Brandyberry
    Description: Location of mail server and stuff
-->

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>Need mail server and sender e-mail</comment>
    <entry key="smtpServer">galileo.csar.uiuc.edu</entry>
    <entry key="fromEmail">mdbrandy@uiuc.edu</entry>
</properties>
```

A.5 RocstarBuild.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : RocstarBuild.xml
    Created on : January 17, 2006, 10:05 PM
    Author     : mdbrandy
    Description: Rocstar Build Data
-->

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>Need base installation path and paths for various directories</comment>
    <entry key="moduleName">rocstar3</entry>
    <entry key="mailResultList">mdbrandy@uiuc.edu,rfiedler@uiuc.edu</entry>
    <entry key="checkoutModule">genx/Codes</entry>
    <entry key="checkoutTargetFlag">-d</entry>
    <entry key="sourceDir">rocstarSource</entry>
    <entry key="checkoutFlags"></entry>
    <entry key="numBuilds">7</entry>
    <entry key="makeline1">gmake -j 4</entry>
    <entry key="prefix1">PREFIX=../gen3plain</entry>
    <entry key="exetarget1">bin/rocstar</entry>
    <entry key="label1">ROCSTAR_PLAIN</entry>
    <entry key="verify1">rocstarPlain.xml</entry>
    <entry key="output1">gen3plain</entry>
    <entry key="testline1">-v -c conf/paths.xml -l ROCSTAR_PLAIN plainTests -r</entry>
    <entry key="makeline2">gmake -j 4 DEBUG=1 ROCPROF=1</entry>
```

```
    <entry key="prefix2">PREFIX=../gen3debugrocprof</entry>
    <entry key="exetarget2">bin/rocstar</entry>
    <entry key="label2">ROCSTAR_DEBUG_ROCPROF</entry>
    <entry key="verify2">rocstarPlain.xml</entry>
    <entry key="output2">gen3debugrocprof</entry>
    <entry key="testline2">-v -c conf/paths.xml plainTests -l ROCSTAR_DBG_PROF -r</entry>
    <entry key="makeline3">gmake -j 4 PEUL=1 PLAG=1</entry>
    <entry key="prefix3">PREFIX=../gen3plagpeul</entry>
    <entry key="exetarget3">bin/rocstar</entry>
    <entry key="label3">ROCSTAR_PEUL_PLAG</entry>
    <entry key="verify3">rocstarPlain.xml</entry>
    <entry key="output3">gen3plagpeul</entry>
    <entry key="testline3">-v -c conf/paths.xml particleTests -l ROCSTAR_PART -r</entry>
    <entry key="makeline4">gmake -j 4 LIBSUF=a</entry>
    <entry key="prefix4">PREFIX=../gen3static</entry>
    <entry key="exetarget4">bin/rocstar_flo</entry>
    <entry key="label4">ROCSTAR_STATIC</entry>
    <entry key="verify4">rocstarStatic.xml</entry>
    <entry key="output4">gen3static</entry>
    <entry key="testline4">-v -c conf/paths.xml staticTests -l ROCSTAR_STATIC_FLO -r</entry>
    <entry key="makeline5">gmake -j 4 TURB=1</entry>
    <entry key="prefix5">PREFIX=../gen3turb</entry>
    <entry key="exetarget5">bin/rocstar</entry>
    <entry key="label5">ROCSTAR_TURB</entry>
    <entry key="verify5">rocstarPlain.xml</entry>
    <entry key="output5">gen3turb</entry>
    <entry key="testline5">-v -c conf/paths.xml turbulenceTests -l ROCSTAR_TURB -r</entry>
    <entry key="makeline6">gmake -j 4 TURB=1 PEUL=1 PLAG=1 DEBUG=1</entry>
    <entry key="prefix6">PREFIX=../gen3turb_part_debug</entry>
    <entry key="exetarget6">bin/rocstar</entry>
    <entry key="label6">ROCSTAR_TURB_PLAG_PEUL_DEBUG</entry>
    <entry key="verify6">rocstarPlain.xml</entry>
    <entry key="output6">gen3turb_part_debug</entry>
    <entry key="makeline7">gmake -j 4 ROCMAN=Rocman3</entry>
    <entry key="prefix7">PREFIX=../gen3rocman3</entry>
    <entry key="exetarget7">bin/rocstar</entry>
    <entry key="label7">ROCSTAR_ROCMAN3</entry>
    <entry key="verify7">rocstarRocman3.xml</entry>
    <entry key="output7">gen3rocman3</entry>
</properties>
```

A.6 Rocstar Example Build File Verification File

Currently there are three different verification files: `rocstarPlain.xml`, `rocstarStatic.xml`, and `rocstarRocman3.xml`. Each file contains a list of files to verify build correctly. The `rocstarBuild.xml` file documented in section F.5 specifies which of these file lists to use for each individual build. The `rocstarPlain.xml` file is provided below. The others are similar.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : rocstarPlain.xml
    Created on : February 16, 2006, 12:45 PM
    Author     : mdbrandy
    Description: Files to be checked in plain rocstar build
-->

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
```

```
    <comment>These files need to be present in a complete roctar build. Paths
relative to build output directory. required1 should be the main
executable</comment>
    <entry key="required1">bin/rocstar</entry>
    <entry key="required2">lib/libRHDF4.so</entry>
    <entry key="required3">lib/libRocblas.so</entry>
    <entry key="required4">lib/libRocburn.so</entry>
    <entry key="required5">lib/libRoccom.so</entry>
    <entry key="required6">lib/libRocface.so</entry>
    <entry key="required7">lib/libRocflo.so</entry>
    <entry key="required8">lib/libRocflo.so</entry>
    <entry key="required9">lib/libRocfrac.so</entry>
    <entry key="required10">lib/libRocin.so</entry>
    <entry key="required11">lib/libRocman.so</entry>
    <entry key="required12">lib/libRocmap.so</entry>
    <entry key="required13">lib/libRocmop.so</entry>
    <entry key="required14">lib/libRocout.so</entry>
    <entry key="required15">lib/libRocpanda.so</entry>
    <entry key="required16">lib/libRocprof.so</entry>
    <entry key="required17">lib/libRocprop.so</entry>
    <entry key="required18">lib/libRocsolid.so</entry>
    <entry key="required19">lib/libRocsurf.so</entry>
    <entry key="opt1">bin/addpconn</entry>
    <entry key="opt2">bin/hdf2plt</entry>
    <entry key="opt3">bin/hdf2vtk</entry>
    <entry key="opt4">bin/makeflo</entry>
    <entry key="opt5">bin/profane</entry>
    <entry key="opt6">bin/rfctest</entry>
    <entry key="opt7">bin/rfloprep</entry>
    <entry key="opt8">bin/rfluinit</entry>
    <entry key="opt9">bin/rflumap</entry>
    <entry key="opt10">bin/rflupart</entry>
    <entry key="opt11">bin/rfracprep</entry>
    <entry key="opt12">bin/rhpm</entry>
    <entry key="opt13">bin/rsolidprep</entry>
    <entry key="opt14">bin/surfdiver</entry>
</properties>
```

A.7 webConnections.xml

The following file is a prototype for future addition of web reporting capabilities to JBuild/JTest.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : webConnections.xml
    Created on : February 27, 2006, 1:05 PM
    Author     : Mark Brandyberry
    Description: Data for connecting to webserver/database
-->

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>Connection information</comment>
```

```
    <entry key="reportToDB">true</entry>
</properties>
```

A.8 webServerProperties.xml

The following file is a prototype for future addition of web reporting capabilities to JBuild/JTest.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : mySQLDBProperties.xml
    Created on : November 10, 2005, 12:05 PM
    Author     : mdbrandy
    Description: Location of webserver to post to
-->

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>Need protocol, servername and port</comment>
    <entry key="protocol">http</entry>
    <entry key="server">musgrave.csar.uiuc.edu</entry>
    <entry key="port">81</entry>
</properties>
```

APPENDIX B. JTEST INPUT FILE DETAILS

JTest also uses the paths.xml file described in section A.1 to define path information for the main directories used by the program. The command line defines most of the other information required to run tests for a specific build. The following files are required.

B.1 JobProperties.xml

This file is placed in EACH regression test problem set directory, with the parameters configured for that specific test. The parameters in this file are tied directly to parameters required by the pj_all job submission script supplied in the genx/Codes/utilities directory for submitting *Rocstar* jobs on various platforms, now including Hilbert.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : JobProperties.xml
    Created on : February 6, 2006, 1:50 PM
    Author     : Mark Brandyberry
    Description: Example Job Properties file that supports pj_all
-->

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>16 Job Properties required for pj_all</comment>
    <entry key="virtualComputeCPUs">16</entry>
    <entry key="pandaServers">0</entry>
    <entry key="totalPhysicalCPUs">16</entry>
```

```
    <entry key="wallClockMinutes">20</entry>
    <entry key="fluidSolver">Rocflo</entry>
    <entry key="fluidAlone">n</entry>
    <entry key="solidSolver">Rocfrac</entry>
    <entry key="solidAlone">n</entry>
    <entry key="burnSolver">RocburnAPN</entry>
    <entry key="systemTimeStep">5.0E-06</entry>
    <entry key="zoomFactor">0.0</entry>
    <entry key="maxPCIterations">1</entry>
    <entry key="endTime">0.0005</entry>
    <entry key="outputInterval">0.0001</entry>
    <entry key="jobName">LSFloFracPlain</entry>
    <entry key="restartFlag">0</entry>
    <entry key="jobCompleteString">GENX System Time Step :  100</entry>
    <entry key="diffDefs">diffs.xml</entry>
</properties>
```

B.2 diffs.xml

This file is also placed into each regression problem set to define what difference comparisons will be performed for this regression test.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : diffs.xml
    Created on : February 6, 2006, 1:50 PM
    Author     : Mark Brandyberry
    Description: Example Job result Directory Comparison Definitions.
                 All paths relative to run directory. This one is set up
                 for a fully-coupled flo/frac/APN run.
-->

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>pairs of directories to compare hdf files in.</comment>
    <entry key="numPairs">3</entry>
    <entry key="Gold1">Gold/Rocflo</entry>
    <entry key="Compare1">Rocflo/Rocout</entry>
    <entry key="metrics1">attributeMetricsRocflo.xml</entry>
    <entry key="Gold2">Gold/Rocfrac</entry>
    <entry key="Compare2">Rocfrac/Rocout</entry>
    <entry key="metrics2">attributeMetricsRocfrac.xml</entry>
    <entry key="Gold3">Gold/Rocburn</entry>
    <entry key="Compare3">RocburnAPN/Rocout</entry>
    <entry key="metrics3">attributeMetricsRocburn.xml</entry>
</properties>
```

B.3 attributeMetricsRocflo.xml

There are several of these files, defining parameters from the result HDF files for each physics code, and comparison criteria to be used for comparing them for each metric to be calculated by the Rocdiff utility.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : attributeMetrics.xml
    Created on : April 14, 2006, 2:30 PM
```

```
    Author     : Mark Brandyberry
    Description: Rocflo Attribute and Metric check definitions for
RocdiffAnalyze class
-->

<attributes>
    <comment>
            Each attribute element needs one or more metric elements.
            Each metric element has one or more criteria elements.
    </comment>
    <attribute key="Tf">
        <comment>Fluid Temperature in degrees K</comment>
        <metric key="MSE">
            <criteria key="lessThan">1.0E-20</criteria>
        </metric>
        <metric key="MDIFF">
            <criteria key="lessThan ">1.0E-12</criteria>
        </metric>
    </attribute>
    <attribute key="pf">
        <comment>Fluid Pressure in Pascals</comment>
        <metric key="MSE">
            <criteria key="lessThan ">1.0E-20</criteria>
        </metric>
        <metric key="MDIFF">
            <criteria key="lessThan ">1.0E-10</criteria>
        </metric>
    </attribute>
    <attribute key="mdot">
        <comment>Surface mass burning Rate in kg/m2s</comment>
        <metric key="MSE">
            <criteria key="lessThan ">1.0E-25</criteria>
        </metric>
        <metric key="MDIFF">
            <criteria key="lessThan ">1.0E-13</criteria>
        </metric>
    </attribute>
</attributes>
```