A CMake-Based Cross Platform Build System for Tcl/Tk

October 27, 2011

Approved for public release; distribution is unlimited.

Clifford Yapp
Quantum Research International Inc.
• BRL-CAD - powerful open source solid modeling system developed originally by the Ballistic Research Laboratory (now the U.S. Army Research Laboratory)
  – More than 25 years of development, with ancestor codes dating to the early 1970s.
  – “the world's oldest source code repository” - August 2007, Robin Luckey, Ohloh Inc.
• Since the early days of its development, BRL-CAD has made use of Tcl/Tk
  – Graphical Editing Environments
  – Scripting
  – Interactive command prompt
• Current efforts to upgrade our GUI
  – Ttk widgets
  – HTML-based help system
  – Tktable-based editing tables
  – etc.
Background – Latest BRL-CAD GUI
Building Tcl/Tk - Why?

• Portability
  – Current “primary” platforms are Linux, Mac OS X, Windows, and FreeBSD
  – Historically, BRL-CAD has run on a vast number of operating systems and architectures – portability is key to code longevity
  – BRL-CAD is only as portable as Tcl/Tk

• Required Libraries
  – In order to isolate bugs, it is sometimes necessary to compile against known “good” versions of libraries.
  – Deployment of BRL-CAD often cannot wait on fixes to system libraries.
  – Deploying a version of a system library new enough to support BRL-CAD may break other applications.
  – BRL-CAD has occasionally needed to make changes and fixes to Tcl/Tk

• Bundling
  – To ensure a viable Tcl/Tk is available at all times on all target platforms, a version known to work is bundled with BRL-CAD's own sources.
  – Building the bundled Tcl/Tk requires integration of Tcl/Tk's build system with BRL-CAD's own build logic.
Existing Tcl/Tk Build Systems

- Tcl Extension Architecture (TEA)
  - Autoconf/M4 based
  - Two versions: SC_* for Tcl/Tk and TEA_* for extensions
  - Functionality tests and Tcl/Tk system configuration detection
  - Covers a very wide range of operating systems – some old enough that support for them is no longer needed by BRL-CAD.

- Windows
  - Visual Studio project files – listed version supported is Visual C++ 6.0
  - NMake build files.
  - MinGW/MSYS build files.
  - Cygwin is specifically listed as not supported.

- BRL-CAD Sub-builds
  - MSVC – Custom Visual Studio project files.
  - Autotools – Wrappers for Tcl/Tk's autoconf/SC_* logic.
  - Workable, but sometimes fragile – required tweaking each Tcl/Tk upgrade.
Tcl/Tk as a CMake Sub-build

• Summer 2010 - decision made to unify BRL-CAD's build logic into a single CMake build system
  – Primary goal - simplify building Windows releases.
  – Most BRL-CAD developers work on non-Windows platforms – Windows build files tended to be out of date
  – Windows is a popular deployment platform for BRL-CAD – important to improve the release building process.

• Building Tcl/Tk from CMake
  – 1\textsuperscript{st} attempt - use Tcl/Tk's existing build files and ExternalProject_Add
  – Worked reasonably well on Linux (except for requiring the installation target be built before building the remainder of BRL-CAD.)
  – Visual Studio integration more difficult – initial efforts unsuccessful.

• Building Tcl/Tk \textit{with} CMake
  – Implement enough of Tcl/Tk's build logic in CMake to support BRL-CAD's target platforms.
  – Avoids complexity of OS-dependent external build system triggers, integrates well with BRL-CAD.
To support BRL-CAD's requirements, a CMake build would need to:

- Build successfully on target platforms:
  - Windows
  - Linux
  - FreeBSD
  - Mac OS X
  - Solaris

- Ideally avoid significant alterations to the Tcl/Tk source code

- Run tclsh and wish from the build directory without requiring installation – needed for BRL-CAD's compilation process

- Support compilation of Tcl/Tk extensions
  - Support using either a system Tcl/Tk or BRL-CAD's local copy – both scenarios plausible.
Invoking CMake

- CMake vs. Autotools
  - Out of source directory builds *highly* recommended.
  - Slightly different invocation syntax (see paper for details):
    - `./tcl/unix/configure`
    - `cmake ../tcl`
  - All operating systems use the same toplevel CmakeLists.txt

- CMake vs MSVC/nmake
  - Run CMake to generate a Visual Studio project
  - Launch Visual Studio to complete the build.

Demonstration...
Tcl and Tk are nominally separate projects, with distinct build systems

Despite this separation, Tk requires *internal* Tcl headers when building
- Tk requires the location of a Tcl source repository, as these headers are not guaranteed to be installed
- Tk cannot be compiled against a system Tcl with any guarantee that the internal headers used match those used to build the system Tcl/Tk. Version numbers may match, but that does not preclude local modifications being present in the system Tcl.

Several external Tcl/Tk packages also require the presence of the Tcl/Tk source code.

To support existing code, CMake build logic also must support this source directory inclusion.

Longer term, can internal header use be eliminated?
To support both build directory and install directory pkgIndex.tcl files, CMake creates two and places them appropriately.

- The “final” version intended for install is written to the current binary directory.
- The “in-build-directory version” is written to the appropriate location in CMake's library output directory.
- Tclsh and Wish binaries running from the binary output directory will find the pkgIndex.tcl file in the library output directory.
CMake provides both the ability to read file contents into variables and apply regular expressions to strings.

One or both of these abilities support a number of key features:

- Intelligent placement of library files (code above is a subset of that macro)
- Parsing tclConfig.sh and tkConfig.sh files (FindTCL.cmake)
- Extracting root names from file names
- Breaking up version numbers (major/minor/patch)
• Occasionally, options that are appropriate to show to the user rely on some particular system library (e.g. X11)
• Rather than show an option when the core system feature is not present, CMake provides a Dependent Option macro to conditionally provide options based on other search results.
• This feature is probably useful in more situations – only used right now for a couple of X11 related cases.

```cmake
include(CMakeDependentOption)
CMAKE_DEPENDENT_OPTION(TK_ENABLE_XFT "Use freetype/fontconfig/xft" ON "TK_SYSTEM_GRAPHICS STREQUAL x11;FREETYPE_FOUND;${X11_Xft_FOUND}" OFF)
```
Extensions

• Can be built “stand-alone” but have the same constraints as TEA builds – need Tcl/Tk sources

• Extensions:
  – Tkhtml3
  – Tktable
  – Itcl
  – Itk
  – Togl
Results and Complexity

- Build times are comparable to Autoconf/SC builds on Linux.
  - Benchmark used a single processor build on a Gentoo Linux AMD Athlon II X2 245 CPU.
  - Observed build times within 10% of standard Tcl/Tk build, given the same compiler options.
- Build system complexity is comparable, given implemented logic
  - Hard to objectively measure “complexity” - Lines of Code are in the same ballpark (5-7k) but CMake does not implement all of Tcl/Tk's SC logic.
- As a sub-build within BRL-CAD Tcl/Tk CMake has been consistent and well behaved
  - Builds on all platforms currently supported by BRL-CAD
  - Runs successfully from within the build directory
  - Works with existing Tcl/Tk code – no significant source code modifications
Conclusions and Future Work

- CMake is an effective tool for providing Tcl/Tk with integrated cross-platform build system support.
- BRL-CAD will be maintaining and enhancing this method of Tcl/Tk compilation as part of ongoing development.
- Remaining items to address:
  - Ensure all necessary functionality tests have been ported
  - Implement CPack logic for tarball and binary generation.
  - Address “multi-config” development environments like Xcode
    - Current macros assume a single build directory target for libraries and executables
    - May need to support Debug/Release/etc. configs for proper Xcode integration.
Thank you!