

# **Geant4 Installation Guide**

**For setting up Geant4 in  
your computing environment**

***Version: geant4 9.4***

**Publication date 17 December, 2010**

**Geant4 Collaboration**

---

# **Geant4 Installation Guide : For setting up Geant4 in your computing environment**

by Geant4 Collaboration

Version: geant4 9.4

Publication date 17 December, 2010

---

---

# Table of Contents

1. Installation Introduction .....	1
1.1. Supported Computers and Operating Systems .....	1
1.2. Required Software .....	1
1.3. Visualization Software .....	1
1.4. Software for Analysis and Histogramming .....	2
2. Installation Procedures .....	3
2.1. Using the <code>Configure</code> Script for installation .....	3
2.2. Configuring the Environment to Use Geant4 .....	4
2.3. Installing Geant4 Manually .....	4
2.3.1. Required Environment Variables .....	4
2.3.2. Optional Environment Variables .....	4
2.4. Integrating Geant4 into a Generic Framework .....	6
3. Tips for Installing on Windows .....	7
3.1. Windows with the Cygwin Toolset and Microsoft Visual C++ .....	7
3.2. Building Kernel Libraries DLLs .....	9
4. Using CMake to Build and Install Geant4 .....	10
4.1. Prerequisite Software .....	10
4.2. Building and Installing Geant4 with CMake .....	10
4.2.1. Running CMake .....	11
4.2.2. Enabling Optional Components .....	12
4.2.3. Building Geant4 from CMake Generated Buildscripts .....	13
4.3. Using Geant4 .....	13
4.4. Known Issues .....	14

---

# Chapter 1. Installation Introduction

This section describes the global computing environment required for installing the Geant4 toolkit. To set up your specific computing environment for Geant4, refer to Section 2 of this Guide.

A shell script (`Configure`) is provided to facilitate the installation procedure, and to help the user set up the environment correctly.

## 1.1. Supported Computers and Operating Systems

Geant4 is supported under the following operating systems:

- Linux on PC with g++ (gcc compiler)
- MacOSX with g++ (gcc compiler)
- Windows/XP with MicroSoft Visual C++

Currently, this is the set of flavors which can be associated with the environment variable `$G4SYSTEM` to identify the system architecture and compiler used:

Linux	- Scientific Linux CERN, SLC5 g++ gcc 4.1.2	G4SYSTEM: Linux-g++
MacOSX	- MacOSX Darwin 10.6 g++ gcc 4.2.1	G4SYSTEM: Darwin-g++
Windows	- Windows XP and Cygwin32 MSVC++ 9.0, Visual Studio 2008	G4SYSTEM: WIN32-VC

For a more detailed description of supported platforms/compilers and versions of third party software, please refer to the release notes available with the current release you're using in `geant4/ReleaseNotes` (also accessible from the web [distribution page](#) ).

## 1.2. Required Software

To run Geant4, the following software must be properly installed in your computing environment:

- A C++ compiler  
(compiler from your Unix vendor, g++ or Visual C++ for Windows systems)
- The CLHEP library  
(see the CLHEP [reference guide](#) .)
- GNU Make (note: g++ preprocessing is used to build file dependencies) is also used and a UNIX shell
- The Geant4 toolkit source code

## 1.3. Visualization Software

The above list of software is the minimum required for a non-graphical setup of Geant4. To build and use visualization graphics in Geant4 and/or a graphical user interface (GUI), at least one of the following graphic systems or packages is required in your computing environment:

- X Windows.
- OpenGL or MesaGL (free software compatible with OpenGL).
- DAWN , a PostScript renderer.
- Qt , a cross platform application and UI framework (version 4 preferred).
- Open Inventor (free software from SGI).
- Open Scientist (Interactive environment, including GUI).
- HepRApp Browser ( HepRep Browser).

- WIRED4 JAS Plug-In ( HepRep Browser).
- Momo (a Java -based GUI environment, GGE, GPE, ...).
- VRML browser.

Alternatively, you can produce an ascii file for VRML or DAWN.

More information is available in Section 8.6, Visualization Drivers, of the User's Guide for Application Developers.

## 1.4. Software for Analysis and Histogramming

Histogramming facilities are provided through the **AIDA** abstract interface. AIDA is **not** required to build Geant4 itself, but if you wish to use it in your applications, it should be installed following the guide at:

- AIDA (Abstract Interfaces for Data Analysis)

External, AIDA-compliant packages which provide the necessary functionalities for doing histogramming (and therefore should be installed if you require this functionality in your applications) are:

- iAIDA (iAIDA, an implementation of AIDA in C++)
- JAS (Java Analysis Studio)
- Open Scientist (Interactive Analysis Environment).
- rAIDA (rAIDA, a Root implementation of AIDA)

---

## Chapter 2. Installation Procedures

Before installing Geant4, the required software listed in Section 1.2 (and Section 1.3 in the case of graphics drivers) of this Installation Guide must already be installed on your system.

In this section, a short tutorial on how to install the toolkit's kernel libraries is given. The installation of the Geant4 kernel libraries and the proper configuration of the environment can be achieved either manually (by setting the proper environment variables) or through the `Configure` shell script, which will allow the installation of just the necessary source code and libraries in a specified installation area.

*Step-by-Step* guides for the installation are also available. See [Appendix - Step-by-Step Installation Guides of Geant4 User's Guide - For Application Developers](#).

### 2.1. Using the `Configure` Script for installation

A shell script is provided for building the libraries and to allow easy installation in a specified area. The `Configure` shell script is placed in the top directory tree of the distribution (`geant4/Configure`) and allows the user or system administrator to install the Geant4 toolkit in a semi-automatic way. Some knowledge of the system is required for the installation, such as:

- the compiler to be used
- the path where the Geant4 toolkit is to be installed (`$G4INSTALL`)
- definition of installation directory paths (optional)
- kind of graphics/analysis system(s) installed in the system, and paths to the installation of each graphics/analysis package
- the kind of library to be generated: static/shared, compound/granular.

To run the installer script for building the libraries of the Geant4 toolkit, one must type the following from the top directory `geant4`:

```
> ./Configure -build
```

and follow the on-screen instructions. No system administrator privileges are required at this stage. The script will ask for the path in the system where the Geant4 libraries should be installed later on. The script provides default settings for most of the environment variables to be set. By pressing `-RETURN-`, the default values will be selected; otherwise the proper selection (or path, in case a path is requested) must be typed in.

In case the installation procedure fails for some reason, or you realise the selected options were not correct at the time the installation started, you can repeat the whole process by manually removing the current installation with:

```
> cd geant4/source
> make clean
```

where the `$G4SYSTEM` environment variable (specifying the kind of architecture and compiler used) is manually set in your environment (according to the flavors listed in Section 1.1).

In case new modules must be added to an existing build (for example a module for visualization), this can be done manually by re-running `Configure` and providing the new settings.

Once the process of building the libraries has been completed successfully, the Geant4 toolkit can be installed in the specified (already existing) installation area by typing:

```
> ./Configure -install
```

Libraries and necessary source code will be installed in `lib/geant4`, `include/geant4` (if selected to install all headers in a single directory), `src/geant4`, respectively. System administrator privileges may be required for the installation stage, depending upon where in the system the installation should happen.

`$G4INSTALL` will be set to `[INSTALLATION_AREA]/src/geant4`.

## 2.2. Configuring the Environment to Use Geant4

Once libraries have been installed, the user's environment must be correctly set up for the usage of the Geant4 toolkit. The `Configure` script provides a way to check the existing installation and provide the correct configuration for the user's environment. Configuration scripts `env[.sh.csh]` can be generated, and should be sourced by the final users in order to configure their environment according to the performed installation.

To generate the configuration scripts, the user should run `Configure` placed in the installation area, as follows:

```
> $G4INSTALL/Configure
```

This will generate the shell script `env.csh` (`env.sh` for bash shell) to be sourced or integrated into the shell login script (`.tcshrc` or `.bashrc`). The shell script will be generated in the user's current directory (`$PWD`). The user can customize it to specify for example her/his proper working directory through the variable `$G4WORKDIR`. Once the generated script is sourced, the user will be ready to start building a Geant4 application.

Refer to section the section *Getting Started with Geant4 - How to Make an Executable Program* of the Geant4 User's Guide for Application Developers for information on how to build an executable in Geant4.

## 2.3. Installing Geant4 Manually

Before proceeding with the installation, some key environment variables must be defined in your user environment in order to specify where all software components are to be placed and to set some compilation options. A complete reference to all environment variables in Geant4 is available in section *Appendix - Makefiles and Environment Variables* of the Geant4 User's Guide for Application Developers .

### 2.3.1. Required Environment Variables

`G4SYSTEM`:

set to one of the flavors listed in section 1.1 to specify the kind of architecture and compiler used

`G4INSTALL`:

path where the Geant4 toolkit tree is installed (ex. `$HOME/geant4`)

`CLHEP_BASE_DIR`:

path to the CLHEP installation

### 2.3.2. Optional Environment Variables

`G4WORKDIR`:

path of the user's working directory (default in `$G4INSTALL`)

`G4LIB`:

path where the kernel libraries should be installed (default in `$G4INSTALL/lib`)

`G4TMP`:

path where temporary files (object files, dependency files) are placed (default in `$G4WORKDIR/tmp`)

`G4BIN`:

path where final executable files are placed (default in `$G4WORKDIR/bin`).

`G4INCLUDE`:

path where source header files may be mirrored at installation by issuing `make includes` (default in `$G4INSTALL/include`)

**G4DEBUG:**

flag specifying that libraries be built with debug symbols (requires a lot of disk space). The default is optimised-mode

**G4LIB\_BUILD\_SHARED:**

flag specifying that kernel libraries be built as shared libraries (libraries will then be used by default). If not set, static archive libraries are built by default

**G4LIB\_BUILD\_STATIC:**

flag specifying that kernel libraries be built as static archive libraries. Note that you may specify this flag in addition to G4LIB\_BUILD\_SHARED to build shared and static libraries simultaneously.

**G4LIB\_BUILD\_G3TOG4:**

flag specifying that the library for the g3tog4 module be built. By default the library will not be built.

**G4LIB\_BUILD\_ZLIB:**

flag specifying that an additional library for file compression should be built (not required on Linux/Unix systems, required on Windows if choosing OpenGL or OpenInventor visualization). By default the library will not be built.

**G4\_NO\_VERBOSE:**

defining this flag prevents the compilation of verbosity code (for better performance). The default is with verbosity on.

The list of all additional flags (also requiring third-party packages installed) can be found in Section 5.2 of the Geant4 User's Guide for Application Developers .

The Geant4 installation requires native STL (the Standard Template Library) as the base foundation class library. This also implies strict ISO-ANSI language compliance. In addition to the above, you might want to set the proper environment for visualization, such as:

- the kind of graphics driver(s) installed in the system
- the path to the installation of each graphics driver

in case you want to build the Geant4 kernel libraries with the graphics drivers built-in. See *Visualization - The Visualization Drivers* of the Geant4 User's Guide for Application Developers .

At this point, you may choose one of two ways to compile and install the kernel libraries, depending on your needs and system resources. From `$G4INSTALL/source`:

1. `make`

This will make one library for each "leaf" category (maximum library granularity) and automatically produce a map of library use and dependencies.

2. `make global`

This will make global libraries, one for each major category.

The main advantage of the first approach is the speed of building the libraries and of the application, which in some cases can be improved by a factor of two or three compared to the "global library" approach.

Using the "granular library" approach a fairly large number (roughly 90) of "leaf" libraries is produced. However, the dependencies and linking list are evaluated and generated automatically on the fly. The top-level GNUmakefile in `$G4INSTALL/source` parses the dependency files of Geant4 and produces a file `libname.map` in `$G4LIB`. `libname.map` is produced by the tool `liblist`, whose source code is in `$G4INSTALL/config`.

When building a binary application the script `binmake.gmk` in `$G4INSTALL/config` will parse the user's dependency files and use `libname.map` to determine through `liblist` the required libraries to add to the linking list. Only the required libraries will be loaded in the link command.

The command `make libmap` issued from `$G4INSTALL/source`, allows manual rebuilding of the dependency map. The command is issued by default in the normal build process for granular libraries.

It is possible to install both "granular" and "compound" libraries, by typing "make" and "make global" in sequence. In this case, to choose usage of granular libraries at link time one should set the flag `G4LIB_USE_GRANULAR` in the environment; otherwise compound libraries will be adopted by default.

## 2.4. Integrating Geant4 into a Generic Framework

As part of the Geant4 kernel libraries installation, it is also possible to put the entire set of header files in a single place, which is determined by the environment variable `G4INCLUDE` specifying the directory path. Therefore, it's rather straightforward to integrate Geant4 into a generic external framework, by simply knowing the path where header files are located in the system (`G4INCLUDE`) and where installed libraries are placed (`G4LIB`).

In Section 5.2. (*Appendix - Makefiles and Environment Variables*) of the Geant4 User's Guide for Application Developers, you can find a list of all the environment variables. In Section 5.3 it is also explained how to integrate external libraries which may or may not use the Geant4 kernel libraries, using the `GNUmake` build system of Geant4.

---

## Chapter 3. Tips for Installing on Windows

### 3.1. Windows with the Cygwin Toolset and Microsoft Visual C++

To compile and run Geant4 under Windows systems, some additional information and tools are required, although the installation procedure is similar to that required on a UNIX based system. On Windows, the Cygwin toolset and the Microsoft Visual C++ compiler are used.

Cygwin32 is a UNIX development environment available for Microsoft Windows. You can freely obtain Cygwin32 from [Cygwin](#) or [Cygwin/X](#) .

We do not support direct use of Visual Studio; i.e. we do not provide Visual Studio workspace (.dsw) or project (.dsp) files, nor we do provide makefiles for the nmake application of MS Visual C++.

We use several of the tools provided by the Cygwin toolset:

- `make.exe` as a make tool
- `g++.exe` as a tool to analyse source file dependencies and create dependency (.d) files
- several other unix tools like `cp`, `mv`, `rm`, `touch`, etc.

At the installation of the Cygwin toolset it is therefore required to explicitly select some packages (i.e. *gmake*, *gcc*, *binutils* and *tcsh* from the "devel" category) in addition to those which are part of the default installation.

For more details on the toolset, please see the documentation available on the [Cygwin pages](#) . The User's Guide has quick start guides and help with setting up Cygwin with `setup.exe`.

Links to some installation tips for the Cygwin toolset and also "step by step" installation guides can be found in the section [Appendix - Step-by-Step Installation Guides - Build for MS Visual C++](#) of *Geant4 User's Guide - For Application Developers*.

The usage of Cygwin32 for the build environment results in a build procedure similar to that on a UNIX system.

The following steps are required to install geant4:

1. Install Cygwin (see also notes above and additional installation notes for known issues related to the most current version):

We do not depend on any specific feature of the version of Cygwin, so newer versions (or slightly older) are expected to work.

2. Set the environment for MS Visual C++, i.e. set the paths to libraries, include files, and executables for MS Visual C++. This can be done by modifying the start-up batch file `cygwin.bat` of Cygwin32 to include all the MSDOS commands found in the file `vcvars32.bat` provided in MS Visual C++ (in the VC++ .NET compiler installation directory, and usually located inside the `Common7/Tools` directory). This modification should be done after the installation of the Cygwin toolset, so that the environment gets properly set at the time of the invocation of the shell.
3. Unpack the source code into a directory of your choice.
4. Start the Cygwin shell from the start menu.
5. At this point you're ready to install Geant4. If manual installation is chosen, you must set the necessary environment variables. There are various ways to do so; for example through a command file for the Cygwin bash shell. This command file could be named `geant4-setup.sh`, and you must execute it with

```
. geant4-setup.sh
```

including the leading dot and blank space. You could also have this as your `.bashrc` file. The commands in the command file should be:

```
# Set G4SYSTEM
export G4SYSTEM=WIN32-VC
#
# Set Path to CLHEP
export CLHEP_BASE_DIR=C:/usr/local
#
# --- Other optional settings
#Turn on verbose to show command used for compilation
#export CPPVERBOSE=1
#
```

Note, in the example above, CLHEP was installed in C:/usr/local, therefore include/CLHEP and lib/CLHEP.lib must be included therein.

6. Once the environment is correctly set, the libraries are built using make from the geant4/source directory typing make from the Cygwin bash shell prompt.

Examples can be built in the same way as the libraries from examples/novice/N01, for instance: type make from the shell prompt.

Note that, depending on which external software is used, there may be some warnings in linking about conflictings libraries. This often seems to be caused by an external library compiled for a different run time environment.

The binary of the example is placed by default into the geant4/bin/WIN32-VC directory. You may run it either from this directory or from the examples/novice/N01 directory; sample input and output files are placed in each of the examples/novice directories. Some of the examples will need to read data files, and the place has to be given in environment variables again similar to the following example:

```
#
# Environment variables needed to find geant4 data files:
#
# Data for neutron scattering processes,
# distributed in a separate tar file, then placed under data
export G4NEUTRONHPDATA=c:/usr/local/geant4/data/G4NDL
#
# Data for evaluated neutron cross-sections on natural composition of elements,
# distributed in a separate tar file, then placed under data
export G4NEUTRONXSDATA=c:/usr/local/geant4/data/G4NEUTRONXS
#
# Nuclear Photon evaporation data,
# distributed with the source files under data
export G4LEVELGAMMADATA=c:/usr/local/geant4/data/PhotonEvaporation
#
# Data for radio-active decay hadronic processes under data,
# distributed in a separate tar file
export G4RADIOACTIVEDATA=c:/usr/local/geant4/data/RadiativeDecay
#
# Data for low energy electromagnetic processes,
# distributed in a separate tar file, then placed under data
export G4LEDDATA=c:/usr/local/geant4/data/G4EMLow
#
# Data for shell ionisation cross-sections,
# distributed in a separate tar file, then placed under data
export G4PIIDATA=c:/usr/local/geant4/data/G4PII
#
# Data for nuclear shell effects for INCL/ABLA hadronic model,
# distributed in a separate tar file, then placed under data
export G4ABLADATA=c:/usr/local/geant4/data/G4ABLA
#
# Data for measured optical surface reflectance in optical processes,
# distributed in a separate tar file, then placed under data
export G4REALSURFACEDATA=c:/usr/local/geant4/data/RealSurface
#
```

All compiler and linker options are set in config/sys/WIN32-VC.gmk. If you require options different from our choice, you can modify this file.

## 3.2. Building Kernel Libraries DLLs

DLLs (Dynamic Link Libraries) on Windows are supported for MS-VC++ and can be built for the compound kernel libraries of Geant4 (see Section 2.3.2 of the Installation Procedure of this Guide for a dissertation on `global/compound` libraries).

The libraries can be built either manually, issuing the command:

```
make dll
```

from the directory `$G4INSTALL/source` or by specifying it through the `Configure` script used for the installation.

Then, to build any application making use of the installed DLLs, the environment variable `G4LIB_USE_DLL` must be set in the environment.

Once the application is built, it is required to specify to the system the path where the DLLs are installed. To do so, add the absolute path (in Cygwin format) of the DLLs installation directory to the `PATH` variable; for example:

```
export PATH=$PATH:/usr/local/geant4/lib/$G4SYSTEM
```

You may then be able to run successfully your application.

---

# Chapter 4. Using CMake to Build and Install Geant4

In release 9.4, a new *beta* build system for Geant4 using the CMake build system is available. CMake has been chosen as an easy to use and maintain, cross-platform tool that addresses several shortcomings of the existing Configure/Make buildsystem. As a *beta*, not all of the UI and Visualization drivers available in Geant4 and supported by the current Configure/make build system are available. Also, only the following systems are currently supported in this *beta* version:

- Linux on PC with g++ (GNU compiler) and CMake Makefile generation
- MacOSX with g++ (GNU compiler) and CMake Makefile generation

It is expected that the CMake generated buildscripts for Eclipse, KDevelop and Xcode will be functional, but have not been tested yet. Windows builds on XP using the MSVC++ compiler through CygWin have not yet been tested. Native Windows builds using Visual Studio are not yet fullyfunctional. We welcome feedback and problem reports, which should be logged on the Geant4 bug tracking system.

## 4.1. Prerequisite Software

To use the CMake build system, the software listed in Section 1.2 and optionally the Qt4, OpenGL, DAWN and VRML packages listed in Section 1.3 should be installed on your system.

In addition, a recent CMake version, specifically one of

- 2.6.4
- 2.8.\*

should be installed on your system. Binary packages for CMake of the required versions are generally available on most mainline Linux distributions, and should be preferred if available. Otherwise, source and binary packages for Linux, MacOSX and Windows are available from the Kitware download page.

## 4.2. Building and Installing Geant4 with CMake

CMake processes simple input scripts to generate output buildscripts for a native build tool such as makefiles on UNIX. The build process for Geant4 using CMake is thus to run CMake first to generate the native buildscripts and then to run the native tool using these buildscripts as input to build and install.

You should have obtained all the prerequisite software described above, and should have unpacked the Geant4 sources to a location of your choice, for instance

```
/path/to/geant4.9.4
```

We call this directory the *source tree*. The CMake build enforces an *out of source* build so that the buildscripts generated by CMake do not mix with the actual Geant4 sources. This is particularly important for developers using a working copy from the geant4 repository, and also means you can build multiple configurations of Geant4 against one single source tree.

You therefore need to build Geant4 in a separate directory which we call the *build tree*. This directory can be created wherever you like, though we strongly recommend it is outside the source tree. In addition, if you wish to use an IDE such as Eclipse, there may be limitations on the location of the build tree relative to the source tree. For these reasons, we recommend creating your build tree directory alongside the source tree, e.g. using our location from above

```
/path/to/geant4.9.4  
/path/to/geant4.9.4-build
```

You are of course free to experiment here, but the above is the recommended way. You may have as many build trees as you like.

## 4.2.1. Running CMake

CMake generates buildscripts (e.g. Makefiles, Xcode projects) for the build tool (e.g. make, Xcode) of your choice. The first step after setting up the source and build trees is therefore to run CMake in the build tree to generate the needed scripts. We shall assume in the following that we are on \*NIX, where CMake defaults to generating Makefiles, and we shall use the CMake command line interface. CMake has other interfaces such as the Curses based `ccmake`, and if you wish to use these, you should consult the documentation supplied with CMake.

If you wish to generate, e.g. Xcode projects, consult the CMake documentation for the `-G` option which allows you to select another tool. The GUI interface to CMake available on some platforms also has a switch to choose the build tool.

Our first step is to move into the build tree directory and then run CMake, pointing it to the source tree, and setting the `CMAKE_INSTALL_PREFIX` variable to set where `geant4` will be installed. Using our example source and build trees from above, and assuming you want to install Geant4 under `/install/location` (NB `$` denotes the prompt, not a literal '\$')

```
$ cd /path/to/geant4.9.4-build
$ cmake -DCMAKE_INSTALL_PREFIX=/install/location ../geant4.9.4
```

The `cmake` command is the basic command line interface to CMake. All being well, you will see some platform dependent output (not complete)

```
$ cmake -DCMAKE_INSTALL_PREFIX=/install/location ../geant4.9.4
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- setting default compiler flags for CXX
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
```

CMake will then proceed to check for needed external packages. Assuming these are all installed correctly, then it should continue, giving the output (on Linux with GNU compiler)

```
-- Found CLHEP version: CLHEP 2.1.0.1
-- Found CLHEP: TRUE
-- Geant4 backwards compatible system name: Linux
-- Geant4 backwards compatible compiler name: g++
-- Geant4 backwards compatible variable G4SYSTEM : Linux-g++
-- Geant4 backwards compatible variable G4INSTALL: /install/location/share/geant4-9.4.0
-- Geant4 backwards compatible variable G4INCLUDE: /install/location/include/geant4
-- Geant4 backwards compatible variable G4LIB : /install/location/lib/geant4-9.4.0
-- The following Geant4 features are enabled:

-- Configuring done
-- Generating done
-- Build files have been written to: /path/to/geant4.9.4-build
```

Depending on the speed of your computer and/or filesystem, there may be pauses at the `Configuring` and `Generating` stages as CMake actually writes out files.

The detection of CLHEP relies on having the `clhep-config` script in the `PATH`, and if this is not the case on your system, you may see an error

## Using CMake to Build and Install Geant4

---

```
CMake Error at /usr/share/share/cmake-2.8/Modules/FindPackageHandleStandardArgs.cmake:70 (MESSAGE):
  Failed to find CLHEP (missing: CLHEP_VERSION_OK CLHEP_LIBRARIES
  CLHEP_INCLUDE_DIRS)
Call Stack (most recent call first):
  cmake/Modules/FindCLHEP.cmake:118 (find_package_handle_standard_args)
  CMakeLists.txt:114 (find_package)

-- Configuring incomplete, errors occurred!
```

If you see this error, you should either add the location of `clhep-config` to your `PATH` and re-run `cmake` as above, or re-run `cmake` and pass it the full path to `clhep-config` as

```
$ cmake -DCLHEP_CONFIG_EXECUTABLE=/full/path/to/clhep-config ../geant4.9.4
```

You should then see a successful configuration as above.

If you now list the contents of your build directory, it will contain something like

```
$ ls
CMakeCache.txt  cmake_install.cmake  CPackSourceConfig.cmake  outputs
CMakeFiles     CPackConfig.cmake   Makefile                 source
```

Here a Makefile has been created, and if you choose to use one of the other generators, different files will be visible.

### 4.2.2. Enabling Optional Components

The procedure for running CMake described in the preceding section configures a default build of Geant4

- Global dynamic libraries.
- No Geant4 modules requiring external library support.

Extra features can be enabled by passing extra `-D` command line arguments to CMake, as was done to tweak the install prefix above through `CMAKE_INSTALL_PREFIX`. Available options to enable extra features and tweak paths are listed below.

- `BUILD_SHARED_LIBS`: Set to `ON` to build dynamic Geant4 libraries (`ON` by default).
- `BUILD_STATIC_LIBS`: Set to `ON` to build archive Geant4 libraries (`OFF` by default).
- `CMAKE_INSTALL_PREFIX`: Install path prefix (`/usr/local` by default on UNIX).
- `GEANT4_USE_GDML`: Set to `ON` to build Geant4 with GDML support (`OFF` by default, requires Xerces-C to be installed).
- `GEANT4_USE_GEANT3TOGEANT4`: Set to `ON` to build Geant4 with 3-to-4 conversion support (`OFF` by default).
- `GEANT4_USE_QT`: Set to `ON` to build Geant4 with Qt support (`OFF` by default, requires Qt4 with QtCore, QtGui and QtOpenGL, and OpenGL to be installed).

Note that if you enable a `USE` option, extra system checks will be triggered to check for the needed headers and libraries. If you encounter errors at this point, the error message should give you details on the cause.

Further variables are available for accurately locating needed programs and libraries, tuning the install and selecting advanced options.

- `CLHEP_CONFIG_EXECUTABLE`: Full path to `clhep-config` executable (only needed for fine tuning or if `clhep-config` is not in your `PATH`).
- `GEANT4_USE_NETWORKDAWN`: Set to `ON` to build Geant4 DAWN driver with Client/Server support (`OFF` by default, only available on \*NIX systems).
- `GEANT4_USE_NETWORKVRML`: Set to `ON` to build Geant4 VRML driver with Client/Server support (`OFF` by default, only available on \*NIX systems).
- `GEANT4_BUILD_GRANULAR_LIBS`: Set to `ON` to build Geant4 granular libraries (`OFF` by default, if set to `ON` global libraries will not be built).

It is *strongly* recommend to build global libraries. Granular libraries are only intended as a tool for Geant4 developers to cleanly debug and test their code.

### 4.2.3. Building Geant4 from CMake Generated Buildscripts

Once CMake has configured the build, we simply need to run the appropriate build tool with the generated build scripts. In our example we have generated Makefiles, so in our build directory, we simply type

```
$ make
```

to build. All being well, you should see the output

```
$ make
Scanning dependencies of target G4global
[ 1%] Building CXX object source/global/CMakeFiles/G4global.dir/HEPNumerics/src/G4AnalyticalPolSolver.c
[ 1%] Building CXX object source/global/CMakeFiles/G4global.dir/HEPNumerics/src/G4ChebyshevApproximation
...
```

By default, CMake Makefiles produce a summary output of what's being built. If you wish to see more detail, you can run make with

```
$ make VERBOSE=1
```

which will output a very detailed report of everything being done.

CMake Makefiles also support parallel builds, so you can also do

```
$ make -jN
```

where N should be selected based on the number of cores your machine has available. If there are errors in the build, make will immediately exit except in parallel mode where it will exit at the next branch point.

You can also build parts of Geant4 selectively.

```
$ make help
```

will print a list of all targets that can be built.

If the build is successful, you can run

```
$ make install
```

to install Geant4 under the location specified by CMAKE\_INSTALL\_PREFIX from earlier. As noted, this defaults to /usr/local on \*NIX, so you will need to change it as described if you don't have write permission there. Note that you can also do a staged install using DESTDIR as

```
$ make install DESTDIR=/path/to/stage
```

If you are not using Makefiles, you will need to consult the documentation of your buildtool. However, CMake generally works by the book of these tools and so using the generated buildscripts should be straightforward.

## 4.3. Using Geant4

Once Geant4 has been installed two options are available for using the toolkit. The installation provides the existing Geant4 GNU makefile structure under CMAKE\_INSTALL\_PREFIX/share/geant4-9.4.0/config, together with shell scripts

- `CMAKE_INSTALL_PREFIX/share/geant4-9.4.0/config/geant4-9.4.0.sh`
- `CMAKE_INSTALL_PREFIX/share/geant4-9.4.0/config/geant4-9.4.0.csh`

which can be sourced or integrated into the shell rc scripts (`.bashrc` or `.tcshrc`). The scripts will configure your environment in the same way as described in Section 2.2, and Geant4 applications can then be built using the Geant4 supplied GNU makefile system. You should not directly edit the shell scripts to adjust variables such as `G4WORKDIR`. It is better to first source the script and then re-set any needed variables. You will also need to manually set the environment variables pointing to any needed data libraries.

If you do not wish to, or cannot, use the Geant4 supplied GNU make system, a program `CMAKE_INSTALL_PREFIX/bin/geant4-config` is supplied. This can be executed to query the installation and extract needed compiler and linker flags to build an application using Geant4. Its usage is as follows

```
Usage: geant4-config [OPTION...]  
  --prefix                output installation prefix of Geant4  
  --version               output version for Geant4  
  --libs                  output all linker flags  
  --cflags                 output all preprocessor  
                           and compiler flags  
  
  --libs-without-gui      output linker flags without  
                           GUI components  
  --cflags-without-gui    output preprocessor and compiler  
                           flags without GUI components  
  
  --has-feature FEATURE  output yes if FEATURE is supported,  
                           or no if not supported  
  
Known Features:  
  gdml[no], qt[no]  
  
Help options  
  -?, --help              show this help message  
  --usage                 display brief usage message
```

Note that the elements in the `Known Features` section will display `yes` if a feature was enabled in the toolkit build. Note that this program will only be installed if global libraries are built.

## 4.4. Known Issues

- The current inter-library dependency tree is overlinked. Work is underway to resolve this.
- The `geant4-config` script may not work correctly if a full static build is required as it does not currently handle full static library resolution.