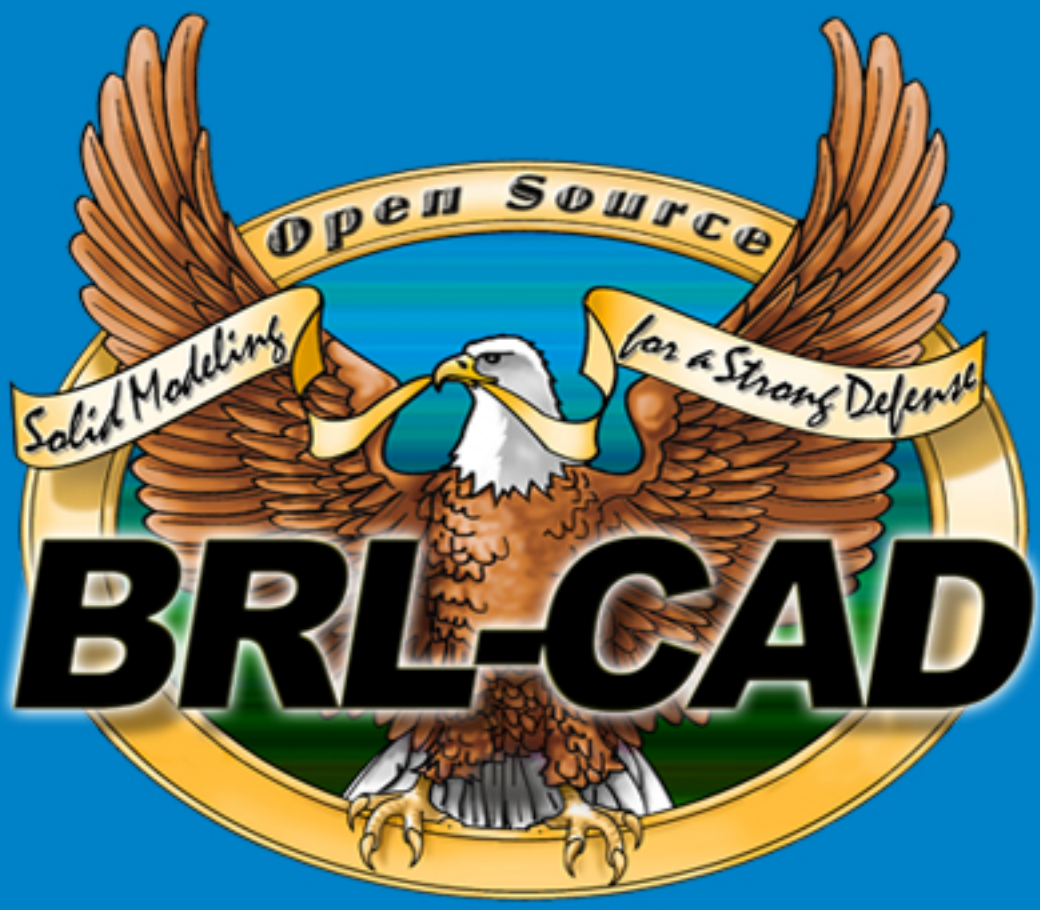
HACKING BRL-CAD

A Contributor's Guide



Build your own reality with BRL-CAD

A Call to Arms (and Contributors)

"The future exists first in the imagination, then in the will, then in reality." - Mike Muuss

Welcome to BRL-CAD! Whether you are a developer, documenter, graphic artist, academic, or someone who just wants to be involved in a unique open source project, BRL-CAD has a place for you. Our contributors come from all over the world and use their diverse backgrounds and talents to help maintain and enhance one of the oldest computer-aided design (CAD) packages used in government and industry today.

What is BRL-CAD?

BRL-CAD (pronounced be-are-el-cad) is a powerful, cross-platform, open source solid modeling system that includes interactive three-dimensional (3D) solid geometry editing, high-performance ray tracing support for rendering and geometric analysis, network-distributed framebuffer support, image and signal-processing tools, path tracing and photon mapping support for realistic image synthesis, a system performance analysis benchmark suite, an embedded scripting interface, and libraries for robust high-performance geometric representation and analysis.

For more than two decades, BRL-CAD has been the primary solid modeling CAD package used by the U.S. government to help model military systems. The package has also been used in a wide range of military, academic, and industrial applications, including the design and analysis of vehicles, mechanical parts, and architecture. Other uses have included radiation dose planning, medical visualization, terrain modeling, constructive solid geometry (CSG), modeling concepts, computer graphics education and system performance benchmark testing.

BRL-CAD supports a wide variety of geometric representations, including an extensive set of traditional implicit "primitive shapes" (such as boxes, ellipsoids, cones, and tori) as well as explicit primitives made from collections of uniform B-spline surfaces, non-uniform rational B-spline (NURBS) surfaces, n-manifold geometry (NMG), and purely faceted polygonal mesh geometry. All geometric objects may be combined using boolean set-theoretic CSG operations such as union, intersection and difference.

Overall, BRL-CAD contains more than 400 tools, utilities, and applications and has been designed to operate on many common operating system environments, including BSD, Linux, Solaris, Mac OS X, and Windows. The package is distributed in binary and source code form as Free Open Source Software (FOSS), provided under Open Source Initiative (OSI) approved license terms.

History and Vision

BRL-CAD was originally conceived and written by the late Michael Muuss, the inventor of the popular PING network program. In 1979, the U.S. Army Ballistic Research Laboratory (BRL) (the agency responsible for creating ENIAC, the world's first general-purpose electronic computer in the 1940s) identified a need for tools that could assist with the computer simulations and analysis of combat vehicle systems and environments. When no existing CAD package was found to be adequate for this specialized purpose, Mike and fellow software developers began developing and assembling a unique suite of utilities capable of interactively displaying, editing, and interrogating geometric models. Those early efforts subsequently became the foundation on which BRL-CAD was built.

Development of BRL-CAD as a unified software package began in 1983, and its first public release came in 1984. Then, in 2004, BRL-CAD was converted from a limited-distribution U.S. government-controlled code to an open source project, with portions licensed under the LGPL and BSD licenses.

The ongoing vision for BRL-CAD development is to provide a robust, powerful, flexible, and comprehensive solid modeling system that includes:

1. Faithful high-performance geometric representation.
2. Efficient and intuitive geometry editing.
3. Comprehensive conversion support for all solid geometry formats.
4. Effective geometric analysis tools for 3D CAD.

Key Strengths

All CAD packages are not alike. Among the many strengths of the BRL-CAD package are the following:

1. BRL-CAD is **open source**! Don't like something? You can make it better.
2. You can leverage **decades of invested development**. BRL-CAD is the most feature-filled open source CAD system available, with hundreds of years time invested.
3. **Your work will get used**. BRL-CAD is in production use and downloaded thousands of times every month by people all around the world.
4. You have the ability to create extensively **detailed realistic models**.
5. You can model objects on scales ranging from (potentially) the subatomic through the galactic, while essentially providing **all the details, all the time**.
6. You can leverage **one of the fastest** raytracers in existence (for many types of geometry).
7. You can convert to and from a wide range of **geometry file formats**.
8. BRL-CAD has a powerful, **customizable scripting interface** with many advanced editing and processing capabilities.

Creating Primitive Shapes

Launching the MGED Program

To launch the MGED program, type **mged** at the Terminal (tty) prompt and then press the **ENTER** key. This brings up two main windows: the MGED Command Window and the MGED Graphics Window (sometimes called the Geometry Window). Both windows will initially be blank, awaiting input from you. To leave the program at any time, at the Command Line type either the letter **q** or the word **quit** and then press the **ENTER** key.

Entering Commands in the Command Window

You can type in commands at the mged> prompt. Many experienced UNIX users prefer this method because it allows them to quickly create a model (which we sometimes refer to as a “design”) without having to point and click on a lot of options.

Using the GUI

Users who are more familiar with Microsoft Windows may prefer to use the GUI pull- down menus at the top of the Command or Graphics Window (they are the same in either window). The menus are divided into logical groupings to help you navigate through the MGED program.

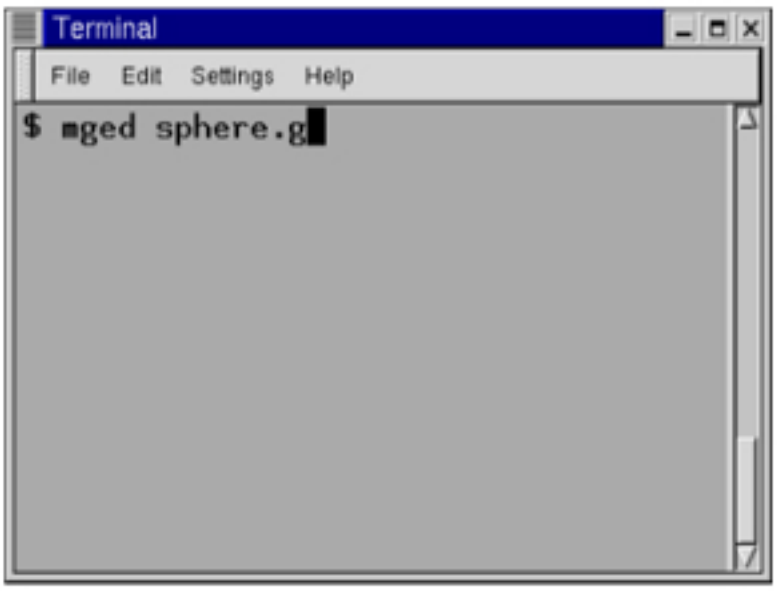
Before you can create a model, you need to open a new database either through the Terminal Window when starting MGED or through the GUI after starting MGED.

Opening or Creating a New Database when Launching MGED

When launching MGED, you can open or create a database at the same time. At the shell prompt (usually a $ or %), in the Terminal Window, type mged followed by a new or existing database name with a .g extension.

For example:

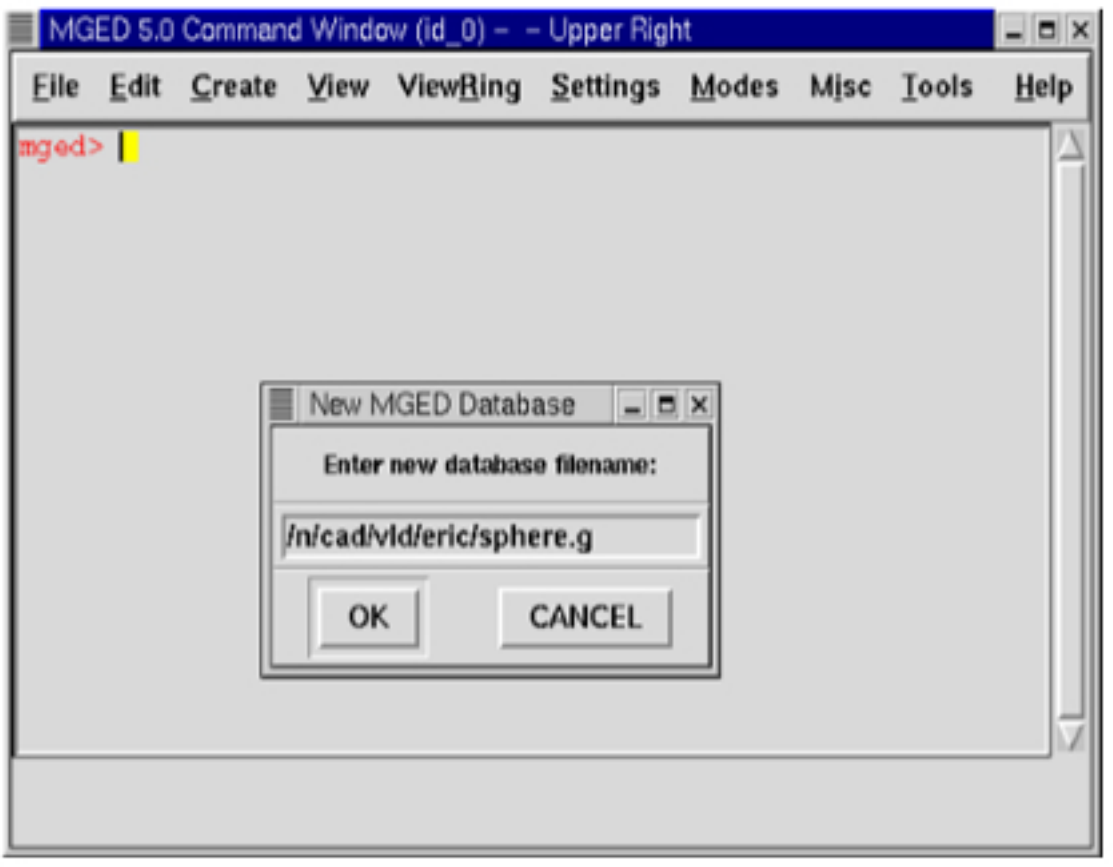
mged sphere.g<ENTER>



Using the GUI to Open or Create a Database

Alternatively, once you have launched MGED, you can open an existing database or create a new database using the GUI menus (at the top of the Command or Graphics Window) by clicking on **File** and then either **Open** or **New**. Both options bring up a small dialog box. The **Open** dialog box will ask you to type in the name of an existing database. The **New** dialog box will ask you to type in the name of a new database. Click on **OK** to accept the database.

For this lesson, create a new database called **sphere.g**. To do this, type **sphere.g** at the end of the path name, as shown in the following illustration. Click on **OK** to accept the database name.



One advantage to using the GUI, if you aren't familiar with UNIX file management, is that this will show you your current path name, so you will know exactly where your database is going to be located. This can be especially helpful if you have a lot of directories or files to manage.

Assigning a Title to Your Database

You can title your new database to provide an audit trail for you or others who might use your database. After the prompt, in the Command Window, type **title** followed by a space and a name that reflects the database you are going to make. When you are done, press the **ENTER** key. For example:

mged> **title MySphere<ENTER>**

Note that in BRL-CAD versions prior to release 6.0, the title is limited to 72 characters.

Selecting a Unit of Length

MGED uses millimeters for all internal mathematical processes; however, you can create your design using some other unit, such as feet. For this lesson, inches is used. To select inches, move your mouse pointer to the **File** menu at the top of the Command Window. Click on **File** and then **Preferences**. A new menu will appear. Select **Units** and then **Inches**. If you are not a "point-and-click" type of person and prefer a Command Line,then just type **units in** after the MGED prompt in the Command Window, followed by the **ENTER key**. The Command Line looks like:

mged> **units in<ENTER>**

Selecting a Primitive Shape

MGED provides a variety of primitive shapes (sometimes referred to as simply shapes or primitives) that you can use to build models. Each type of shape has parameters that define its position, orientation, and size.

Creating a Sphere from the Command Line

For this lesson, you are going to create a single sphere. There are two ways you can create a primitive shape. You can create all shapes through the Command Window and most shapes through the GUI.

You can easily create a sphere from the prompt in the Command Window by typing just a few commands. At the MGED prompt, type:

make sph1.s sph<ENTER>

A default sphere will be created, and a wireframe representation of the primitive shape will appear in the Graphics Window. In Lesson 4, you will give your sphere a solid, three-dimensional look.

This command will draw the primitive shape in the Graphics Window.

Clearing the Graphics Window

To build another object or work on another primitive shape, you can easily clear the Graphics Window through the Command Window. At the Command Line prompt, type an uppercase **Z** (for zap) followed by **ENTER**.

Erasing an Item from the Graphics Window

You may occasionally want to erase a particular item from the display in the Graphics Window. You can use the **erase** command to remove the item without any file operation being performed; the item remains in the database. To delete the **sph1.s** object from the display, at the Command Window prompt, type:

erase sph1.s<ENTER>

Working with Our Code

BRL-CAD consists of more than 1 million lines of source code spanning more than 20 foundation libraries and 400 application modules.

The majority of BRL-CAD is written in highly portable C and C++, with some GUI and scripting components written in Tcl/Tk. There is also some support for, and bindings to, other languages available. POSIX shell scripts are used for deployment integration testing. BRL-CAD uses the CMake build system for compilation and unit testing.

The Big Picture

The source code and most project data are stored in a Subversion version control system for change tracking and collaborative development. Trunk development is generally stable, but cross-platform compilation is not guaranteed. A separate branch (named STABLE) provides a higher level of quality assurance. Every released version of BRL-CAD is tested and tagged.

The project aims for an **It Just Works** approach to compilation whereby a functional build of BRL-CAD is possible without needing to install more than a compiler, CMake, and a build environment--for example, GNU Make or Microsoft Visual Studio. BRL-CAD provides all of the necessary third-party dependencies for download and compilation convenience within source distributions but by default will build using system versions of those dependencies if available.

As with any large system that has been under development for a number of years, there are vast sections of code that may be unfamiliar, uninteresting, or even daunting. Don't panic. BRL-CAD has been intentionally designed with layering and modularity in mind.

You can generally focus in on the enhancement or change that interests you without being too concerned with other portions of the code. You should, however, do some basic research to make sure what you plan to contribute isn't already in the BRL-CAD code base.

History of the Code

As mentioned previously, the initial architecture and design of BRL-CAD began in 1979. Development as a unified package began in 1983. The first public release was in 1984. And on December 21, 2004, BRL-CAD became an open source project.

BRL-CAD is a mature code base that has remained active over decades due to continual attention on design and maintainability. Since the project's inception, more than 200 people have directly contributed to BRL-CAD. The project has historically received support from numerous organizations within academia, commercial industry, various government agencies, and from various independent contributors. We credit all contributors in BRL-CAD's authorship documentation.

The following diagram illustrates how the number of lines of code in BRL-CAD has changed over time:

System Architecture

BRL-CAD is designed based on a UNIX methodology of the command-line services, providing many tools that work in harmony to complete a specific task. These tools include geometry and image converters, signal and image processing tools, various raytrace applications, geometry manipulators, and much more.

To support what has grown into a relatively large software system, BRL-CAD takes advantage of a variety of support libraries that encapsulate and simplify application development. At the heart of BRL-CAD is a multi-representation ray tracing library named LIBRT. BRL-CAD specifies its own file format (files with the extension .g or .asc) for storing information on disk. The ray tracing library uses a suite of other libraries for other basic application functionality.

Tenets of Good Software

BRL-CAD's architecture is designed 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 new development.

The code adheres to a published change deprecation and obsolescence policy whereby features that have been made publicly available are not removed without appropriate notification. Generally there should be a compelling motivation to remove any existing functionality, but improvements are encouraged.

BRL-CAD has a longstanding heritage of maintaining verifiable, validated, and repeatable results in critical portions of the package, particularly in the ray tracing library. BRL-CAD includes regression tests that will compare runtime behavior against known results and report any deviations from previous results as failures. Considerable attention is put into verification and validation throughout BRL-CAD. Incorrect behavior does not need to be preserved simply to maintain consistency, but it is rare to find genuine errors in the baseline testing results. So, anyone proposing such a behavior change will have to conclusively demonstrate that the previous result is incorrect.

Working with Our Documentation

BRL-CAD provides documentation in the following formats:

1. UNIX man pages.
2. HyperText Markup Language (HTML) for the web.
3. PDF for documents needing a well-defined, consistent appearance.

Our challenge is to maintain BRL-CAD's documentation in multiple formats. It is difficult enough to keep software documentation up to date without needing to update multiple documents using different formats that contain the same information. As well, it is not possible to supply documentation in a single format that works optimally on all platforms. For example, while UNIX man pages are standard across all UNIX and UNIX-like systems, most Windows systems will not understand that format and will require HTML versions of those documents.

Instead of using a mix of formats and tools, BRL-CAD uses the DocBook documentation format and toolchain to produce documentation in the range of required formats.

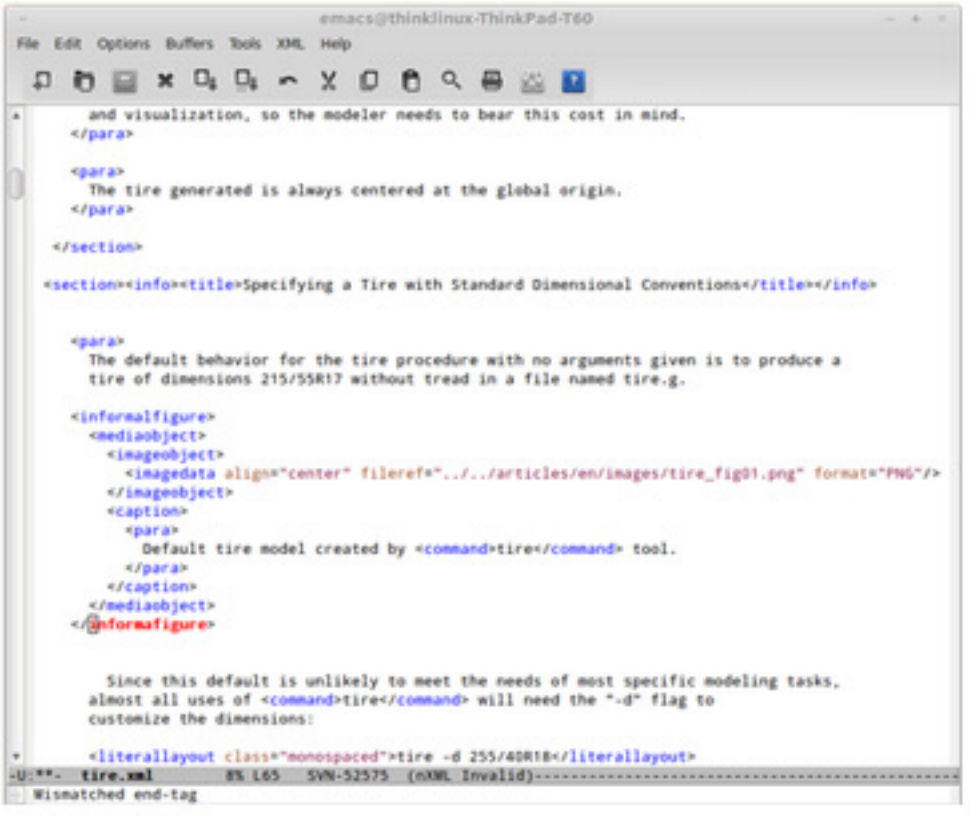
What is DocBook?

DocBook is a schema (a structured approach to organization of information) that uses the eXtensible Markup Language standard (XML) as its fundamental framework and builds atop that framework a vocabulary for describing the content and structure of technical documentation. BRL-CAD uses the DocBook 5.0 documentation format to describe its documentation. For detailed documentation for DocBook 5.0, see http://www.docbook.org/tdg5/en/html/docbook.html.

Tools for Working with DocBook

While you can write documentation in DocBook using WYSIWYG (What You See Is What You Get) editors, we require that a document saved to DocBook from an editing tool should be inspected for human readability and, if necessary, reformatted for simplicity.

If you are comfortable with working with DocBook XML directly, we recommend that you use the Emacs editor and its nXML module. nXML can automatically recognize and highlight mistakes in the structure of a document while you are editing. The following image illustrates nXML identifying an incorrect closing tag for an informal figure object:



Aside from error checking tools like nXML, the ability to pinpoint errors in a document's formatting is built into the BRL-CAD compilation process. That process uses a tool called xmllint to report incorrect formatting. When, for example, the error illustrated in the image above is encountered during BRL-CAD's build, xmllint produces the following error:

[40%] Validating DocBook source with xmllint:
/home/user/brlcad/doc/docbook/articles/en/tire.xml:65: parser error : Opening and ending tag mismatch: informalfigure line 54 and informafigure

</informafigure>

CMake Error at tire_validate.cmake:39 (message):

xmllint failure: 1

In this case, the error is reasonably informative. However, xmllint is not the only tool available for this sort of error checking. You can specify the following validation tools when you configure your environment:

1. Oracle Multi-Schema XML Validator (https://msv.java.net) - specified as msv on the command line.
2. oNVDL (http://sourceforge.net/projects/onvdl) - specified as nvld on the command line.
3. Relax NG Validator (rnv) (http://sourceforge.net/projects/rnv) - specified as rsv on the command line.

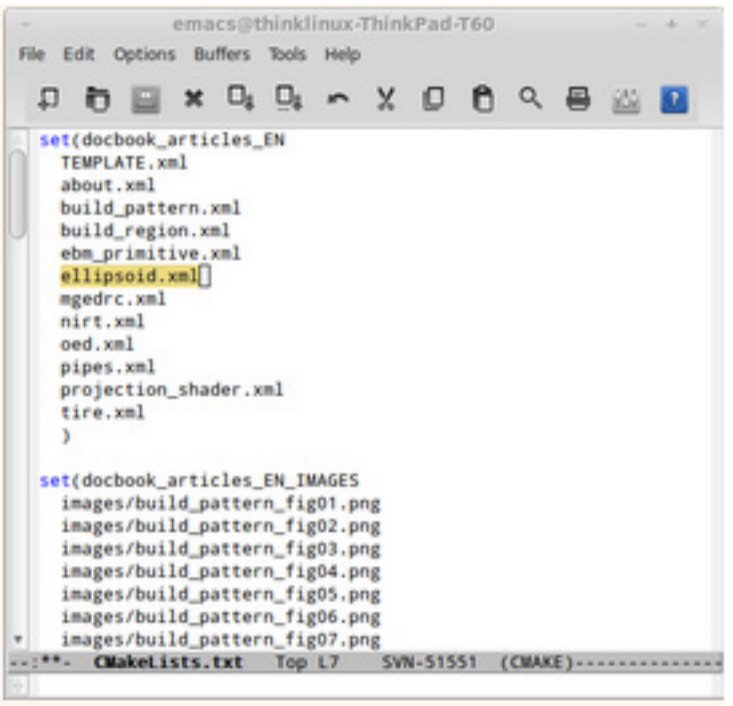
Adding a New Document to BRL-CAD

Because creating and editing DocBook documentation is greatly simplified by BRL-CAD's management of the conversion process, it is usually a good idea to add a new document to the build system at the beginning of the document creation and editing process. To do this, copy a template file from the source directories to the file name to be used for the new document.

For example, if you are writing a new DocBook article in English about the ellipsoid, use the following command to copy the article template to the filename ellipsoid.xml in the English articles directory:

~/brlcad $ cp doc/docbook/articles/en/TEMPLATE.xml doc/docbook/articles/en/ellipsoid.xml

Next, open the file doc/docbook/articles/en/CMakeLists.txt in a text editor. Then, add the name of the new document to the file to alert the build system of its existence:



BRL-CAD now knows about the new file and can generate output for it.

You will generally only want to rebuild a specific output (say, HTML) to confirm that output renders properly. To set up the specific targets for the new file, run the command below to refresh the build targets (in this example, the build output directory is called build):

~/brlcad/build $ cmake ..

This creates a new build target, ellipsoid_article_html, which will build only the HTML output of the document and its dependencies:

~/brlcad/build $ make ellipsoid_article_html

[0%] Built target printtimestamp
[0%] Built target buildtimestart

Build Time: Tue Oct 15 19:14:42 2013

[0%] Built target timestamp
[0%] Built target zlib
[100%] Built target xml
[100%] Built target xslt
[100%] Built target exslt
[100%] Built target xmllint
[100%] Built target xsltproc
[100%] Built target schema-expand
[100%] Built target fonts-dejavu-expand
[100%] Built target fonts-stix-expand
[100%] Built target offo-2-expand
[100%] Built target svg-dtd-expand
[100%] Built target xsl-expand
[100%] Built target docbook_articles_EN_IMAGES_cp
Scanning dependencies of target ellipsoid_article_html
[100%] Validating DocBook source with xmllint:
/home/cyapp/brlcad/doc/docbook/articles/en/ellipsoid.xml validates
[100%] Generating ../../../../share/doc/html/articles/en/ellipsoid.html
[100%] Built target ellipsoid_article_html
~/brlcad/build $