CLHEP Infrastructure Improvements

L. Garren, FNAL, Batavia, IL 60510, USA

Abstract

CLHEP is a set of HEP-specific foundation and utility classes such as random number generators, physics vectors, and particle data tables. Although CLHEP has traditionally been distributed as one large library, the user community has long wanted the ability to build and use CLHEP packages separately.

With the release of CLHEP 1.9, CLHEP has been reorganized and enhanced to enable building and using CLHEP packages individually as well as collectively. The revised build strategy employs all the components of the standard autotools suite: automake, autoconf, and libtool. In combination with the reorganization, the use of these components makes it easy not only to rebuild any single package (*e.g.*, when that package changes), but also to add new packages.

INTRODUCTION

CLHEP [1] contains a number of Hep-specific C++ packages, which include Vector, Random, Matrix, HepPDT, HepMC, and other useful classes. CLHEP is used by Geant4, ThePeg [2], and many individual HEP experiments.

CLHEP has traditionally been distributed as one large software package that builds one large library. However, some packages are updated more frequently than the complete CLHEP release and users may wish to get single package updates. Also, some users want only some of the CLHEP packages.

CHANGES TO CLHEP

As agreed at the 2003 CLHEP Workshop [3], separate builds of each CLHEP package have been enabled. This requires each package to have a configure file, all necessary build scripts, and some knowledge of the other packages it might depend on. Building the entire CLHEP library is still the default option, but the full package build now uses the infrastructure in each package.

Backwards compatibility with CLHEP 1.8.x is required, but only for the first release (1.9.x) of the newly reconfigured CLHEP.

Code Structure

Table 1 shows the changes to the source code structure. CLHEP headers are invoked as "`CLHEP/`*package*`/myheader.h`". Note that this does not match the source code directory structure. The install step creates an `include/CLHEP` header tree under the install directory. Since this tree is not present when building the libraries, we create a temporary header directory during the build process.

Other Changes to CLHEP

C++ namespaces are now in use within all CLHEP packages. If the package had no other defined namespace, the CLHEP namespace was added. When backwards compatibility is enabled, headers will contain relevant "`using namespace`" statements, so that namespace use is transparent to users.

Exception handling packages Exceptions, Cast, and Refcount have been added. These packages are disabled by default.

The Geometry package has been substantially rewritten. Several other packages have been updated.

USING AUTOTOOLS

The CLHEP build scripts and makefiles were almost completely rewritten to take advantage of the power of autotools. The support of various operating system and compiler configurations by the CLHEP maitainer requires recent versions of the autotools suite: autoconf 2.59 [4] or later, automake 1.9.1 [5] or later, and libtool 1.9b [6] or later. However, neither installation nor use of CLHEP requires any of the autotools.

Maintainer bootstrap proceedures invoke aclocal, autoheader, automake, and autoconf. Autoconf creates a configure script from the configure.in instructions and the `aclocal.m4` instructions generated by aclocal. Autoheader creates the template defs.h.in configuration file from instructions found in configure.in. Automake creates template Makefile.in's from very simple Makefile.am's. Libtool is used by the generated makefiles to build both static and shared libraries.

Each package contains various Makefile.am files, a configure.in, and a bootstrap file which runs the autotools. Usually only code developers need to bootstrap. Source code tarballs are available for user installation.

configure.in

The instructions in configure.in provide the basis for configure, the Makefiles, defs.h, and other generated files. This section describes a few interesting autoconf commands.

Table 1: Changes to CLHEP Code Structure

	Old	**New**
*.hh, *.h, *.icc	CLHEP/*package*	CLHEP/*package*/*package*
*.cc	CLHEP/*package*	CLHEP/*package*/src
documentation	CLHEP/*package*/doc	CLHEP/*package*/doc
validation tests	CLHEP/test	CLHEP/*package*/test

AC_INIT(CLHEP Vector,1.9.1.1,CLHEP@cern.ch) declares this as the CLHEP Vector package. The package version is 1.9.1.1, and mail about the package should be sent to CLHEP@cern.ch.

AM_CONFIG_HEADER(Vector/defs.h) tells autoconf that we will create a configuration header named defs.h in the Vector subdirectory. configure.in contains boilerplate for defs.h. Backwards compatibility is enabled if the appropriate lines are included, as illustrated in Table 2.

Files which will be generated from *.in files are specified with the AC_CONFIG_FILES command: AC_CONFIG_FILES([Vector/Makefile]). Makefiles are generated in two steps from Makefile.am files. First, automake generates Makefile.in using Makefile.am and the instructions in configure.in. Then, the configure script generates the final Makefile.

The C++ compiler is chosen from the list in AC_PROG_CXX(cl g++ c++ aCC CC cxx cc++) and AC_LANG(C++) tells configure that C++ is the default compiler. AM_CXXFLAGS is set according to the chosen compiler and operating system, as shown in Table 3. After AM_CXXFLAGS is defined, AC_SUBST(AM_CXXFLAGS) must be called.

Autoconf has tests available for most C++ deficiencies. For instance, AC_CHECK_HEADERS([sstream]) will check for the sstream header and AC_CHECK_TYPES([ptrdiff_t]) will verify that ptrdiff_t is available.

AC_OUTPUT signals the end of configure.in.

Makefiles

Each buildable directory, including the header directory, contains a Makefile.am.

CLHEP/*package*/Makefile.am contains a list of subdirectories (SUBDIRS) to build when "make" is invoked and a list of subdirectories that should be distributed (DIST_SUBDIRS). These lists are often different. Because defs.h must be created and then copied into a temporary header directory tree, the SUBDIRS list for a package is "*package* . src test". Automake recognizes a number of local targets which can be defined by the user. The all-local target builds the temporary header tree by invoking another target in the header subdirectory of packages needed for the build.

CLHEP/*package*/*package*/Makefile.am, in the header directory, contains the rule to copy headers for this package into the temporary header tree. Each header file is individually listed in pkginclude_HEADERS. Note that defs.h must also be mentioned in the pkginclude_HEADERS list.

CLHEP/*package*/src/Makefile.am, is very simple. It contains only three directives. The source code Makefile.am defines INCLUDES (where to find headers), lib_LTLIBRARIES (the library name), and libCLHEP_Vector_@VERSION@_la_SOURCES. Each source code file must be explicitly listed in libCLHEP_Vector_@VERSION@_la_SOURCES.

Autoconf produces a Makefile.in from each Makefile.am, and configure processes the Makefile.in files to generate a Makefile tailored for the specified operating system and compiler.

BUILDING CLHEP

Users should start by downloading and unwinding a CLHEP source code tarball. The build proceedures expect you to create a separate build directory and to identify an installation directory. The default installation directory is /usr/local. Building documents is a separate step since this is not always either feasible or desired.

- cd build-directory
- .../CLHEP/configure
  --prefix=/fullpath/installdir [options]
- make
- make check
- make install
- make docs
- make install-docs

The same steps are used to update or build a single package, with the exception that you call the package's configure script: .../CLHEP/*package*/configure [options].

Useful configure options include:

- CXX=xyz (set the C++ compiler)
- CXXFLAGS="some stuff"
  (append compiler flags to those defined in configure)
- --disable-shared (build only static libraries)
- --enable-exceptions
  (use the Exceptions package)
- --help

It is often necessary to define both the C++ compiler (CXX) and the C compiler (CC). Using environmental variables, *e.g.*, to set CXX, is not recommended.

Table 2: defs.h boilerplate from configure.in

```
AH_TOP([#ifndef VECTOR_DEFS_H
#define VECTOR_DEFS_H])
...
## backwards compatibility
AH_VERBATIM([ENABLE_BACKWARDS_COMPATIBILITY ],
[/* backwards compatibility will be enabled ONLY in CLHEP 1.9 */
#ifndef ENABLE_BACKWARDS_COMPATIBILITY
#define ENABLE_BACKWARDS_COMPATIBILITY
#endif])
AH_BOTTOM([#endif  // VECTOR_DEFS_H])
```

Table 3: defs.h boilerplate from configure.in

```
case "$CXX" in
g++)
   case "$target" in
   *-*-linux*) AM_CXXFLAGS="-O -ansi -pedantic -Wall -D_GNU_SOURCE";;
   *) AM_CXXFLAGS="-O -ansi -pedantic -Wall"
   esac;;
cl) AM_CXXFLAGS="-EHsc";;
*)   echo UNEXPECTED CHOICE OF C++ COMPILER: $CXX
esac
```

CLHEP UTILITIES

CLHEP contains several new utility scripts: `clheplib`, `clhep-config`, and *package*`-config`. `clheplib` will list the library path and library name for use when linking.

- `clheplib`
  `-L/fullpath/lib`
  `-lCLHEP-1.9.1.1`
- `clheplib HepMC`
  `-L/fullpath/lib`
  `-lCLHEP-HepMC-1.9.1.1`
  `-lCLHEP-Geometry-1.9.1.1`
  `-lCLHEP-HepPDT-1.9.1.1`

`clhep-config` and *package*`-config` provide information about the compiler, compiler flags, etc. Use `--help` to view the list of options.

CLHEP RELEASES

There are presently two concurrent releases: CLHEP 1.9.x and CLHEP 2.0.x.

CLHEP 1.9.x is fully backwards compatible. The headers in this release contain relevant "using namespace" statements. This release also contains `CLHEP/config/CLHEP.h` and code to support gcc 2.95.2.

CLHEP 2.0.x is identical to 1.9.x, but without backwards compatibility. There is no `CLHEP/config` package. All configuration information is in the defs.h header found in each package. The defs.h headers are also present in 1.9.x.

Major changes to CLHEP source will appear in CLHEP 2.1.

Supported Compilers

Gcc 3.3 is supported on all platforms. Gcc 3.4 has been tested, but is not yet officially supported. Gcc 2.95.2 is supported only for backwards compatibility. Other actively supported compilers are Solaris CC 5.4, and Windows VisualC++ 7.1. CLHEP must be compiled under cygwin on a Windows machine, but you do not need to invoke cygwin to use the libraries. Instructions for building under cygwin are in CLHEP/ReadMe.cygwin-VC71.

CONCLUSION

The autotools suite (autoconf, autoheader, libtool) is very powerful. A lot of knowledge about compilers and operating systems is embedded in the tools. Libtool makes it very easy to build libraries cleanly and properly.

However, we note that the autotools suite is sometimes too smart. For instance, the generated Makefiles will reconfigure and even attempt to rerun the bootstrap steps if the timestamps on the source files change. This is good for development work, but can be a problem if a user inadvertently changes the timestamps.

Also, there are a lot of assumptions about compiler flags and compile and link commands built directly into libtool. This is usually good, but sometimes causes a problem on a specific compiler and operating system combination. For

instance, versions of libtool prior to 1.9b do not support CC 5.4 on Solaris.

Despite occasional frustration, the autotools suite has been very useful for CLHEP.

ACKNOWLEDGEMENTS

I wish to thank Walter Brown for substantial help with the design of the autotools scripts. In addition, I wish to thank Andreas Pfeiffer for help making CLHEP work with various operating systems and compilers, and for presenting this talk.

REFERENCES

[1] http://savannah.cern.ch/projects/clhep

[2] http://www.thep.lu.se/ThePEG/

[3] http://proj-clhep.web.cern.ch/proj-clhep/Workshop-2003/main.html

[4] http://www.gnu.org/software/autoconf/

[5] http://www.gnu.org/software/automake/

[6] http://www.gnu.org/software/libtool/