Los Alamos
NATIONAL LABORATORY
EST.1943

To/MS: Distribution
From/MS: Kelly Thompson/CCS–2, D409
Phone/FAX: (505)665–8090
Symbol: CCS–2:12-04(U)
Date: Jan 24, 2012

memorandum
CCS–2: Computational Physics & Methods
CCS-2, Mail Stop D413
Phone: 505-667-7029 Fax: 505-665-4972

Draco Release Policy and Procedures

Executive Summary

The purpose of this memo is to outline the release policy and procedures for Draco. This document is derived from the original memorandum written in 1999 by Tom Evans [1] and shares many features with the FreeBSD Release Engineering process [2]. These procedures should also apply to Draco-clients such as Capsaicin[1], and Jayenne (ClubIMC, Wedgehog, Milagro)[2]. We require releases in order to provide capability to users and for developers of software products that use Draco's capabilities. Formal releases allow CCS–2 code projects to maintain quality control, code reference, and project planning sanity. They are also used to provide usability, capability and performance enhancements.

This memo contains two primary sections. Section 3 covers release policy and § 4 reviews the mechanics used to produce a release. An Overview (§ 1) and Summary (§ 6) are also provided.

1. Overview

Draco evolved as a loose collection of individual components bound by a common build-system. While Draco is more tightly integrated in its present incarnation, Draco development still concentrates on individual components. Draco components are written in C++, utilize an Object Oriented Design (OOD), enforce Design-by-Contract verification, enforces component level encapsulation and levelized dependency between components. Draco is a software project that is continually evolving. Many components that were once actively developed and used have been retired. New components are frequently added to meet today's programming requirements.

Historically, the general release policy in Draco is to separate the package release number (e.g.: `c4-3_1_0`) from the Draco release number (e.g.: `draco-6_3_0`). Today, Draco component numbers are fully synchronized with the Draco release number. This change was made because Draco is relatively small and tagging of individual components wasn't usually necessary.

The next section (§ 2) of this document covers some nomenclature followed by a description of the Draco release policy. Adherence to these policies will expedite code reviews in the Draco project and will present a clean interface to outside users. Deviations from policies described herein are allowed under special circumstances as judged by the Draco project team. The Draco release procedure as described in § 4 utilizes the SVN version control system. Thus, Draco developers should peruse the SVN manual [3] to become intimate with its controls and use.

2. Nomenclature

At this point, we will illuminate some terminology that will be used throughout this memorandum. In the Draco source directory tree, any grouping of files that shares a common purpose is a Draco *component*. Examples of Draco *components* are `draco/src/c4/` and `draco/config/`. Draco-*clients* are code systems that use Draco components. Capsaicin and Milagro are Draco-*clients*. Draco *products* refer to anything produced from the Draco source-code tree (e.g.

[1] Email Jae Chang (jhchang@lanl.gov) for details

[2] Email Todd Urbatsch (tmonster@lanl.gov) for details

`librtt_c4.so`). On a final note, the term *packages* is often used in a supra-Draco sense. Milagro and Capsaicin are referred to as *packages*. These are fundamentally different than Draco packages. While we try to keep these distinctions firmly in place, one must often derive the meaning of package from context.

3. Draco Release Policies

The Draco release policy is a set of general requirements that controls the process of software release in the Draco code system. To start, we will write down the software release policies for Draco. Afterwards, we describe these policies in more detail. Note that these policies may be waived in special circumstances by the Draco development team.

The Draco release policies are:

1. Agreement to release: The Draco board must agree to the release content, schedule and determine if the release will be considered a major or minor release.
2. All components must pass their respective suite of regression tests on all target platforms. Inspection of the project dashboard is sufficient for this check.
3. Each Draco component must compile without error or warning using the build system standard compile flags. For this test, the code be configured, built and testing in at least two configurations:
   (a) Debug: `-DCMAKE_BUILD_TYPE=DEBUG -DGCC_ENABLE_ALL_WARNINGS=ON -DGCC_ENABLE_GLIBCXX_DEBUG=ON`
   (b) Release: `-DCMAKE_BUILD_TYPE=RELEASE`

   For each of these build types, all warnings should be eliminated before the release. Inspection of the project dashboard is sufficient is sufficient for this check.
4. The release sponsor must update the `ChangeLog` to reflect items to be included in the release. Then the Draco board will review the changes, assign and release number and set a target release date. A notification of the intended release will be emailed to the Draco mailing list (`draco@lanl.gov`). The `ChangeLog` entry for the release must include the following information:
   (a) The tag name (release revision name) must conform to requirements listed in § 3.1.
   (b) Associated client-software release names.
   (c) Target platforms and compilers for the release.
   (d) Executive summary of changes included with the release.
   (e) A detailed description of changes implemented for each component (including changes build system and the developer environment).
   (f) A list of known defects.
   (g) A list of defects resolved in this release.
   (h) Code metrics including lines-of-code and code coverage values.
5. Draco must pass a review that validates the release. The review team will include, at a minimum, all Draco developers that have contributed to any components since the last Draco release. Any objections concerning the release content or schedule from Draco team members must be communicated through the Draco mailing list. Objections might call out scheduling issues or major defects that must be resolved before the release can continue.
6. For major releases, a release memorandum must be generated by the sponsor that includes all of the `ChangeLog` information. The memorandum must include the following information:
   (a) All of the items from the `ChangeLog` as listed above.
   (b) A list of contributing authors.

(c) A brief history of prior releases (see § 4.8).

(d) A brief description of each component provided in the release.

(e) A discussion of design requirements: levelized component dependencies, thorough Design-by-Contract, unit testing, inheritance, etc.

(f) A figure showing the levelized component dependencies.

(g) A plot showing the lines-of-code evolved through time.

(h) Details about installation location and build options.

(i) A quick start guide for Draco developers and for developers who will use files from a Draco installation.

7. Each file in Draco will have at least a Draco-level tag conforming to the requirements listed in § 3.1. The Draco tag is a numbered release tag that refers to a fully-stable, released version of Draco. Additional Draco components such as `doc/`, `config/`, `environment/` and `autodoc/` may or may not have tags in addition to the common tags. This is a discretionary policy of the Draco team.
8. Each component must have its own unit tests that cover 70% of the component code base as measured by the nightly code coverage testing scripts (Bullseye Code Coverage [4] reported on the CCS–2 developer's dashboard [5,6].
9. Any priority 1 defects listed in the Draco bug tracker (TeamForge) must be closed.
10. No non-Draco tags should exist in Draco components that sit under the `draco/` directory in the SVN repository.
11. The file `Copyright.txt` must be updated to reflect the current Copyright dates.
12. Report the completed release to the Draco mailing list and add the release to TeamForge.
13. Third party software (e.g.: Trilinos, OpenMPI, etc.) must either be installed by the system administrators on the target machine or it must be installed by Draco developers a standard location such as `/usr/projects/draco/vendors` or `/ccs/codes/radtran/vendors`.
14. The version numbers found in the top level `CMakeLists.txt` must match the release version name. This number will automatically be inserted in the `Release()` functions.
15. While not formally reviewed Draco is deemed to be export controlled and should be protected as such.
16. Update documents (policy, process, user manuals, etc.) as needed to support the new version.

3.1. **Release Numbers**

The Draco project uses a uniform numbering scheme for all releases, Draco code releases are numbered according to the following scheme:

$$\langle name \rangle - \langle major \rangle\_\langle minor \rangle\_\langle branch \rangle \ .$$

In the above, $\langle name \rangle$ is the name Draco or the name of the Draco-client (e.g.: Capsaicin or Milagro), and $\langle major \rangle$ and $\langle minor \rangle$ are the major and minor release numbers, respectively. Bug fixes to a particular package release utilize SVN branches. A release of such a branch is noted by $\langle branch \rangle$. Release numbers are not decimal numbers. For example, `draco-1_32_0` is the thirty-second minor revision of release 1 of Draco. Most releases will have '0' as the *branch* version.

Release numbers are assigned at the discretion of the Draco board. Minor updates to a major release should be indicated by increasing the minor release number. Major changes or releases are indicated by the major release number. The Draco board decides whether a release is a major or minor revision. The only requirement is that the release numbers increase monotonically.

When updating to a new major release, the minor release number should be reset to zero. Thus, when moving from `draco-1_109_0` to release 2, the new release number should be `draco-2_0_0`. Additionally, we stated in § 3 that the

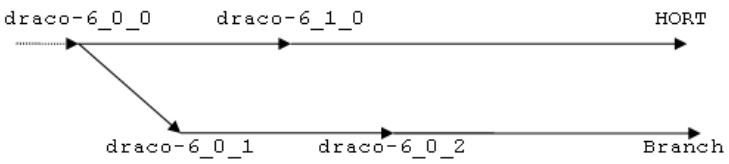

FIG. 1: Schematic of a bug-fix branch in SVN .

major release numbers in Draco package tags should correspond to the SVN major revision number.

On a final note, Draco-client systems such as Capsaicin and Milagro should be tagged in a similar manner. The only requirement for Draco-modeled projects is that they contain a system-level tag. Thus, Milagro is required, at a minimum, to have a `milagro-1_0_0` tag. Any additional tags are left to the discretion of the project development team. Remember, if the project uses Draco build-system components, it cannot tag any of them with a non-Draco tag.

3.2. **Code Branches and Bug Fix Releases**

As alluded to in earlier sections, bugs are handled on SVN branches that join at a particular release. For example, consider a bug fix to Draco release `draco-1_4_0` when the current release of Draco is `draco-1_6_0`. In this case, a branch is generated at the `draco-1_4_0` release as illustrated in Figure 1.

4. **Draco Release Procedures**

In § 3 we have defined a set of policies that dictate a code release process for Draco. In this section we shall proceed to define procedures for accomplishing those policies.

4.1. **Prerequisite for Release**

The following checklist provides the basic procedure that should be followed when making a formal release of the Draco libraries and binaries.

1. Determine what capabilities/bugs must be implemented/fixed for release. This is often a list generated at a team meeting or through email discussions. The TeamForge *version* key on tracker artifacts can also be used to help with this item. This information should be recorded in the top level `ChangeLog`. The format for your release notes should be similar to previous release notes in the `ChangeLog`.
2. Because several projects link against Draco, the release should be coordinated with all parties involved. At a minimum, the Capsaicin and Jayenne teams should be involved with planning the release.
3. Ensure appropriate unit testing for each package. All tests must report pass when CTest is executed from the Draco build directory for each build type (e.g: `Debug` with full DBC, `Release` without DBC, etc.). The code coverage reports found on CCS–2's CDash server may also be used to evaluate if additional tests are needed.
4. Ensure that the software will compile and all tests will pass for target platforms and build types. See the provided build script in § 4.5 and Table 1 for details about how to compile Draco and run the tests.
5. Propose a release date. This proposal should be send to the Draco mailing list.
6. Vendors: Note that you may also need to build and install vendor libraries at `/usr/projects/draco/vendors` (i.e.: atlas, gsl, etc.). See § 5 for more details.

With the above items complete, the release process can begin.

4.2. **Identify the Last Release**

The last release can be identified in a number of ways. Start by examining the `ChangeLog`, the version numbers found in `CMakeLists.txt`, release notes and email notification from prior releases. However, this process will rely primary on the SVN tag values for the top level `CMakeLists.txt`.

The tagged version of Draco can be found using the '`svn list`' command as follows:

```
svn list file:///ccs/codes/radtran/svn/draco/tags | grep draco-
        draco-6_3_0
        draco-6_2_1
        draco-6_2_0
        draco-6_1_0
        draco-6_0_0
        draco-5_21_0
...
```

The tag with the largest value should be the same as the most recent release entry in the `ChangeLog`. In the example above, the most recent release was `draco-6_3_0`. If the next release is a major release, its version tag will be `draco-7_0_0`. If the next release is a minor release, the version tag will be `draco-6_4_0` and if the next release is a bug fix or patch release it will be named `draco-6_3_1`.

4.3. **Generate ChangeLog**

The `ChangeLog` is a text file in the source tree that provides a summary of the release. See § 3 for a list of items that must be included in the `ChangeLog` for each release. Often it is desirable to cut and paste previous release notes from the `ChangeLog`. The *release coordinator* should generate the release entry in the `ChangeLog`.

The first line of release entry should be the proposed release version and specify in text if this is a *major*, *minor* or *patch* release. The type of release must be decided by the Draco board based on the number and significance of proposed changes.

The second entry should list the supported platforms, compilers and vendors for the release. This is usually specified by the customer.

4.3.1. *Target Platforms.* Draco is used by the Eulerian Applications Project, EAP, and the target platforms for each release will mirror EAP's targets. The best way to determine the target platforms and the compiler and vendor versions for each platform is to examine EAP's configuration file `/usr/projects/crestone/dotfiles/Cshrc`. Currently, the machines and compiler sets listed in Table 1 are supported.

TABLE 1: Supported Platforms and Compilers for Draco Releases

Machine	Compiler Set	Vendors
Hurricane/Turing/Typhoon/Luna/Moonlight	Intel 12.1.2	OpenMPI-1.4.5
Ceilo/Ceilito	Intel 12.1.2	MPICH2-5.4.2
Roadrunner/rr_dev	GCC 4.3.0	OpenMPI-1.4.3, DaCS
YellowRail	PGI 9.0-3	OpenMPI-1.4.3
Dawn	–	–
Mapache	–	–

Development is primary done on CCS–2 LAN machines such as `ccscs8`, `ccscs9` and on personal RedHat Linux and Macintosh OS/X workstations. These platforms are informally supported with various compiler (GNU, Portland Group, Intel) and vendor sets.

4.3.2. *Identify and record changes since last release.* A summary changes per component will be included in the `ChangeLog`, including changes to the build system, Draco developer environment and documents. Here is the process that can provide this information.

1. Generate a list of files that have been modified since the last release and count the files.

```
svn diff -r"{2012-01-06}":"{2012-06-28}" --summarize &> modified_files.log
num_files_changed='cat modified_files.log | awk '{print $2}' | sort -u | wc -l'
echo $num_files_changed
```

2. Generate a log file that contains all commit messages since the last release. This command doesn't work with symbolic tag names and you must use a date range instead. The date of the last release can be taken from the `ChangeLog`.

```
svn log -r"{2012-01-06}":"{2012-06-28}" --verbose &> svn_messages.log
number_of_commits='cat svn_message.log | grep "Changed paths:" | wc -l'
```

3. Go through the file `svn_messages.log` and summarize the log entries in the `ChangeLog` following the format from previous release messages.

Record the number of files modified and the number of commits to the `ChangeLog`. Summarize the SVN commit messages per component. If a SVN commit message is confusing or incomplete the release coordinator should contact the author to clarify the change.

4.3.3. *Generate a list of known and corrected defects.* Draco uses the LANL TeamForge server for tracking defects. A list of known defects can be copied directly from the `Tracker::Bugs` summary page. A snapshot of this defect list is shown in Figure 2. At this time it is useful to go through the bugs listed in the tracker and provide updates as needed. This is a good opportunity to remind developers of defects they are assigned to. Note that all *Priority 1* defects must be resolved before the release can continue.

Resolved defects are also extracted from the Draco page on TeamForge. You will need to navigate to the `Reports` tab and then select and edit the report for `Recently closed bugs`. Update the date range for *Closed* to match the date of the previous release up to the current date and then generate the report by selecting the *Report Details* tab as shown in Figure 3. The list of resolved defects can be copied to `ChangeLog`.

4.3.4. *Code Metrics.* The last set of data required for the release note in the `ChangeLog` are the current code statistics including total lines-of-code, LOC, as reported by the tool `cloc`[3] and the function point code coverage as reported by the Bullseye tool [4]. Both of these reports can be extracted from the nightly dashboard data by using the Draco script `regression/metrics_report.sh`. Edit this script and set the email address to your own, then run the script. An email with all of the relevant data will be created. The data will look something like what follows and should be copied into the `ChangeLog` file.

```
========================================================================
Code Metrics for Draco and Jayenne, Thu Jul 5 12:21:43 MDT 2012
========================================================================

______________________
Draco
```

[3] `http://cloc.sourceforge.net/`

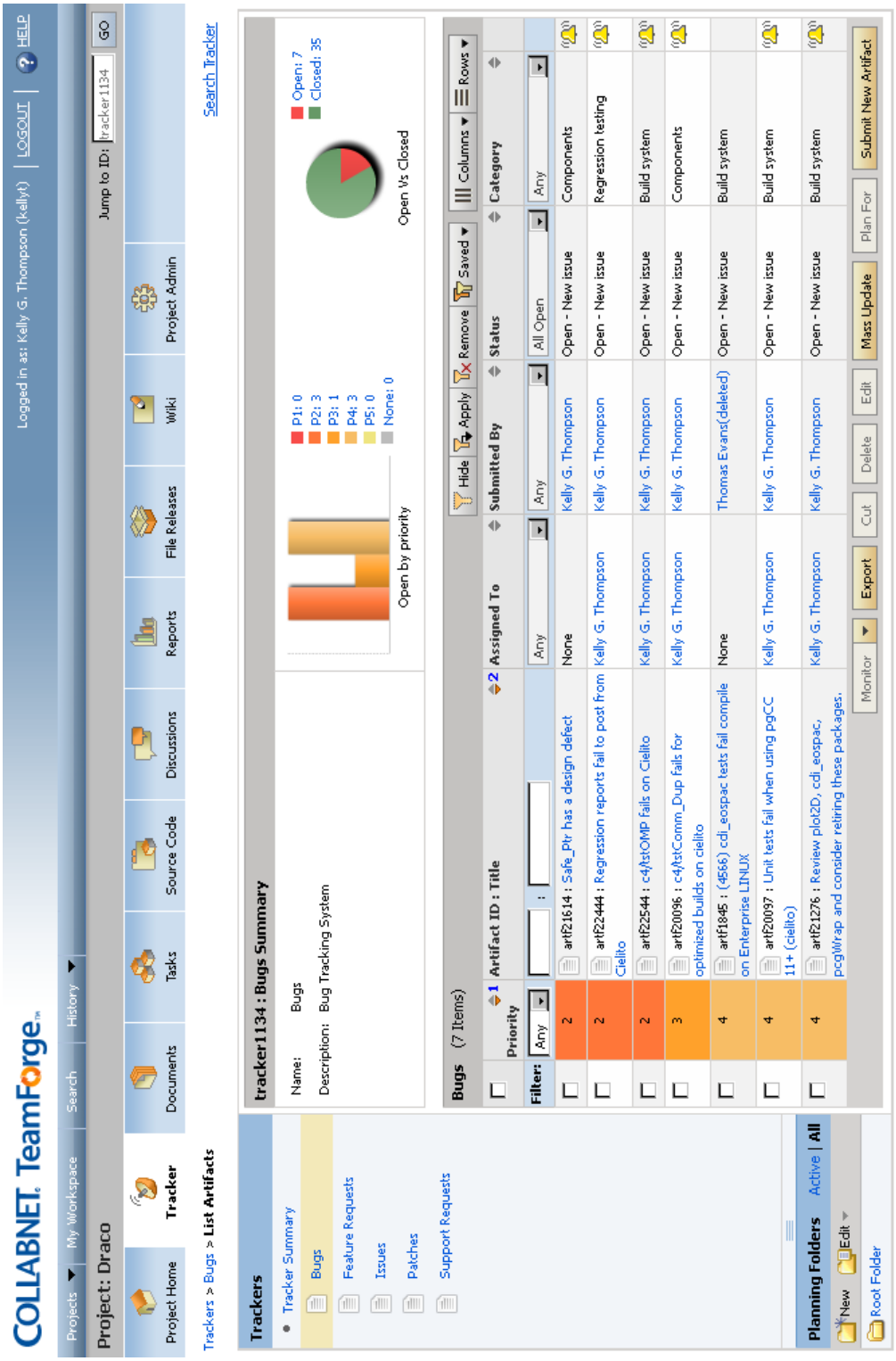

FIG. 2: A list of active defects for Draco is maintained on LANL's TeamForge server. This list can be copied directly into the `ChangeLog`.

FIG. 3: A list of resolved defects for Draco is reported by LANL's TeamForge server. This list can be copied directly into the `ChangeLog`.

Lines of code

```
http://cloc.sourceforge.net v 1.54  T=21.0 s (54.6 files/s, 12985.3 lines/s)
```

Language	files	blank	comment	code
C++	423	15036	22227	72905
Text	178	6486	6816	47634
C/C++ Header	355	8334	22978	20914
Lisp	13	1801	2517	11916
Perl	2	1294	1009	8330
CMake	104	1729	3693	4797
XML	2	933	0	2418
Python	26	1210	1343	2197
C	14	277	552	652
Bourne Shell	10	170	488	548
CSS	1	113	37	395
make	11	171	108	344
Fortran 90	2	19	34	54
C Shell	2	18	40	51
HTML	2	8	15	45
awk	1	6	2	27
SUM:	1146	37605	61859	173227

Code coverage

Directory	Function Coverage	C/D Coverage
units/	82 / 82 = 100%	58 / 58 = 100%
traits/	16 / 16 = 100%	0 / 0
diagnostics/	8 / 8 = 100%	4 / 4 = 100%
special_functions/	29 / 29 = 100%	190 / 198 = 95%
fit/	1 / 1 = 100%	15 / 16 = 93%
ode/	5 / 5 = 100%	29 / 32 = 90%
cdi/	39 / 39 = 100%	27 / 30 = 90%
min/	9 / 9 = 100%	137 / 154 = 88%
linear/	16 / 16 = 100%	324 / 392 = 82%
roots/	8 / 8 = 100%	256 / 312 = 82%
mesh_element/	24 / 24 = 100%	165 / 213 = 77%
cdi_ipcress/	106 / 106 = 100%	191 / 252 = 75%
meshReaders/	15 / 15 = 100%	127 / 172 = 73%
fpe_trap/	2 / 2 = 100%	4 / 11 = 36%
ds++/	667 / 670 = 99%	586 / 665 = 88%
RTT_Format_Reader/	319 / 321 = 99%	273 / 356 = 76%
parser/	266 / 269 = 98%	1075 / 1357 = 79%
timestep/	62 / 63 = 98%	104 / 178 = 58%
rng/	68 / 70 = 97%	65 / 70 = 92%
norms/	25 / 26 = 96%	21 / 34 = 61%
viz/	24 / 25 = 96%	129 / 168 = 76%
cdi_analytic/	133 / 140 = 95%	163 / 194 = 84%
quadrature/	179 / 191 = 93%	544 / 637 = 85%
c4/	177 / 190 = 93%	356 / 414 = 85%
plot2D/	22 / 29 = 75%	46 / 76 = 60%
lapack_wrap/	12 / 16 = 75%	0 / 0
Total	2314 / 2370 = 97%	4889 / 5993 = 81%

4.4. **Update the version tags**

Now that the `ChangeLog` is complete, it is time to update the version information in the code and tag the SVN repository. The version number as reported by ds++'s `Release()` function is recorded in the code at `CMakeLists.txt`. Update the following two lines and commit the file to SVN.

```
#
# The Draco version number.
#
set(DRACO_VERSION_MAJOR 6)
set(DRACO_VERSION_MINOR 3)
```

The value of `DRACO_VERSION_PATCH` is set manually during a release build. For development builds the third portion of the version tag will be set to the configure date (e.g.: `draco-6_3_20120113` for code configured on January 13, 2012). Note that the version tag is stored in the file `ds++/config.h` and can be extracted with the command

```
grep DRACO_VERSION_FULL ds++/config.h
```

It may be necessary to update the version name in `README.draco` and in other documents found in the source repository. Use your discretion to determine what documents need to be updated. For the `README.draco` file, the author list may needed to be updated.

Before tagging the SVN repository, ensure that all changes in your local copy are checked in. This is often done with a command similar to

```
svn commit -m"Preparing for the release of draco-6_4_0"
svn update
```

Assuming that the working directory is fully sychronized with the HEAD revision of the SVN repository, we can tag the release.

```
svn copy file:///ccs/codes/radtran/svn/draco/trunk \
   file:///ccs/codes/radtran/svn/draco/tags/draco-6_4_0 \
   -m "Tagging the draco-6_4_0 release."
```

4.4.1. *Bug Fixes.* As mentioned in the policy section, bug fixes are handled on SVN branches. To generate such a branch the following SVN commands are used:

```
svn copy file:///ccs/codes/radtran/svn/draco/tags/draco-1_4_0 \
   file:///ccs/codes/radtran/svn/draco/tags/draco-1_4_1 \
   -m "Creating the a branch for a patch to Draco-1_4_0."
```

This creates a branch to Draco release 1.4. If the bug fix is required in a Draco release, Draco should be re-released with the appropriate branch number. Branches are used for bug fixes only.

Additional patch branches to fix other bugs for Draco-2_4_0 should be generated by using the `svn copy` command to create a new branch at `/ccs/codes/radtran/svn/draco/tags`. Thus, to make a tag to this version use the following commands:

```
svn copy file:///ccs/codes/radtran/svn/draco/tags/draco-1_4_1 \
   file:///ccs/codes/radtran/svn/draco/tags/draco-1_4_2 \
   -m "Creating the a branch for a patch to Draco-1_4_1."
(make changes)
```

```
(run regression tests)
```

We note that SVN branches automatically release code to the end of the branch. Hence, '`svn checkout file:///ccs/codes/radtran/svn/draco/tags/draco-1_4_1 draco`' will automatically give the sources at the end of the Draco branch connected to release 1.4. We add tags to clarify the development and bug tracking process in Draco even though they may not be strictly necessary in all cases. For more information on branches, see Chapter 4 of the SVN manual [3].

4.5. **Generating the release**

Releases are only generated on HPC hardware (Turing, Cielito, etc.). Because of software environment changes, this set of instruction is only a guide and may need to be modified slightly based on each situation. The steps for generating a release are: creating the release file structure, setting the software environment, obtained the source code, building, testing and installing the product, and checking for errors or warnings.

1. Create the install location by following the example of previous releases. Create a release directory structure based on the version name.

```
mkdir -p /usr/projects/draco/draco-6_3_0/source
mkdir -p /usr/projects/draco/draco-6_3_0/logs
cp -r /usr/projects/draco/draco-6_2_0/scripts /usr/projects/draco/draco-6_3_0/scripts
```

2. Access the tagged source code.

```
cd /usr/projects/draco/draco-6_3_0/source
svn checkout svn+ssh://ccscs8/ccs/codes/radtran/svn/draco/tags/draco-6_3_0 draco
```

3. Edit the build/test/release scripts found at `/usr/projects/draco/draco-6_3_0/scripts`.
4. Review the customer's desired compiler set for this platform. This can be accomplished by reviewing or sourcing `/usr/projects/crestone/dotfiles/Cshrc`. These changes should be added to the build/test/release scripts.
5. Update the developer environment (e.g.: Modules [7], vendor setups, environment variables) with Draco specific settings. These changes should be added to the build/test/release scripts.
6. Run the build/test/release script. This script ususally builds several versions (debug, opt, opt non-reproducible, etc) of the code for one platform. The script may be either a shell script or a `msub` script for submission under Torque.
7. Once the script completes, check the log file to ensure that no errors occured and that all tests pass.
8. A separate script must be run and the log file examined for each supported platform and compiler set as listed in Table 1

An sample release script is shown below. This script should be edited carefully before it is run to ensure that version numbers, locations, build types and build environment are correctly set

```
#!/bin/bash -l

#MSUB -l walltime=01:00:00
#MSUB -l nodes=1:ppn=16
#MSUB -j oe
#MSUB -o /usr/projects/draco/draco-6_3_0/logs/release_tu.log

#------------------------------------------------------------------------------#
```

```
# The script starts here
#-------------------------------------------------------------------------#

# Permissions - new files should be marked u+rwx,g+rwx,o+rx
umask 0002
build_permissions="g+rwX"
install_permissions="g+rwX,o=g-w"

# environment (use draco modules)
module purge
module load friendly-testing python cmake
module load intel/12.1.2 openmpi/1.4.5
module load gsl/1.14-intel lapack/3.4.0-intel numdiff
module list

# This is the bug fix number.  Normally set this to zero.
CONFIG_BASE="-DDRACO_VERSION_PATCH=0"

# Define your source and build information here.

ddir="draco-6_3_0"
platform="tu"
dmpi=openmpi145
dcpp=intel1212
df90=$dcpp

# Locations for source, build and install
source_prefix="/usr/projects/draco/$ddir"
build_prefix="/scratch/$USER/$ddir/$platform-${dmpi}-${df90}-${dcpp}"
install_prefix="$source_prefix/$platform-${dmpi}-${df90}-${dcpp}"

# Helpful functions:
die () { echo "ERROR: $1"; exit 1;}

run () {
    if [ $dry_run ]
    then echo $1
    else eval $1
    fi
}

# Define the meanings of various configure features:
DBC_OFF="-DDRACO_DBC_LEVEL=0"
DBC_ON="-DDRACO_DBC_LEVEL=7"

OPTIMIZE_ON="-DCMAKE_BUILD_TYPE=Release"
OPTIMIZE_OFF="-DCMAKE_BUILD_TYPE=Debug"

LOGGING_ON="-DDRACO_DIAGNOSTICS=7 -DDRACO_TIMING=1"
LOGGING_OFF="-DDRACO_DIAGNOSTICS=0 -DDRACO_TIMING=0"

NR_ON="-DENABLE_RNG_NR=ON"
NR_OFF="-DENABLE_RNG_NR=OFF"

# Define the meanings of the various code versions:
VERSIONS=(\
    "debug"    "debug_nodbc"    "opt"    "opt_log" \
    "debug_nr" "debug_nodbc_nr" "opt_nr" "opt_log_nr" \
)
OPTIONS=(\
    "$DBC_ON  $OPTIMIZE_OFF $LOGGING_ON  $NR_OFF" \
    "$DBC_OFF $OPTIMIZE_OFF $LOGGING_ON  $NR_OFF" \
    "$DBC_OFF $OPTIMIZE_ON  $LOGGING_OFF $NR_OFF" \
    "$DBC_OFF $OPTIMIZE_ON  $LOGGING_ON  $NR_OFF" \
```

```
\
    "$DBC_ON  $OPTIMIZE_OFF $LOGGING_ON  $NR_ON"  \
    "$DBC_OFF $OPTIMIZE_OFF $LOGGING_ON  $NR_ON"  \
    "$DBC_OFF $OPTIMIZE_ON  $LOGGING_OFF $NR_ON"  \
    "$DBC_OFF $OPTIMIZE_ON  $LOGGING_ON  $NR_ON"  \
)
PACKAGES=("draco")

# =============
# Configure, Build and Run the Tests
# =============


# Loop over the code versions:

for (( i=0 ; i < ${#VERSIONS[@]} ; ++i ))
do

    version=${VERSIONS[$i]}
    options=${OPTIONS[$i]}

    echo
    echo
    echo "# Code Version: $version"
    echo "# ---------------"
    echo

    # Create install directory
    install_dir="$install_prefix/$version"
    run "mkdir -p $install_dir" || die "Could not create $install_dir"

    # Loop over the packages.
    for package in ${PACKAGES[@]}
    do
        echo
        echo "#    Package: $package"
        echo "#    --------"
        echo

        source_dir="$source_prefix/source/$package"
        build_dir="$build_prefix/$version/${package:0:1}"

        run "mkdir -p $build_dir" || die "Could not create directory $build_dir."
        run "cd $build_dir"
        run "cmake -DCMAKE_INSTALL_PREFIX=$install_dir \
            $options $CONFIG_BASE $source_dir" \
            || die "Could not configure in $build_dir from source at $source_dir"
        run "make -j20 all"  || die "Could not build code/tests in $build_dir"
        run "ctest -j16" # Allow some tests to fail.
        run "make -j20 install"  || die "Could not build code/tests in $build_dir"
        run "chmod -R $build_permissions $build_dir"

    done

    # Set access to install dir.
    run "chmod -R $install_permissions $install_dir"

done

# Set access to top level install dir.
run "chmod $install_permissions $install_prefix"
run "chgrp -R draco $install_prefix"
```

4.6. **Release notification**

Once the release is complete, a notification via email to `draco@lanl.gov` must be issued. You may need to contact other interested parties as well. Sample email notices can be found in the source repository at `/doc/releases`. The notification should also be made on TeamForge by adding the new version to the *File Releases* section and by posting a *News Post*.

4.7. **Backup the release**

The release must also be backed up to the tri-lab archive. From a CCS–2 LAN machine, tar the SVN repository and copy the tar file to `/usr/projects/asc_bkups` on HPC. This can only be done at certain times of the year when the `asc_bkups` is open for backup submissions.

```
# CCS–2 LAN
svnadmin dump /ccs/codes/radtran/svn/draco | bzip2 > draco.svn.bz2
push draco.svn.bz2
# HPC
cd /usr/projects/asc_bkups/draco
pull 1 draco.svn.bz2
```

4.8. **Generating a contributing author list**

If a contributing author list is required (i.e.: for use in a release memo or other document), the following procedure may be used:

1. Generate a list of files found in this release:

```
files=`find . -name '*.hh' -o -name '*.cc' -o -name '*.txt' \
       -o -name '*.cmake' -o -name '*.in' -o -name '*.h'`
svn annotate $files > file_list
```

2. Generate a list of authors.

```
user_list=`cat file_list | awk '{print $2}' | sort -u`
```

3. Generate a report

```
for name in $user_list; do numlines=`grep $name file_list | wc -l`; \
     echo "$numlines: $name"; done > author_loc
cat author_loc | sort -rn

68326: kellyt
45126: kgbudge
23521: tme
7378: lowrie
6153: bta
5541: mwbuksas
4360: mcghee
2807: pautz
2489: phenning
2368: warsa
2186: rsqrd
2005: gaber
1201: furnish
...
```

Monikers can be converted to full names by using the `finger` command.

5. Vendors

Draco depends on a few third party vendors to provide full functionality. These vendors are typically loaded into the developer environment through the use of `module` commands [7]. Vendor libraries are loaded from the local system if available. Otherwise, the vendor libraries can be loaded from either `/usr/projects/draco/vendors` or `/ccs/codes/radtran/vendors/Linux64` through the use of Draco '`module load`' commands.

Currently, Draco can make use of the following vendor libraries

TABLE 2: Vendor libraries used by Draco

Vendor name	Recommended Version	Required or Optional
OpenMPI	1.4.5+	Required
GNU Scientific Library	1.14+	Required
Numdiff	5.2.1+	Required
Python	<3.0	Required
CMake	2.8.8+	Required
LAPACK	3.4.0+	Optional
Grace (gnuplot)	4.4+	Optional
PAPI	4.1+	Optional
CUDA	4.1+	Optional

6. Summary

We have detailed the release policies and procedures for Draco projects in group CCS–2. As with all Draco policies, exceptions to the standard rules are acceptable pending review by Draco team members. These procedures have been reviewed and are part of the Draco development process. While the basic procedures explained in this document are robust, all procedures could benefit from future enhancements and this document should be updated as needed.

References

[1] T. EVANS, "Draco release policy and procedures," Tech. Memo XTM:99–36(U), Los Alamos National Lab., 1999.

[2] M. STOKELY, "FreeBSD Release Engineering." http://www.freebsd.org/doc/en/articles/releng, 2012.

[3] C. M. P. BEN COLLINS-SUSSMAN, BRIAN W. FITZPATRICK, "Version Control with Subversion." http://svnbook.red-bean.com/en/1.7/index.html, 2012.

[4] "BullseyeCoverage." http://www.bullseye.com, 2011.

[5] "Cdash dashboard for ccs-2." http://coder.lanl.gov/cdash, 2011.

[6] "CMake." http://www.cmake.org, 2011.

[7] "Environment modules project." http://modules.sourceforge.net, 2011.

KGT:kgt

Distribution:
Baker Randal S. CCS-2 D409
Budge Kent G. CCS-2 D409
Chang Jae Ho CCS-2 D409
Densmore Jeffery D. CCS-2 D409
Lowrie Robert B. CCS-2 D413
Palmer Todd S. CCS-2 D409
Rockefeller Gabriel M. CCS-2 D409
Rosa Massimiliano CCS-2 K784
Turner Scott A. CCS-2 D409
Urbatsch Todd J. CCS-2 D409
Warsa James S. CCS-2 D409
Wollaber Allan B. CCS-2 D409

CCS–2 Files D409