The Hacker's Guide to BRL-CAD

Please read this document if you are contributing to BRL-CAD.

BRL-CAD is a relatively large package with a long history and heritage. Many people have contributed over the years, and there continues to be room for involvement and improvement in just about every aspect of the package. As such, contributions to BRL-CAD are very welcome and appreciated.

For the sake of code consistency, project coherency, and the long-term evolution of BRL-CAD, there are guidelines for getting involved. Contributors are strongly encouraged to follow these guidelines and to likewise ensure that other contributors similarly follow the guidelines. There are simply too many religious wars over what usually come down to personal preferences and familiarity that are distractions from making productive progress.

These guidelines apply to all developers, documentation writers, testers, porters, graphic designers, web designers, and anyone else that is directly contributing to BRL-CAD. As all contributors are also directly assisting in the development of BRL-CAD as a package, this guide often simply refers to contributors as developers.

Although BRL-CAD was originally developed and supported primarily by the U.S. Army Research Laboratory and its affiliates, it is now a true Open Source project with contributions coming in to the project from around the world. The U.S. Army Research Laboratory continues to support and contribute to BRL-CAD, but now the project is primarily driven by a team of core developers and the BRL-CAD open source community. Contact the BRL-CAD developers for more information.

TABLE OF CONTENTS

GETTING STARTED

As there are many ways to get started with BRL-CAD, one of the most important steps for new contributors to do is get involved in the discussions and communicate with the BRL-CAD developers. There are mailing lists, on-line forums, and an IRC channel dedicated to BRL-CAD development and project communication. All contributors are encouraged to participate in any of the available communication channels:

* Internet Relay Chat

  The primary and generally preferred mechanism for interactive developer discussions is via Internet Relay Chat (IRC). Several of the core developers and core contributors of BRL-CAD hang out in #brlcad on the Freenode network. With most any IRC client, you should be able to join #brlcad on irc.freenode.net, port 6667. See http://freenode.net and http://irchelp.org for more information

* E-mail Mailing Lists

  There are several mailing lists available for interaction, e.g. the http://sourceforge.net/mail/?group_id=105292 "brlcad-devel" mailing list. More involved contributors may also be interested in joining the "brlcad-commits" and "brlcad-tracker" mailing lists.

* On-line Forums

  Discussion forums are available on the project site at http://sourceforge.net/forum/?group_id=105292 for both developers and users. Of particular interest to developers is, of course, the "Developers" forum where all contributors are encouraged to participate.

HOW TO CONTRIBUTE

BRL-CAD's open source management structure is best described as a meritocracy. Roughly stated, this basically means that the power to make decisions lies directly with the individuals that have ability or merit with respect to BRL-CAD. An individual's ability and merit is basically a function of their past and present contributions to the project. Those who constructively contribute, frequently interact, and remain highly involved have more say than those who do not.

As BRL-CAD is comprised of a rather large code base with extensive existing documentation and web resources, there are many many places where one may begin to get involved with the project. More than likely, there is some new goal you already have in mind, be it a new

geometry converter, support for a different image type, a fix to some bug, an update to existing documentation, a new web page, or something else entirely. Regardless of the goal or contribution, it is highly encouraged that you interact with the existing developers and discuss your intentions. This is particularly important if you would like to see your modification added to BRL-CAD and you do not yet have contributor access. When in doubt, working on resolving existing bugs, improving performance, documentation, and writing tests are perfect places to begin.

For many, it can be overwhelming at just how much there is. To a certain extent, you will need to familiarize yourself with the basic existing infrastructure before significantly changing or adding a completely new feature. There is documentation available in the source distribution's doc/ directory, throughout the source hierarchy in manpages, on the website, and potentially in the documentation tracker at http://sourceforge.net/docman/?group_id=105292 covering a wide variety of topics. Consult the existing documentation, sources, and developers to become more familiar with what already exists.

See the PATCH SUBMISSION GUIDELINES section below for details on preparing and providing contributions.

REFACTORING

proportion -> integrity -> clarity

Refactoring is one of the most useful activities a contributor can make to BRL-CAD. Code refactoring involves reviewing and rewriting source code to be more maintainable through reduced complexity and improved readability, structure, and extensibility.

For each source file in BRL-CAD, the following checklist applies:

* Consistent indentation. See CODING STYLE & STANDARDS below. Indents every 4 characters, tab stops at 8 characters with BSD KNF indentation style. The sh/indent.sh script will format a file automatically, but requires a manual review afterwards.

* Consistent whitespace. See CODING STYLE & STANDARDS below, section on stylistic whitespace.

* Headers. Only including headers that declare functions used by that file. If system headers are required, then common.h should be the first header included.

* Comments. All public functions are documented with doxygen comments. Move public comments to the public header that declares the function. Format block comments to column 70 with only one space (not tabs) after the asterisk. Comments should explain why more than what.

* Magic numbers. Eliminate constant numbers embedded in the code wherever feasible, instead preferring dynamic/unbounded allocation.

* Public symbols. Public API symbols should be prefixed with the library that they belong to and declared in a public header. Public symbols should consistently (only) use underscores, not CamelCase.

* Private symbols. Private functions should be declared HIDDEN.

* Dead code. Code that is commented out should be removed unless it serves a specific documentation purpose.

* Duplicate code. Combine common functionality into a private function or new public API routine. Once and only once.

* Verbose compilation warnings. Quell them all.

* Globals. Eliminate globals by pulling them into an appropriate scope and passing as parameters or embedding them in structures as data.

In addition, don't be afraid to rewrite code or throw away code that "smells bad". No code is sacred. Perfection is achieved not when there is nothing more to add but, rather, when there is nothing more to take away.

SYSTEM ARCHITECTURE

At a glance, BRL-CAD consists of about a dozen libraries and over 400 executable binaries. The package has been designed from the ground up adopting a UNIX methodology, providing many tools that may often be used in harmony in order to complete a task at hand. These tools include geometry and image converters, image and signal processing tools, various raytrace applications, geometry manipulators, and more.

One of the firm design intents of the architecture is to be as cross-platform and portable as is realistically and reasonably possible. As such, BRL-CAD maintains support for many legacy systems and devices provided that maintaining such support is not a significant burden on developer resources. Whether it is a burden or not is of course a potentially subjective matter. As a general guideline, there needs to be a strong compelling motivation to actually remove any functionality. Code posterity, readability, and complexity are generally not sufficient reasons. This applies to sections of code that are no longer being used, might not compile, or might even have major issues (bugs). This applies to bundled 3rd party libraries, compilation environments, compiler support, and language constructs.

In correlation with a long-standing heritage of support is a design intent to maintain verifiable repeatable results throughout the package, in particular in the raytrace library. BRL-CAD includes

scripts that will compare a given compilation against the performance of one of the very first systems to support BRL-CAD: a VAX 11/780 running BSD. As the BRL-CAD Benchmark is a metric of the raytrace application itself, the performance results are a very useful metric for weighing the relative computational strength of a given platform. The mathematically intensive computations exercise the processing unit, system memory, various levels of data and instruction cache, the operating system, and compiler optimization capabilities.

To support what has evolved to be a relatively large software package, there are a variety of support libraries and interfaces that have aided to encapsulate and simplify application programming. At the heart of BRL-CAD is a Constructive Solid Geometry (CSG) raytrace library. BRL-CAD has its own database format for storing geometry to disk which includes both binary and text file format representations. The raytrace library utilizes a suite of other libraries that provide other basic application functionality.

LIBRARIES

librt: The BRL-CAD Ray-Trace library is a performance and accuracy-oriented ray intersection, geometric analysis, and geometry representation library that supports a wide variety of geometric forms. Geometry can be grouped into combinations and regions using CSG boolean operations.
Depends on: libbn libbu libregex libm (openNURBS)

libbu: The BRL-CAD Utility library contains a wide variety of routines for memory allocation, threading, string handling, argument support, linked lists, and more.
Depends on: (threading) (malloc)

libbg: The BRL-CAD Geometry library implements generic algorithms widely used in computational geometry, such as convex hull and triangle/triangle intersection calculations.
Depends on: libbn libbu

libbn: The BRL-CAD Numerics library provides many floating-point math manipulation routines for vector and matrix math, a polynomial equation solver, noise functions, random number generators, complex number support, and more as well.
Depends on: libbu libm

libbrep: The BRL-CAD Boundary Representation library implements routines needed for the manipulation and analysis of Non-Uniform Rational BSpline based boundary representations.
Depends on: libbg libbn libbu OpenNURBS

libcursor: The cursor library is a lightweight cursor manipulation library similar to curses but with less overhead.
Depends on: termlib

libdm: The display manager library contains the logic for generalizing

a drawable context. This includes the ability to output drawing/plotting instructions to a variety of devices such as X11, Postscript, OpenGL, plot files, text files, and more. There is generally structured order to data going to a display manager (like the wireframe in the geometry editor).
Depends on: librt libbn libbu libtcl libpng (X11)

libfb: The framebuffer library is an interface for managing a graphics context that consists of pixel data. This library supports multiple devices directly, providing a generic device-independent method of using a frame buffer or files containing frame buffer images.
Depends on: libbu libpkg libtcl (X11) (OpenGL)

libfft: The Fast-Fourier Transform library is a signal processing library for performing FFTs or inverse FFTs efficiently.
Depends on: libm

liboptical: The optical library is the basis for BRL-CAD's shaders. This includes shaders such as the Phong and Cook-Torrence lighting models, as well as various visual effects such as texturing, marble, wood graining, air, gravel, grass, clouds, fire, and more.
Depends on: librt libbn libbu libtcl

libpkg: The "Package" library is a network communications library that supports multiplexing and demultiplexing synchronous and asynchronous messages across stream connections. The library supports a client-server communication model.
Depends on: libbu

libtclcad: The Tcl-CAD library is a thin interface that assists in the binding of an image to a Tk graphics context.
Depends on: libbn libbu libfb libtcl libtk

libtermio: The terminal I/O library is a TTY control library for managing a terminal interface.
Depends on nothing

libwdb: The "write database" library provides a simple interface for the generation of BRL-CAD geometry files supporting a majority of the various primitives available as well as combination/region support. The library is a write-only interface (librt is necessary to read & write) and is useful for procedural geometry.
Depends on: librt libbn libbu

libbrlcad: This "conglomerate" library provides the core geometry engine facilities in BRL-CAD by combining the numerics, ray-tracing, and geometry database processing libraries into one library.
Depends on: libwdb librt libbn libbu
Includes dependent libraries as part of this library

FILESYSTEM ORGANIZATION

BRL-CAD has a STABLE branch in SVN that should always compile and run with expected behavior on all supported platforms. Contrary to STABLE, the SVN trunk is generally expected to compile but more flexibility is allowed for resolution of cross-platform build issues and on-going development.

Included below is a sample (not comprehensive) of how some of the sources are organized in the source distribution. For the directories under src/ and doc/docbook, see the provided README file for more details on subdirectories and authorship not covered here.

bench/
    The BRL-CAD Benchmark Suite
db/
    Example geometry databases
doc/
    Documentation
doc/docbook
    Documentation in DocBook xml format, see doc/docbook/README
    for more details
include/
    Public headers
misc/
    Anything not categorized or is sufficiently en masse
pix/
    Sample raytrace images, includes benchmark reference images
regress/
    Scripts and resources for regression testing
sh/
    Utility scripts, used primarily by the build system
src/
    Sources, see src/README for more details
src/adrt
    Advanced Distributed Ray Tracer
src/conv/
    Various geometry converters
src/conv/iges/
    IGES converter
src/fb/
    Tools for displaying data to or reading from framebuffers
src/fbserv/
    Framebuffer server
src/java/
    Java geometry server interface to librt
src/libbn/
    BRL-CAD numerics library
src/libbu
    BRL-CAD utility library
src/libfb/
    BRL-CAD Framebuffer library
src/libfft/
    Fast Fourier transform library
src/libged/
    Geometry editing library

```
src/libpkg/
    Network "package" library
src/librt/
    BRL-CAD Ray-trace library
src/libwdb/
    Write database library
src/mged/
    Multi-device geometry editor
src/other/
    External frameworks (Tcl/Tk, libpng, zlib, etc.)
src/proc-db/
    Procedural geometry tools, create models programmatically
src/remrt/
    Distributed raytrace support
src/rt/
    Raytracers, various
src/util/
    Various image processing utilities
```

SOURCE CODE LANGUAGES

The vast majority of BRL-CAD is written in ANSI C with the intent to be strictly conformant to the C standard. A majority of the MGED geometry editor is written in a combination of C, Tcl/Tk, and Incr Tcl/Tk. The BRL-CAD Benchmark, build system, and utility scripts are written in what should be POSIX-compliant Bourne Shell Script. An initial implementation of a BRL-CAD Geometry Server is written in PHP.

With release 7.0, BRL-CAD has moved forward and worked toward making all of BRL-CAD C code conform strictly with the ANSI/ISO standard for C language compilation (ISO/IEC 9899:1990, i.e. c89). Support for older compilers and older K&R-based system facilities is being migrated to build system declarations, preprocessor defines, or being removed outright. It's okay to make modifications that assume compiler conformance with the ANSI C standard (c89).

There is currently no C++ interface to the core BRL-CAD libraries. There are a few tools and enhancements to libraries that are implemented in C++ (the new BREP object type in librt, for example), and C++ is used underneath quite a number of libraries, but there is presently no public C++ API in use outside of OpenNURBS.

While C++ as an implementation language of new tools and new library interfaces is not prohibited, the mixing of C++ semantics and C++ code (including simple // style comments) in the existing C files is not allowed. As new interfaces are developed, new contributors become involved, and C++ code integration becomes more prevalent, the contributor guidelines will become reinforced with more details.

CODING STYLE & STANDARDS

For anyone who plans on contributing code, the following conventions should be followed. Contributions that do not conform are likely to be ridiculed and rejected until they do. ;-)

Violations of these rules in the existing code are not excuses to follow suit. If code is seen that doesn't conform, it may and should be fixed.

Code Organization:

Code that is potentially useful to another application, or that may be abstracted in such a way that it is useful to other applications, should be put in a library and not in application directories.

C files use the .c extension. Header files use the .h extension. C++ files use the .cpp extension. PHP files use the .php extension. Tcl/Tk files use the .tcl/.tk extensions. POSIX Bourne-style shell scripts use the .sh extension. Perl files use the .pl (program) or .pm (module) extensions.

Source files go into the src/ directory on the top level, into a corresponding directory for the category of binary or library that it belongs to. Documentation files go into the doc/ directory on the top level, with the exception of manual (man) pages that should be colocated with any corresponding source files.

Header files private to a library go into that library's directory. Public header files go into the include/ directory on the top level. Public header files should not include any headers that are private. Headers should include any other headers that they require for correct parsing (this is an on-going clean-up effort). Public header files should not include the common header.

Headers should be included in a particular order. That order is generally as follows:

- any single "interface" header [optional]
- the common header (unless the interface header includes it)
- system headers
- public headers
- private headers

Applications may optionally provide an interface header that defines common structures applicable to most or all files being compiled for that application. That interface header will generally be the first file to be included, as it usually includes the common header and system headers. The common header should always be included before any system header. Standard C system headers should be included before library system headers. Headers should be written to be self-contained, not requiring other headers to be necessarily included before they may be used. If another header is necessary for a header to function correctly, it should include it.

Build System:

The CMake build system (more specifically, compilation test macros defined in misc/CMake/BRLCAD_CheckFunctions.cmake) should be used extensively to test for availability of system services such as standard header files, available libraries, and data types. No assumptions should be made regarding the availability of any particular header, function, datatype, or other resource. After running cmake, there will be an autogenerated include/brlcad_config.h file that contains many preprocessor directives and type declarations that may be used where needed.

Generic platform checks (e.g. #ifdef unix, #ifdef _WIN32) are highly discouraged and should generally not be used. Replace system checks with tests for the actual facility being utilized instead.

The Windows platform utilizes its own manually-generated configure results header (include/config_win.h) that has to be manually updated if new tests are added to the CMake build logic.

Only the BRL-CAD sources should include and utilize the common.h header. They should not include brlcad_config.h or config_win.h directly. If used, the common.h header should be listed before any system headers.

Language Compliance:

Features of C that conform to the ISO/IEC 9899-1990 C standard (C90) are generally the baseline for strict language conformance. As BRL-CAD used to be K&R syntax conformant, there remains an on-going effort to ensure a full conversion to a standards compliant ISO C implementation.

Code Conventions:

Globals variables, structures, classes, and other public data containers are highly discouraged within application code. Do not add any new globals to existing libraries. Globals are often a quick solution to some deeper coding problem. However, they carry significant maintenance costs, introduce (spaghetti) code complexity, make multi-threading support more costly, pollute public API (symbol-wise at a minimum), increase security risks, are error-prone to use, and usually complicate refactoring and code restructuring in the future. Using static variables (whether function- or static/file-scoped) is a viable alternative. Restructuring the logic to not be stateful is even better.

Functions should always specify a return type, including functions that return int or void. ANSI C prototypes should be used to declare functions, not K&R function prototypes.

Exact floating point comparisons are unreliable without requiring IEEE-compliant floating point math, but BRL-CAD does not require such math for portability and performance reasons. When floating point

comparisons are necessary, use the NEAR_EQUAL and NEAR_ZERO macros with a specified tolerance or the EQUAL and ZERO macros where a tolerance is indeterminate (all the macros are available by including vmath.h). Examples:

For known tolerances:

* instead of "foo == 2.0" use "NEAR_EQUAL(foo, 2.0, tol)"
* instead of "foo != 0.0" use "foo !NEAR_ZERO(foo, tol)"

For indeterminate tolerances:

* instead of "foo == 2.0" use "EQUAL(foo, 2.0)"
* instead of "foo != 0.0" use "foo !ZERO(foo)"

There are several functions whose functionality are either wrapped or implemented in a cross-platform manner by libbu. This includes functions related to memory allocation, command option parsing, logging routines, and more. The following functions and global variables should be utilized instead of the standard C facility:

bu_malloc() instead of malloc()
bu_calloc() instead of calloc()
bu_realloc() instead of realloc()
bu_fgets() instead of fgets()
bu_free() instead of free()
bu_log() instead of printf()
bu_bomb() instead of abort()
bu_exit() instead of printf()+exit()
bu_dirname() instead of dirname()
bu_getopt() instead of getopt()
bu_opterr instead of opterr
bu_optind instead of optind
bu_optopt instead of optopt
bu_optarg instead of optarg
bu_strdup() instead of strdup()
bu_strlcat() instead of strcat(), strncat(), and strlcat()
bu_strlcpy() instead of strcpy(), strncpy(), and strlcpy()
bu_strcmp() and BU_STR_EQUAL() instead of strcmp()
bu_strcasecmp() and BU_STR_EQUIV() instead of stricmp()/strcasecmp()
bu_strncmp() instead of strncmp()
bu_strcasecmp() instead of strnicmp()/strncasecmp()
bu_file_delete() instead of unlink(), rmdir(), and remove()
bu_sort() instead of qsort()

Similarly, ANSI C functions are preferred over the BSD and POSIX interfaces. The following functions should be used:

memset() instead of bzero()
memcpy() instead of bcopy()

The code should strive to achieve conformance with the GNU coding standard with a few exceptions. One such exception is NOT utilizing the GNU indentation style, but instead utilizing the BSD KNF

indentation style which is basically the K&R indentation style with 4 character indents. The following examples should be strictly adhered to, if only for the sake of being consistent.

1) Indentation whitespace

Indents are 4 characters, tabs are 8 characters. There should be an emacs and vi local variables block setting at the end of each file to adopt, enforce, and otherwise remind one of this convention. The following lines should be in all C and C++ source and header files at the end of the file:

```
/*
 * Local Variables:
 * mode: C
 * tab-width: 8
 * indent-tabs-mode: t
 * c-file-style: "stroustrup"
 * End:
 * ex: shiftwidth=4 tabstop=8
 */
```

In emacs, the 'indent-region' command (bound to C-M-\ by default) does a good job of making the needed changes to conform to this convention. Vi can be configured to respect the ex: modeline by adding 'set modeline=1' to your .vimrc configuration file. Microsoft Visual Studio should have tabs size set to 8 and indent size set to 4 with tabs kept under Tools -> Options -> Text Editor -> C/C++ -> Tabs. The exTabSettings project will also make MSVC conform by reading our file footers.

A similar block can be used on source and script files in other languages (such as Tcl, Shell, Perl, etc.). See the local variable footer script in sh/footer.sh to automatically set/update files.

Here is an example where '.' represents a literal space character (0x20) and '[ ]' represents a literal tab character (0x09):

```
int
main(int ac, char *av[])
{
....int i;
....int foo = 0;

....for (i = 0 ; i < 10; i++) {
[       ]foo += 1;
[       ]if (foo % 2) {
[       ]....printf("hello\n");
[       ]....if (foo > 5) {
[       ][      ]printf("world\n");
[       ]....}
[       ]}
....}
....return 0;
```

```
}
```

We may change this at some point in the future, but this is the style for now. If this is confusing, use spaces to indent and run sh/ws.sh to convert spaces to tabs. We value consistency in order to preserve maintainability.

2) Stylistic whitespace

No space immediately inside parentheses.

```
  while (1) { ...                   /* ok */
  for (i = 0; i < max; i++) { ...   /* ok */
  while ( max ) { ...               /* discouraged */
```

Commas and semicolons are followed by whitespace.

```
  int main(int argc, char *argv[]); /* ok */
  for (i = 0; i < max; i++) { ...   /* ok */
```

Operators and arguments generally are separated with whitespace.

```
  if (FLAG & MORE_FLAGS) { ...      /* ok */
  for (i = 0; i < max; i++) { ...   /* ok */
  if (FLAG&MORE_FLAGS) { ...        /* discouraged */
  for (i=0; i<max; i++) { ...       /* discouraged */
```

No space on arrow operators.

```
  structure->member = 5;            /* ok */
  structure -> member = 5;          /* bad */
```

Native language statements (if, while, for, switch, and return) have a separating space, functions do not.

```
  int my_function(int i);           /* ok, no space */
  while (argc--) ...                /* ok, has space */
  if( var == val )                  /* discouraged */
  switch(foo) ...                   /* discouraged */
```

Comments should have an interior space and be without tabs.

```
  /** good single-line doxygen */
  /* good */
  /*bad*/
  /*  discouraged */
  /*  discouraged  */
  /**
   * good:
   * multiple-line doxygen comment
   */
```

3) Braces

BRL-CAD uses the "The One True Brace Style" from BSD KNF and K&R. Opening braces should be on the same line as their statement, closing braces should line up with that same statement. Functions, however, are treated specially and we place their opening braces on separate lines. See http://en.wikipedia.org/wiki/Indent_style for details.

```
int
some_function(char *j)
{
    for (i = 0; i < 100; i++) {
      if (i % 10 == 0) {
          j += 1;
      } else {
          j -= 1;
      }
    }
}
```

4) Names

Variable and public API function names should almost always begin with a lowercase letter.

```
double localVariable; /* ok */
double LocalVariable; /* bad (looks like class or constructor) */
double _localVar;     /* bad (looks like member variable)      */
```

Variables are not to be "decorated" to show their type (i.e., do not use Hungarian notation or variations thereof) with a slight exception for pointers on occasion. The name should use a concise, meaningful name that is not cryptic (typing a descriptive name is preferred over someone else hunting down what was meant).

```
char *name;    /* ok  */
char *pName;   /* discouraged for new code, but okay */
char *fooPtr;  /* bad */
char *lpszFoo; /* bad */
```

Constants should be all upper-case with word boundaries optionally separated by underscores.

```
static const int MAX_READ = 2;  /* ok  */
static const int arraySize = 8; /* bad */
```

Public API (global) function names are in lowercase with underscores to separate words. Most functions within the core libraries are named with the following convention: [library]_[group]_[action]

```
bu_vls_strcat()
bn_mat_transpose()
```

Naming exceptions are allowed where the API intentionally mirrors some other familiar programming construct (e.g., bu_malloc()+bu_free()), but care should otherwise be taken to be as consistent as possible within a file and across a library's API.

Here are some naming convention pairings commonly used:

```
Allocation   => alloc()
Deallocation => free()

Initialization      => init()
De/Reinitialization => clear()
```

```
  Allocation + Init      => create()   (new for C++)
  Deinitialize + Dealloc => destroy()  (delete for C++)

  Resource acquire => open()
  Resource release => close()
```

5) Debugging

Compilation preprocessor defines should never change the size of structures.

```
  struct Foo {
  #ifdef DEBUG_CODE  // bad
    int _magic;
  #endif
  };
```

6) Comments

"//" style comments are not allowed in C source files for portability. Comment blocks should utilize an asterisk at the beginning of each new line. Doxygen comments should start on the second line unless it's a succinct /** single-line */ comment.

```
/* This is a
 * comment block.
 */

/**
 * This is a doxygen comment.
 */
```

7) Line length

We do not impose a fixed line length for source code. Comments blocks are formatted to column 70.

Long 'if' statements and function prototypes are okay. Expressions and function arguments are sometimes separated one per line where it helps readability, but reducing the complexity of expressions and number of function arguments is usually better.

DOCUMENTATION

BRL-CAD has extensive documentation in various formats and presently maintained in various locations. It is an on-going desire and goal of the project to have all documentation located along with the source code in our Subversion (SVN) repository.

In line with that goal and where beneficial, a large portion of the tutorial documentation is being converted to the DocBook XML format. Having the tutorial documentation in the DocBook XML format allows for easier maintenance, better export conversion support, and

representation in a textual format that may be revision controlled and tracked.

Documenting Source Code:

The source code should always be reasonably documented, this almost goes without saying for any body of code that is to maintain some longevity. Determining just how much documentation is sufficient and how much is too much is generally resolved over time, but it should generally be looked at from the perspective of "If I look at this code in a couple years from now, what would help me remember or understand it better?" and add documentation accordingly.

All public library functions and most private or application functions should be appropriately documented using Doxygen/Javadoc style comments. Without getting into the advanced details, this minimally means that you need to add an additional asterisk to a comment that precedes your functions:

```
/**
 * Computes the answer to the meaning of life, the universe, and
 * everything.
 */
int
the_answer(void)
{
    return 42;
}
```

TESTING & DEBUGGING

BRL-CAD has extensive testing infrastructure in place to ensure tools and APIs working a particular manner keep working in an expected way. A testing failure usually indicates an unintended change to behavior that must be reviewed with the provoking code justified, reverted, or (most commonly) fixed. There are system integration, performance, regression, and unit tests accessible through three primary build targets.

The BRL-CAD Benchmark, contained in the bench/ directory, validates critical raytrace library behavior:

```
  make benchmark
```

A series of system integration regression tests, described in a series of scripts in the regress/ directory, examines applications:

```
  make regress
```

API unit testing is contained within tests/ subdirectories of the respective library being tested (e.g., src/libbu/tests):

```
  make test
```

Note that changes to publicly documented tools and library APIs must adhere to BRL-CAD's change policy. This means changes to any tests may require thoughtful deployment. See the CHANGES file for details.

These tests are run nightly on BRL-CAD's server to catch problems quickly - developers should be aware of the status of BRL-CAD's tests as seen at http://brlcad.org/CDash

To add new regression tests, look at regress/weight.sh for an integration test example and src/libbu/tests/bu_vls_vprintf.c for an API unit test example. See their corresponding CMakeLists.txt build file for examples of how they are added to the build. Individual regression tests have separate regress-TARGET build targets defined (e.g., make regress-weight) to facilitate manual testing.

In addition to benchmark, regression, and unit testing, individual programs may be tested after a build without installing by running them from the build directory. Typically, installed binaries will be found in the bin/ subdirectory of the top-level build directory, e.g.:

```
  ./bin/mged
```

Binaries not intended for installation (i.e., marked NO_INSTALL in the CMakeLists.txt file) are in a build path corresponding to the source code location, e.g. the unit tests in src/libbu/tests compile into binaries in the path/to/builddir/src/libbu/tests build directory.

A profile build, useful for some types of performance inspection (e.g., gprof) but not enabled by default, may be specified via:

```
   cmake .. -DBRLCAD_ENABLE_PROFILING=ON
```

Debug symbols are enabled by default, even for installation, for all libraries and binaries in order to facilitate diagnosing and inspecting problems. Graceful crashes (i.e., an application "bomb") may result in a crash log (named appname-#####-bomb.log) getting written out on platforms that have the GNU Debugger (gdb) available. On most platforms, gdb can also be interactively utilized to inspect a reproducible crash (graceful or otherwise):

```
$ gdb --args bin/mged -c file.g
...
gdb> break bu_bomb
gdb> run
...[interact with application until it crashes or bombs]
gdb> backtrace
```

PATCH SUBMISSION GUIDELINES

To contribute to BRL-CAD, begin by submitting modifications to the

patches section at http://sf.net/projects/brlcad/. Patches in the unified diff format (diff -u) are generally preferred when modifying existing source or other text files. Otherwise, contributors are welcome to submit their changes in full to the patches tracker as compressed file attachments.

All text patches should be submitted in the unified diff format where it's feasible to create one either using SVN or using the unmodified original file. This is generally the "-u" option to diff, and is also supported by the SVN diff command:

```
svn diff > mychanges.patch
```

Where possible, patch files should be generated against the latest sources available to make it easier to review and apply the changes. If a modification involves the addition or removal of files, those files can be provided separately with instructions on where they belong (or what should be removed). If SVN cannot be used, please provide the complete release and build number of the files you worked with.

ANY MODIFICATIONS THAT ARE PROVIDED MUST NOT MODIFY OR CONFLICT WITH THE EXISTING "COPYING" FILE DISTRIBUTION REQUIREMENTS. This means that most modifications must be LGPL-compatible. Contributors are asked to only provide patches that may legally be incorporated into BRL-CAD under the existing distribution provisions described in the COPYING file.

Patches that are difficult to apply are difficult to accept.

BUGS & UNEXPECTED BEHAVIOR

When a bug or unexpected behavior is encountered that cannot be quickly fixed, it needs to be documented in our BUGS file or more formally reported to the SourceForge bug tracker at:

http://sourceforge.net/tracker/?atid=640802&group_id=105292&func=browse

The tracker is the main source for user-reported bugs and/or any issues that require significant discussion or benefit from having their status publicly announced. Issues listed in BUGS file are not necessarily listed in the tracker, and vice-versa. The BUGS file is also a convenience notepad of long and short term development issues.

COMMIT ACCESS

Commit access is evaluated on a person-to-person basis at the discretion of existing contributors. Commit access is generally granted after a contributor demonstrates strong competency with our developer guidelines and an existing developer with commit access

vouches for the new developer.

If you would like to have commit access, do not ask for it. Getting involved with the other contributors and making patches will result in automatic consideration for commit access. That said, the following steps represent a minimum that needs to occur in order for commit access to be granted:

1) Read this file completely.

2) Be able to compile BRL-CAD successfully from an SVN trunk checkout.

3) Join the brlcad-devel developer mailing list, introduce yourself.

4) Create a SourceForge account, submit at least two patches that demonstrate competency with our coding style, apply flawlessly, and provide some significant improvement.

5) Get to know the other developers. One of them will need to vouch for your commit access.

*) Be a nice person. ;)

Those with commit access have a responsibility to ensure that other developers are following the guidelines that have been described in this developer's guide within reasonable expectations. A basic rule of thumb is: don't break anything.

CONTRIBUTOR RESPONSIBILITIES

Contributors of BRL-CAD that are granted commit access are expected to uphold standards of conduct and adhere to conventions and procedures outlines in this guide. All contributors are directly supporting the on-going development of BRL-CAD by being granted commit access. As such, these individuals are also considered "developers" whether they are modifying source code or not, and have similar obligations and expectations. To summarize some of the expected responsibilities:

0) Primum non nocere. All contributors are expected in good faith to help, or at least to do no harm. Be helpful and respectful.

1) Developers are expected to uphold the quality, functionality, maintainability, and portability of the source code at all times. In part, this means that changes should be tested to avoid breaking the build and short-term fixes are discouraged. Committing code that is actively being worked on is encouraged but care should be taken to minimize impact on others and to respond quickly when an issue arises.

2) Bugs, typos, and compilation errors are to be expected as part of the process of active software development and documentation, but it is ultimately unacceptable to allow them to persist. If it is discovered that a recent modification introduces a new problem, such

as causing a compilation portability failure, then it is the responsibility of the contributor that introduced the change to assist in resolving the issue promptly. It is the responsibility of all developers to address issues as they are encountered regardless of who introduces the problem.

3) Contributors making commits to the repository are required to uphold the legal conventions and requirements summarized in the COPYING file. This includes the implicit assignment of copyright to the U.S. Government on all contributed code unless otherwise explicitly arranged as well as the usage of appropriate license headers in all files where expected. Contributors are also expected to denote appropriate credits into the AUTHORS file when applying contributed code and patches.

4) It is expected that developers will adhere to the coding style conventions and filesystem organization outlined in this developer's guide. This includes the utilization of the specified coding style as well as the prescribed K&R indentation convention.

5) Contributors are expected to communicate with other contributors and to avoid code or file territorialism. All contributors are expected and encouraged to fix problems as they are found regardless of where in the package they are located. Care should of course be taken to ensure that fixing in a bug in a section of code does not introduce other issues, especially for unfamiliar code. All contributors are expect to communicate their changes publicly by keeping documentation up-to-date, including making note of user-visible changes in the NEWS file following the inscribed format convention.

VERSION NUMBERS & COMPATIBILITY

The BRL-CAD version number is in the include/conf directory in the MAJOR, MINOR, and PATCH files. The README, ChangeLog, and NEWS files as well as a variety of documents in the doc/ directory may also make references to version numbers. See the MAKING A RELEASE steps listed below for a more concise list of what needs to be updated.

Starting with release 7.0.0, BRL-CAD has adopted a three-digit version number convention for identifying and tracking future releases. This number follows a common convention where the three numbers represent:

    {MAJOR_VERSION}.{MINOR_VERSION}.{PATCH_VERSION}

All "development" builds use an odd number for the minor version. All "release" builds use an even number for the minor version. Patched versions should include a release count:

    {MAJOR_VERSION}.{MINOR_VERSION}.{PATCH_VERSION}[-{RELEASE_COUNT}]

The MAJOR_VERSION should only increment when it is deemed that a

significant amount of major changes have accumulated, new features have been added, or enough significant backwards incompatibilities were added to make a release warrant a major version increment. In general, releases of BRL-CAD that differ in the MAJOR_VERSION are not considered compatible with each other.

The MINOR_VERSION is updated more frequently and serves the dual role as previously mentioned of easily identifying a build as a public release. A minor version update is generally issued after significant development activity (generally several months of activity) has been tested and deemed sufficiently stable.

The PATCH_VERSION may and should be updated as frequently as is necessary. Every public maintenance release should increment the patch version. Every development version modification that is backwards incompatible in some manner should increment the patch version number.

If it becomes necessary to update a posted release, use and increment the RELEASE_COUNT. The first posted release is implicitly the "-0" release count (e.g., 7.10.2 is implicitly 7.10.2-0) with subsequent updated releases incrementing the count (e.g., 7.10.2-1).

NAMING A SOURCE RELEASE

In order to achieve some consistency when preparing a source release, the following format should be used as the filename convention:

```
brlcad-{VERSION}.{EXTENSION}
```

VERSION is the usual version triplet described above under the section entitled VERSION NUMBERS & COMPATIBILITY. Example: 7.12.4

EXTENSION is the filename extension. Compressed tar files should use the expanded form (i.e. not tgz, etc.) unless otherwise dictated as necessary convention by a source package management system. Examples:

```
tar.gz
tar.bz2
dmg
exe
zip
```

NAMING A BINARY RELEASE

In order to achieve some consistency when preparing a binary release, the following format should be used as the filename convention:

```
BRL-CAD_{VERSION}[_{OS}][_{VENDOR}][_{CPU}][_{NOTE}].{EXTENSION}
```

Notably, the filename should use 'BRL-CAD' instead of 'brlcad' for the name unless technically problematic, it should delimit sections of the filename with underscores instead of spaces, dots, or dashes, and should always include the version number and an extension for the file type. The optional _{OS}_{VENDOR}_{CPU} portion is a reverse (and simplified version) of the GNU autotools config.guess canonical host triplet identifier.

VERSION is the usual version triplet described above under the section entitled VERSION NUMBERS & COMPATIBILITY. Example: 7.12.4 or 7.10.2-1

OS is an optional identifier for the target operating system for binary distributions. It should only be used if the file extension is not already platform specific (e.g., .dmg already indicates Mac OS X, .exe indicates Windows, etc.). The OS name can be the same as the GNU autotools OS identifier without the version number. If there are other platform considerations like the version of libc or the version of the OS that need to be called out, they can be included. Examples:

  freebsd
  linux
  linux_glibc3
  solaris

VENDOR is an optional operating system qualifier that isn't necessary for most platforms. It's generally only useful for vendors that use customized versions of common operating systems. An example would be something like SGI using a fairly customized variant of Linux on their Altix line. In that case, it's useful to include the vendor: sgi

CPU is the hardware architecture identifier. It should be the first identifier in the config.guess triplet or the lowercase hardware identifier returned by uname -a (if any). Examples:

  x86
  x86_64
  ia64
  sparc
  ppc
  power5
  mips
  mips_64

NOTE can be just about anything but should only be used when absolutely necessary. As an example, a note might be used to indicate that a binary distribution is a partial or custom release. Examples:

  dll
  benchmark

EXTENSION is the filename extension. Compressed tar files should use the expanded form (i.e. not tgz, etc.) unless otherwise dictated as necessary convention by a binary package management system. Examples:

tar.gz
tar.bz2
dmg
exe
zip

PATCHING A RELEASE

Should it become necessary to patch a release that has already been posted and announced, the mechanism is to post patch files for the source release and update the uploaded release notes README file.

Example: stop_rt_crashing-0.patch and fix_fbserv-1.patch

It's expected that all patch files are independent and will be applied sequentially. They should be consistently and incrementally numbered. Users should be instructed in the release notes README to download and apply all available patch files.

For binary releases, it's recommended to just "move on" and let issues become resolved in the next release unless there's a critical security or significant data corruption issue involved. If it becomes necessary to repost a release, use the RELEASE_COUNT file name convention described in VERSION NUMBERS & COMPATIBILITY.

Example: BRL-CAD-7.12.2.dmg is superseded by BRL-CAD-7.12.2-1.dmg

Patched binary releases may be moved to the hidden attic folder if critical, though not necessary or recommended. They are preserved for historic record and should never be deleted once announced. Releases not yet been announced may be updated within two days of being posted without involving a patch.

MAKING A RELEASE

BRL-CAD is developed on a monthly iteration development schedule with a release planned (not always actualized) at the end of every month.

In order to make a release, the sources need to be appropriately documented, tested, synchronized, and tagged. BRL-CAD releases must pass strict verification and validation testing.

Any developer may cause a release to be made provided testing passes and the appropriate release steps are taken.

When a release intention is announced, it is recommended to sync the trunk sources to the RELEASE branch so appropriate review testing, repairs, and release steps can occur without being impacted by other development activity.

Release steps are as follows:

```
######################################################################
# 00: Notify developer mailing list of intention to release.

    echo "Release preparations are under way.  Trunk sources merged to
the RELEASE branch will be reviewed for sync to STABLE and release
tagging.  Please help test by compiling distcheck-full and running
benchmark, mged, and archer." | mail -s "Release preparations commencing"
brlcad-devel@lists.sourceforge.net -r "your@subscribed.address"

######################################################################
# 01: Sync trunk to RELEASE branch.

    # NON-AUTO: Review the log and obtain the last trunk merge
    # revision number from comments.
    svn log --stop-on-copy
https://svn.code.sf.net/p/brlcad/code/brlcad/branches/RELEASE | grep -E
'r[0-9]{5}'
    PREV=[[last_trunk_rev]]
    echo "PREV=$PREV"

    svn co svn://svn.code.sf.net/p/brlcad/code/brlcad/branches/RELEASE
brlcad.RELEASE
    cd brlcad.RELEASE
    svn merge https://svn.code.sf.net/p/brlcad/code/brlcad/trunk@$PREV
https://svn.code.sf.net/p/brlcad/code/brlcad/trunk@HEAD .

######################################################################
# 02: Test the merge.

    mkdir -p .build && cd .build
    cmake .. -DBRLCAD_BUNDLED_LIBS=ON -DCMAKE_BUILD_TYPE=Release && make

    # Manually check rt, mged, and archer.
    bin/rt        # should report usage with correct library versions
    bin/mged -c test.g "make sph sph ; draw sph ; rt"   # pops up a
sphere
    bin/mged      # displays gui, run: opendb test.g ; draw sph ; rt
    bin/archer    # displays gui, run: opendb test.g ; draw sph ; rt

    # clean up
    cd .. && rm -rf .build

######################################################################
# 03: Commit the merge.

    CURR=`svn log --xml
https://svn.code.sf.net/p/brlcad/code/brlcad/trunk | grep 'revision=' |
head -n 1 | sed 's/.*="\([0-9][0-9]*\)".*/\1/g'`
    echo "CURR=$CURR"

    svn commit -m "merge of trunk to RELEASE branch, r$PREV through
r$CURR"
```

```
########################################################################
# 04: Reprioritize or address remaining TODO items.

    # NON-AUTO: Do what's necessary to release, update priorities.

########################################################################
# 05: Review all commits.

    # NON-AUTO: Verify all user-visible changes are in NEWS.
    # NON-AUTO: Verify all interface changes are in CHANGES.
    # NON-AUTO: Verify usage/options are documented in doc/.

########################################################################
# 06: Update the version numbers.

    # NON-AUTO: NEWS
    # NON-AUTO: README
    # NON-AUTO: include/conf/PATCH
    # NON-AUTO: include/conf/MINOR (See VERSION NUMBERS & COMPATIBILITY
section.)
    # NON-AUTO: misc/debian/changelog
    # NON-AUTO: misc/macosx/Resources/ReadMe.rtfd/TXT.rtf
    # NON-AUTO: misc/macosx/Resources/Welcome.rtfd/TXT.rtf

########################################################################
# 07: Update ChangeLog.  Use the YYYY-MM-DD of previous NEWS entry.

    LAST=`grep -E "\--- [0-9]{4}-[0-9]{2}-[0-9]{2}" NEWS | head -n 2 |
tail -n 1 | awk '{print $2}'`
    echo "LAST=$LAST"
    LAST=r57446
    svn2cl --break-before-msg --include-rev --stdout -r HEAD:{$LAST} >
ChangeLog
    svn commit -m "update log with commits through $LAST" ChangeLog

########################################################################
# 08: Run distcheck-full on at least two platforms.

    mkdir -p .build && cd .build && cmake .. && make distcheck-full

########################################################################
# 09: Sync RELEASE to STABLE branch:

    # Update the NEWS release date for the current release to today's
date

    # NON-AUTO: Review the log and obtain the last merge revision
    # number from comments.
    svn log --stop-on-copy
https://svn.code.sf.net/p/brlcad/code/brlcad/branches/STABLE | less
    PREV=[[last_trunk_rev]]
      ### OR ###
```

```
    PREV=`svn log --xml --stop-on-copy
https://svn.code.sf.net/p/brlcad/code/brlcad/branches/STABLE | grep
'revision=' | head -n 1 | sed 's/.*revision="\([0-9][0-9]*\)".*/\1/g'`
    echo "PREV=$PREV"

    # merge that range of changes into STABLE
    svn co https://svn.code.sf.net/p/brlcad/code/brlcad/branches/STABLE
brlcad.STABLE && cd brlcad.STABLE
    svn merge
https://svn.code.sf.net/p/brlcad/code/brlcad/branches/RELEASE@$PREV
https://svn.code.sf.net/p/brlcad/code/brlcad/branches/RELEASE@HEAD .

######################################################################
# 10: Test the merge.

    mkdir .merge && cd .merge
    cmake .. -DBRLCAD_BUNDLED_LIBS=ON -DCMAKE_BUILD_TYPE=Release && make
distcheck-full && cd .. && rm -rf .merge

######################################################################
# 11: Commit the merge.

    CURR=`svn log --xml
https://svn.code.sf.net/p/brlcad/code/brlcad/branches/RELEASE | grep
'revision=' | head -n 1 | sed 's/.*="\([0-9][0-9]*\)".*/\1/g'`
    echo "CURR=$CURR"

    svn commit -m "merging RELEASE branch to STABLE branch, r$PREV
through r$CURR"

######################################################################
# 12: Tag the release using "rel-MAJOR-MINOR-PATCH" format:

    MAJOR=`awk '{print $1}' include/conf/MAJOR`
    MINOR=`awk '{print $1}' include/conf/MINOR`
    PATCH=`awk '{print $1}' include/conf/PATCH`
    echo "Tagging rel-$MAJOR-$MINOR-$PATCH"
    svn cp https://svn.code.sf.net/p/brlcad/code/brlcad/branches/STABLE
https://svn.code.sf.net/p/brlcad/code/brlcad/tags/rel-$MAJOR-$MINOR-
$PATCH

######################################################################
# 13: Increment and commit the next BRL-CAD release numbers to SVN.

    # Update the include/conf/(MAJOR|MINOR|PATCH) version files
    # immediately to odd-numbered minor version or new patch developer
    # version (e.g. 7.11.0 or 7.34.1).  Update README and NEWS version
    # to next *expected* release number (e.g. 7.12.0 or 7.34.2).

    echo "`expr $PATCH + 1`" > include/conf/PATCH
    NEXT="`expr $PATCH + 2`"
    perl -pi -e "s/Release [0-9]+\.[0-9]+\.[0-9]+/Release
$MAJOR.$MINOR.$NEXT/" README
```

```
    EXPR="s/(@---[[:space:]@-]*[0-9]{4}-)([0-9]{2}-[0-
9]{2})([[:space:]]*Release[[:space:]]*${MAJOR}\.${MINOR}\.)(${PATCH})([[:
space:]@-]*-@)/\$1XX-XX\${3}${NEXT}\$5@* TBD@@\1\2\3\4\5/"
    cat NEWS | tr '\n' '@' | perl -pi -e "$EXPR" | tr '@' '\n' > NEWS
    svn commit -m "bump to next development revision after tagging the
$MAJOR.$MINOR.$PATCH release" ../include/conf/PATCH

######################################################################
# 14: Sync RELEASE changes back to trunk.

    # Work-in-progress

######################################################################
# 15: Obtain a tagged version of the sources from the repository, make
# final distribution tarballs:

    svn checkout https://svn.code.sf.net/p/brlcad/code/brlcad/tags/rel-
$MAJOR-$MINOR-$PATCH brlcad-$MAJOR.$MINOR.$PATCH
    cd brlcad-$MAJOR.$MINOR.$PATCH-build && mkdir .build && cd .build
    cmake .. -DBRLCAD_BUNDLED_LIBS=ON -DCMAKE_BUILD_TYPE=Release && make
distcheck-full

######################################################################
# 16: Upload the source distributions and release notes.  Use source
# tarballs to create binaries.

    # create a shell session and source dir
    SFUSERNAME=`ls ~/.subversion/auth/svn.simple/* | xargs -n 1 grep -A4
sourceforge | tail -1`
    echo "SFUSERNAME=$SFUSERNAME MAJOR=$MAJOR MINOR=$MINOR PATCH=$PATCH"
    ssh -v $SFUSERNAME,brlcad@shell.sf.net create
    ssh -v $SFUSERNAME,brlcad@shell.sf.net mkdir
"/home/frs/project/brlcad/BRL-CAD\ Source/$MAJOR.$MINOR.$PATCH"

    # create binary dirs (as needed)
    ssh -v $SFUSERNAME,brlcad@shell.sf.net mkdir
"/home/frs/project/brlcad/BRL-CAD\ Runtime\
Libraries/$MAJOR.$MINOR.$PATCH"
    ssh -v $SFUSERNAME,brlcad@shell.sf.net mkdir
"/home/frs/project/brlcad/BRL-CAD\ for\ BSD/$MAJOR.$MINOR.$PATCH"
    ssh -v $SFUSERNAME,brlcad@shell.sf.net mkdir
"/home/frs/project/brlcad/BRL-CAD\ for\ Linux/$MAJOR.$MINOR.$PATCH"
    ssh -v $SFUSERNAME,brlcad@shell.sf.net mkdir
"/home/frs/project/brlcad/BRL-CAD\ for\ Mac\ OS\ X/$MAJOR.$MINOR.$PATCH"
    ssh -v $SFUSERNAME,brlcad@shell.sf.net mkdir
"/home/frs/project/brlcad/BRL-CAD\ for\ Windows/$MAJOR.$MINOR.$PATCH"

    # upload source dist and any binaries
    scp brlcad-$MAJOR.$MINOR.$PATCH*
"$SFUSERNAME,brlcad@shell.sf.net:/home/frs/project/brlcad/BRL-CAD\
Source/$MAJOR.$MINOR.$PATCH/."

    # extract and upload release notes to source dir
```

```
    cat ../NEWS | tr '\n' '@' | perl -p -e "s/.*?(@---[[:space:]@-]*[0-
9]{4}-[0-9]{2}-[0-
9]{2}[[:space:]]*Release[[:space:]]*${MAJOR}\.${MINOR}\.${PATCH}[[:space:
]@-]*.*?)@---.*/\1/" | tr '@' '\n' > README-${MAJOR}-${MINOR}-
${PATCH}.txt
    echo "Release notes for $MAJOR.$MINOR.$PATCH" && echo "==="
    cat README-$MAJOR-$MINOR-$PATCH.txt && echo "==="
    scp README-$MAJOR-$MINOR-$PATCH.txt
"$SFUSERNAME,brlcad@shell.sf.net:/home/frs/project/brlcad/BRL-CAD\
Source/$MAJOR.$MINOR.$PATCH/."

    # NON-AUTO: Following the NAMING A BINARY RELEASE convention,
    # upload any release binaries and platform-specific release notes.

    scp README-$MAJOR-$MINOR-$PATCH.txt
"$SFUSERNAME,brlcad@shell.sf.net:/home/frs/project/brlcad/BRL-CAD\
Runtime\ Libraries/$MAJOR.$MINOR.$PATCH/."
    scp README-$MAJOR-$MINOR-$PATCH.txt
"$SFUSERNAME,brlcad@shell.sf.net:/home/frs/project/brlcad/BRL-CAD\ for\
BSD/$MAJOR.$MINOR.$PATCH/."
    scp README-$MAJOR-$MINOR-$PATCH.txt
"$SFUSERNAME,brlcad@shell.sf.net:/home/frs/project/brlcad/BRL-CAD\ for\
Linux/$MAJOR.$MINOR.$PATCH/."
    scp README-$MAJOR-$MINOR-$PATCH.txt
"$SFUSERNAME,brlcad@shell.sf.net:/home/frs/project/brlcad/BRL-CAD\ for\
Mac\ OS\ X/$MAJOR.$MINOR.$PATCH/."
    scp README-$MAJOR-$MINOR-$PATCH.txt
"$SFUSERNAME,brlcad@shell.sf.net:/home/frs/project/brlcad/BRL-CAD\ for\
Windows/$MAJOR.$MINOR.$PATCH/."

    # NON-AUTO: Be sure to mark binaries as default download, then
    # close shell session
    ssh -v $SFUSERNAME,brlcad@shell.sf.net shutdown

#########################################################################
# 17: Notify binary platform maintainers:

    T2 package maintainer
      http://t2-project.org/packages/brlcad.html

    OpenSUSE package maintainer
        https://build.opensuse.org/package/users/Education/brlcad

    FreeBSD ports maintainer
      http://www.freebsd.org/cgi/cvsweb.cgi/ports/cad/brlcad/

    Gentoo portage maintainer
      http://packages.gentoo.org/package/media-gfx/brlcad

    Ubuntu/Debian .deb maintainer
      Jordi Sayol <g.sayol@yahoo.es>

    Debian apt package maintainer
      http://git.debian.org/?p=debian-science/packages/brlcad.git
```

```
    Slackware maintainer
      http://slackbuilds.org/result/?search=brlcad

    Fedora maintainer
      https://fedoraproject.org/wiki/User:Germano#Contact

######################################################################
# 18: Announce the new release.

The NEWS file should generally be used as a basis for making release
announcements though the announcements almost always require
modification and customization tailored to the particular forum and
audience.  Always notify the following when a release is made:

    BRL-CAD Website  (authorized can submit)
      http://brlcad.org/d/node/add/story

    BRL-CAD NEWS Mailing List  (anyone can submit, posting moderated)
      brlcad-news@lists.sourceforge.net

    BRL-CAD SourceForge NEWS  (authorized can submit)
      https://sourceforge.net/p/brlcad/news/new

    BRL-CAD on Facebook  (authorized can submit)
      http://www.facebook.com/pages/BRL-CAD/387112738872
      short summary without news details (sentence format)

    BRL-CAD on Twitter  (authorized can submit)
      http://twitter.com/#!/BRL_CAD
      short summary without news details (whatever fits)

If appropriate, notify and/or update the following information outlets
with the details of the new release:

Linux release:

    CAD on Linux mailing list (plain text)
      cad-linux@freelists.org
      with linux-specific details

    CAD on Linux Dev mailing list
      cad-linux-dev@freelists.org
      with developer-centric details

    Linux Softpedia
      http://linux.softpedia.com/get/Multimedia/Graphics/BRL-CAD-
105.shtml

Mac OS X release:

    Versiontracker  (only 'brlcad' can update)
      http://www.versiontracker.com/dyn/moreinfo/macosx/26289
      http://www.versiontracker.com/dyn/moreinfo/win/64903
```

short without news details (list format)

Mac Softpedia (anyone can update)
http://mac.softpedia.com/user/pad.shtml
maceditor@softpedia.com
(either provide doc/pad_file.xml or e-mail)
http://mac.softpedia.com/get/Multimedia/BRLCAD.shtml

Fink package maintainer
jack@krass.com

Multiple platform major release and announcements:

CADinfo.NET
copyboy@cadinfo.net
with news details

TenLinks Daily
http://www.tenlinks.com/NEWS/tl_daily/submit_news.htm
news@tenlinks.com
with news details

Slashdot
http://slashdot.org
short without news details

DevMaster.net (if engine-related)
http://www.devmaster.net/news/submit.php

CADCAM Insider (plain text)
http://cadcam-insider.com/index.php/News-submission-guidelines.html
copyboy@cadcam-insider.com

```
#####################################################################
# 19: Sit back and enjoy a beverage for a job well done.
```

GETTING HELP

See the GETTING STARTED section above. Basically, communicate with the existing developers. There is an IRC channel, e-mail mailing lists, and on-line forums dedicated to developer discussions.