Geant4 Installation Guide

Building and Installing Geant4 for Users and Developers

Version: Update to geant4 9.5

Publication date 23rd January, 2012

Geant4 Collaboration

Geant4 Installation Guide : Building and Installing Geant4 for Users and Developers

by Geant4 Collaboration

Version: Update to geant4 9.5

Publication date 23rd January, 2012

Table of Contents

Chapter 1. Getting Started

As of version 9.5.0, Geant4 uses the CMake build system to compile and install the toolkit. CMake has been chosen as an easy to use and maintain, cross-platform tool to replace the old Configure/Make system. This document covers the basics of using CMake to build and install Geant4. For more information on CMake itself, the CMake Help and Documentation should be consulted.

The "manual" installation through environment variables and GNUMake is still available and is documented on a separate chapter.

1.1. Supported and Tested Platforms

Geant4 is officially supported on the following operating system and compiler combinations:

- Linux with gcc 4.1.2 or 4.3, and Intel icc 11 or 12.

  Tested on Scientific Linux CERN 5 (SLC5, based on RedHat Linux Enterprise 5).

  Geant4 has also been successfully compiled on other Linux distributions, including Debian, Ubuntu and openSUSE.

- Mac OS X 10.7(Lion) and 10.6(Snow Leopard) with gcc 4.2.1 (Apple).

- Windows 7 and XP with Visual Studio 9 and 10.

We welcome user feedback and/or bug reports via our HyperNews Forum and Bugzilla.

1.2. Software Required to Build Geant4

The following minimal set of software *must* be present to build Geant4:

- Geant4 Toolkit Source Code.

- CMake 2.6.4 or higher.

  We suggest version 2.8.0 or higher as this provides some additional helpful features, but it is not required.

- C++ Compiler:

  - Linux: GCC(g++).

  - Mac OS X: GCC(g++) (Xcode 3 or 4).

  - Windows: Visual Studio 2009 and 2010, either Express or Higher Editions.

- Linux/Mac only: Make

On Linux, it is strongly recommended that you use CMake as provided through the package management system of your distribution, unless this does not meet the minimum version requirement of CMake 2.6.4. In that case, we recommend you install it using the Linux binary installer for the latest version of CMake, available with instructions from the Kitware download site.

On Mac and Windows, CMake is not installed by default, so we recommend that you install it using the Darwin64 dmg (Mac) or Win32 exe (Windows) installerpackages supplied by the Kitware download site.

On Linux and Mac, it is strongly recommended to use the g++ compiler supplied by the package management system of your distribution (Linux), or supplied with Xcode (Mac).

1.2.1. Changes to Usage of the CLHEP Software

As of version 9.5.0, a minimal version of the CLHEP library is supplied with the Geant4 sources. Geant4 will build and use this internal version of the CLHEP library by default, and so having an external install of CLHEP is no longer a prerequisite.

However, Geant4 can still be configured to use an existing install of CLHEP if required by your use case. This configuration is done by passing extra options to CMake, and if your require this feature you should consult Section 2.3.

1.3. Software Required to Build Optional Components of Geant4

Geant4 has several optional components which if enabled require further software to be preinstalled on your system. These components and their requirements are listed below.

- GDML Support (*All Platforms*)

  Requires: Xerces-C++ headers and library.

- Qt4 User Interface and Visualization (*All Platforms*)

  Requires: Qt4 headers and libraries, OpenGL or MesaGL headers and libraries.

  Mac OS X Lion should use Qt 4.8 or higher, at the time of writing (December 2011) this is available as a release candidate from Nokia.

- Motif User Interface and Visualization (*Linux and Mac OS X*)

  Requires: Motif and X11 headers and libraries, OpenGL or MesaGL headers and libraries.

- X11 OpenGL Visualization (*Linux and Mac OS X*)

  Requires: X11 headers and libraries,OpenGL or MesaGL headers and libraries.

- WIN32 OpenGL Visualization (*Windows*)

  Requires: OpenGL or MesaGL headers and libraries.

- OpenInventor Visualization (*All Platforms*)

  Requires: Implementation of OpenInventor, such as Coin3d with SoXt(SoWin) graphics binding on Linux/ Mac(Windows).

- X11 RayTracer Visualization (*Linux and Mac OS X*)

  Requires: X11 headers and libraries.

On Linux, it is strongly recommend that you use the binary packages as supplied through the package management system of your distribution. If you require a component that uses OpenGL, we also recommend that you install the OpenGL package supplied for your video card (e.g. NVIDIA). You should consult the documentation of your distribution for information on the packages that provide the needed software libraries and headers.

On Mac and Windows, we strongly recommend installing any required packages through binary dmg/exe installers supplied through the vendor links above. Note that Mac OS X already has OpenGL and X11 installed, and Visual Studio supplies an install of OpenGL on Windows. Installation of packages on Mac through MacPorts, fink or homebrew is not fully tested or supported, apart from the cases listed above, but you may build Geant4 using packages supplied by these package management systems with that caveat.

1.4. Software Suggested for Use With Geant4

Geant4 includes many cross-platform, file-based visualization drivers, together with the lightweight inexlib for basic analysis. Geant4 does not require any additional software over and above that listed in Section 1.2 to *build and install* these components.

However, you may wish to install the software suggested below to make use of these components when running your Geant4 application. We again emphasize that you do not need these packages to build and install Geant4.

- DAWN postscript renderer (for use with DAWN visualization driver).
- HepRApp Browser (for use with HepRep visualization driver).
- WIRED4 JAS Plug-In (for use with HepRep visualization driver).
- VRML Browser (for use with VRML visualization driver).
- OpenScientist interactive environment for analysis.
- AIDA implementation such as OpenScientist, iAIDA, JAS3 or rAIDA.

For more details on Geant4's visualization and analysis components, you should consult the relevant sections in the Geant4 User's Guide for Application Developers.

Chapter 2. Building and Installing Geant4

2.1. Building and Installing on Unix Platforms

Unpack the Geant4 source package `geant4.9.5.tar.gz` to a location of your choice. For illustration *only*, this guide will assume it's been unpacked in a directory `/path/to`, so that the Geant4 source package sits in a subdirectory

```
/path/to/geant4.9.5
```

We refer to this directory as the *source directory*. The next step is to create a directory in which to configure and run the build and store the build products. This directory should not be the same as, or inside, the source directory. In this guide, we create this *build directory* alongside our source directory:

```
$ cd /path/to
$ mkdir geant4.9.5-build
$ ls
geant4.9.5  geant4.9.5-build
```

To configure the build, change into the build directory and run CMake:

```
$ cd /path/to/geant4.9.5-build
$ cmake -DCMAKE_INSTALL_PREFIX=/path/to/geant4.9.5-install /path/to/geant4.9.5
```

Here, the CMake Variable `CMAKE_INSTALL_PREFIX` is used to set the *install directory*, the directory under which the Geant4 libraries, headers and support files will be installed. It must be an absolute path. The second argument to CMake is the path to the source directory. In this example, we have used the absolute path to the source directory, but you can also use the relative path from your build directory to your source directory.

Additional arguments may be passed to CMake to activate optional components of Geant4, such as visualization drivers, or tune the build and install parameters. See Section 2.3 for details of these options. If you run CMake and decide afterwards you want to activate additional options, simply rerun CMake passing it those options.

On executing this CMake command, it will run, configuring the build and generating Unix Makefiles to perform the actual build. CMake has the capability to generate buildscripts for other tools, such as Eclipse and Xcode, but please note that *we do not support user installs of Geant4 with these tools*. On Linux, you will see output similar to:

```
$ cmake -DCMAKE_INSTALL_PREFIX=/path/to/geant4.9.5-install /path/to/geant4.9.5
-- 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
-- Found EXPAT: /usr/lib64/libexpat.so
-- The following Geant4 features are enabled:
GEANT4_USE_SYSTEM_EXPAT: Using system install of EXPAT

-- Configuring done
-- Generating done
-- Build files have been written to: /path/to/geant4.9.5-build
```

On Mac OS X, the output will have slight differences, but the last three lines at least should be the same. These indicate a successful configuration.

If you see any errors at this point, carefully check the messages output by CMake, and check your install of CMake and C++ compiler first. The default configuration of Geant4 is very simple, and provided CMake and the compiler are installed correctly, you should not see errors.

After the configuration has run, CMake will have generated Unix Makefiles for building Geant4. To run the build, simply execute make in the build directory:

```
$ make -jN
```

where `N` is the number of parallel jobs you require (e.g. if your machine has a dual core processor, you could set `N` to 2).

The build will now run, and will output information on the progress of the build and current operations. If you need more output to help resolve issues or simply for information, run make as

```
$ make -jN VERBOSE=1
```

Once the build has completed, you can install Geant4 to the directory you specified earlier in `CMAKE_INSTALL_PREFIX` by running

```
$ make install
```

in the build directory. The libraries, headers and resource files are installed under your chosen install prefix in a standard Unix-style hierarchy of directories, described below in Section 3.1.

2.2. Building and Installing on Windows Platforms

Unpack the Geant4 source package, `geant4.9.5.zip` to a location of your choice. For illustration *only*, this guide will assume it's been unpacked in a directory `C:\Users\Ben\Documents\geant4\`, so that the Geant4 source package resides in the subdirectory

```
C:\Users\Ben\Documents\geant4\geant4.9.5
```

We refer to this directory as the *source directory*.

Whilst CMake can be used from the Windows `cmd` program, we recommend using the CMake GUI for configuring the build. The steps required to use the CMake GUI and Visual Studio to build and install Geant4 are described below, and a slideshow of screenshots of the steps is provided.

- Step 1:

  Open the CMake (cmake-gui) executable, and click on the *Browse Source...* button in the top right hand corner of the window. Use the file browser popup to locate the Geant4 source directory, and click *OK*.

- Step 2:

  Now we create a directory in which to create the Visual Studio project files and hold the build products. This directory should not be the same as, or inside, the the source directory. Here we will create this *build directory* alongside our source directory.

  Click on the *Browse Build...* button in the top right hand side of the CMake GUI window. Use the file browser popup to browse back to `C:\Users\Ben\Documents\geant4\`, and click on the *Make New Folder* button. Rename the created folder to *geant4.9.5-build*, and click on the *OK* button. The two text entries at the top of the GUI should now contain `C:/Users/Ben/Documents/geant4/geant4.9.5` and `C:/Users/Ben/Documents/geant4/geant4.9.5-build` respectively (*Note*: CMake always represents Windows paths with forward slashes).

- Step 3:

  Click on the *Configure* button on the bottom left hand side of the GUI.

- Step 4:

  In the popup window, select *Visual Studio 2010* or *Visual Studio 2009* as appropriate for your system. Ensure only the *Use default native compilers* radio box is checked. Then click on the *Finish* button.

- Step 5:

  CMake will then run to check features and provide an initial configuration. All being well, you will see red highlighted entries in the main options window including *CMAKE* and *GEANT4* entries. Note that the red highlighting means CMake has not fully resolved all configuration variables yet, not that has been an error. On Express editions of Visual Studio, you may see a warning about missing system runtime libraries, in the logging window at the bottom of the GUI, but this can be ignored for a standard install. Any other errors will be reported in this logging window.

- Step 6:

  By default, CMake will configure the build to install Geant4 under `C:/Program Files/Geant4`. If you do not wish to use this directory, or do not have permission to install there, you can change the installation location. Click on the arrow next to the the *CMAKE* entry in the central options display to expand all CMake specific configuration options. Scroll down to locate the `CMAKE_INSTALL_PREFIX` entry. Click on the entry and edit the path directly or via the file browser popup. In this example we will modify this *install directory* to `C:/Users/Ben/Documents/geant4/geant4.9.5-install`.

- If you need to activate extra components of Geant4, click on the expansion arrow next to the *GEANT4* entry in the central options display to expand all Geant4 specific configuration options. We will not adjust any here, but they are all documented in Section 2.3. Simply click on the tick box next to an option to select it if you require.

- Step 7:

  Once you have adjusted any options, click on the *Configure* button again. After this has finished, all the options listed in the central option display should be white. If you still have entries in red, click *Configure* to reconfigure until all entries are white.

- Step 8:

  Now click on the *Generate* button to generate the Visual Studio project. All being well, CMake will generate the solution files and finally report *Configuring done, Generating done* in the logging window at the bottom of the GUI. You can now close the CMake GUI.

- Step 9:

  Now start up Visual Studio, and choose *Open Project*. This guide is somewhat specific to Visual Studio 2010 Express, but the solutions should appear with the same names in both 2009 and 2010. Browse to your build directory, and open the Geant4.sln Microsoft Visual Studio Solution file. Note that it may take some time to fully configure, open and parse all files in the solution. You may also use the MSBuild command line utility to build the solution directly if you are familiar with that tool, but we do not as yet document or support this.

- Step 10:

  By default, the Visual Studio solution builds the Debug Configuration, so if you want an optimized build, you should select the *Release* option from the drop down *Solutions Configurations* menu in the toolbar (Visual Studio 2010).

- Step 11:

  In the Solution Explorer, right click on the *INSTALL* solution, and select *Build* from the popup menu. The solution will now build and install to the directory you chose for `CMAKE_INSTALL_PREFIX` earlier. If you

wish to build a different configuration (*Release* if you built *Debug* first, for example), then simply change the solution configuration as described in the previous paragraph, and build the *INSTALL* solution again.

- Step 12:

  If you see a successful build, you can exit Visual Studio.

The file and directory structure of the installation follows that of the Unix build, and is described in Section 3.1.

2.3. Geant4 Build Options

Both Section 2.1 and Section 2.2 give the minimal proceedure to build and install Geant4. Many additional options can be passed to CMake to adjust the way Geant4 is built and installed and to enable optional components.

On the command line, these options may be set by passing their name and value to the cmake command via `-D` flags, for example

```
$ cmake -DCMAKE_INSTALL_PREFIX=/opt/geant4 -DGEANT4_USE_GDML=ON /path/to/geant4.9.5
```

would set up the build of Geant4 for installation under `/opt/geant4` and compilation of support for GDML.

In the CMake GUI, the options are listed as textbox entries, and values may be set directly by clicking on the entry for the option and entering the required information (for example, if a path is required, the GUI will pop up a file browser).

On Unix, CMake also provides a curses based interface, `ccmake`, which can be used to browse and set options in the terminal. Please see the CMake documenation for more information.

If you have already created a build directory and used CMake to configure the build, you can always rerun CMake in that directory with new options to regenerate the buildscripts (Makefiles or IDE solutions). In the CMake GUI, you should set the *Where is the source code:* path to that of your source directory, and the *Where to build the binaries:* path to that of the build directory you wish to reconfigure. You will then need to rebuild and reinstall to pick up the changes. You can also *deactivate* a previously selected option to remove a component from the build. For example, we could do

```
$ cmake -DCMAKE_INSTALL_PREFIX=/opt/geant4 -DGEANT4_USE_GDML=OFF /path/to/geant4.9.5
```

to explicitly remove support for GDML from a build (In the CMake GUI, we would simply uncheck the tick box for `GEANT4_USE_GDML`). Note however that if you reconfigure to *unset* an option and rebuild and reinstall, your install may contain files installed by the previously set option (for example headers).

Options are divided into *Standard* options, which any user or developer can set directly, and *Advanced* options, which in general are only needed by advanced users, developers or to give very fine control over the build and and install. Some options enable components of Geant4 which require external software (as listed in Section 1.2). If these options are enabled, the required software will be searched for, and hence there are also options which control where CMake should look for these packages. If a required software package is not found, then CMake will exit with an error message detailing what was not found.

2.3.1. Standard Options

We list standard options here in logical order. If you use CMake's curses or GUI interfaces, they will be listed alphabetically.

- `CMAKE_INSTALL_PREFIX`
  - Sets the installation prefix for Geant4. Equivalent to `--prefix` in Autotools. Its default is platform dependent:

Unix: `/usr/local`

Windows: `C:\Program Files\Geant4`

It should be supplied as an absolute path, otherwise CMake will interpret its value relative to your build directory.

See also the `CMAKE_INSTALL_XXXDIR` Advanced Options for fine control of installation locations.

- `CMAKE_BUILD_TYPE : (DEFAULT : Release)`
  - Controls the type of build, at present only the additional flags passed to the compiler. It defaults to `Release` which gives an optimized build with no debugging symbols. The most useful values are:

    `Release` : Optimized build, no debugging symbols

    `Debug` : Debugging symbols, no optimization

    `RelWithDebInfo` : Optimized build with debugging symbols

    Note that if you use a build system which supports multiconfiguration builds (e.g. Xcode, Visual Studio), this variable has no effect as all build types are available inside the CMake generated project.
- `GEANT4_INSTALL_DATA : (DEFAULT : OFF)`
  - If set to `ON`, download and install all Geant4 data libraries (i.e. G4NDL etc). Each library will be unpacked and installed in a directory named `share/Geant4-9.5.0/data` under `CMAKE_INSTALL_PREFIX`.

    `REQUIRES :` CMake 2.8 or higher and a working network connection. It is highly recommended to switch this option on if you have the requisite CMake version and network connection to give the best integration with application development.
- `GEANT4_USE_GDML : (DEFAULT : OFF | ON if XERCESC_ROOT_DIR is set)`
  - If set to `ON`, build the `G4persistency` library with support for GDML.

    `REQUIRES :` Xerces-C++ libraries and headers, see the `XERCESC_ROOT_DIR` option.
- `XERCESC_ROOT_DIR`
  - If your Xerces-C++ installation is in a non-standard location, set this variable to the root directory of the installation (i.e. the directory containing the include and lib subdirectories for Xerces-C++). If this is not sufficient to locate Xerces-C++, see the Advanced `XERCESC_INCLUDE_DIR` and `XERCESC_LIBRARY` options.
- `GEANT4_USE_QT (DEFAULT : OFF)`
  - If set to `ON`, build Qt4 User Interface and Visualization drivers.

    `REQUIRES :` Qt4 and OpenGL libraries and headers. See the `QT_QMAKE_EXECUTABLE` option.
- `QT_QMAKE_EXECUTABLE`
  - If your Qt4 installation is in a non-standard location, set this variable to point to the `qmake` executable of the Qt4 installation you wish to use. If you have a system install on Linux or the binary SDK install from Nokia on other platforms, Qt4 will in general be found automatically (CMake should also honor the `QTDIR` environment variable).
- `GEANT4_USE_XM (DEFAULT : OFF, Unix Only)`
  - If set to `ON`, build Motif User Interface and and Visualization drivers.

REQUIRES : Motif and OpenGL libraries and headers. In most cases, these should be found automatically, but if not, see the Advanced MOTIF_INCLUDE_DIR and MOTIF_LIBRARIES options.

- GEANT4_USE_OPENGL_X11 (DEFAULT : OFF, Unix Only)
  - If set to ON, build the X11 OpenGL visualization driver.

    REQUIRES : X11 and OpenGL libraries and headers.
- GEANT4_USE_OPENGL_WIN32 (DEFAULT : OFF, Windows Only)
  - If set to ON, build the X11 OpenGL visualization driver.

    REQUIRES : OpenGL libraries and headers.
- GEANT4_USE_INVENTOR (DEFAULT : OFF)
  - If set to ON, build the OpenInventor visualization driver.

    REQUIRES : OpenInventor implementation (Coin3d recommended), Xt (Unix) or Win (Windows) binding, and OpenGL libraries and headers. CMake will use coin-config and soxt-config if present to locate the Coin3d implementation, and will honor the COINDIR environment variable. In case of issues with locating the Inventor implementation, see the Advanced INVENTOR_INCLUDE_DIR, INVENTOR_LIBRARY_RELEASE, INVENTOR_SOXT_LIBRARY and INVENTOR_SOWIN_LIBRARY options.
- GEANT4_USE_RAYTRACER_X11
  - If set to ON, build RayTracer visualization with X11 support.

    REQUIRES : X11 Headers and Libraries.
- GEANT4_USE_SYSTEM_CLHEP (DEFAULT : OFF | ON if CLHEP_ROOT_DIR set)
  - If set to ON, build Geant4 with an external install of CLHEP. You *should not* set this unless your usage of Geant4 mandates a specific external CLHEP installation (e.g. if your experiment's software uses CLHEP in other tools and requires consistent use of the same CLHEP across the software). If the CLHEP_ROOT_DIR option is not set, CLHEP will be searched for under standard system paths.

    REQUIRES : CLHEP libraries and headers.
- CLHEP_ROOT_DIR
  - If you wish GEANT4 to use a specific installation of CLHEP, set this variable to point to the root install directory of the CLHEP installation you wish to use. This directory should contain the include and lib subdirectories containing the CLHEP headers and libraries respectively. If this is not sufficient to locate CLHEP, see the Advanced CLHEP_INCLUDE_DIR and CLHEP_LIBRARY options.

2.3.2. Advanced Options

Most builds should never need to touch the Advanced options, but should you need more control or CMake has problems locating needed software packages, they can be very helpful. We only list the options most relevent for Geant4.

Advanced options are not displayed by default in CMake's curses and GUI interfaces, but can be displayed by pressing t in ccmake, or clicking the 'advanced' check box in the CMake GUI. Note that displaying advanced options will also display a large number of options used by CMake and to record the locations of Third Party packages. On the command line, advanced options can be set like any other option.

- GEANT4_USE_NETWORKDAWN : (DEFAULT : OFF, Unix Only)

  - If set to ON, build network server/client support for DAWN visualization driver. You do *not* need this to view DAWN files.

- GEANT4_USE_NETWORKVRML : (DEFAULT : OFF, Unix Only)
  - If set to ON, build network server/client support for VRML visualization driver. You do *not* need this to view VRML files.

- GEANT4_INSTALL_EXAMPLES : (DEFAULT : OFF)
  - If set to ON, install Geant4 example application source code under CMAKE_INSTALL_DATAROOTDIR.

- BUILD_SHARED_LIBS : (DEFAULT : ON)
  - If set to ON build Geant4 shared libraries.

- BUILD_STATIC_LIBS : (DEFAULT : OFF)
  - If set to ON, build Geant4 static libraries.

- CMAKE_INSTALL_BINDIR : (DEFAULT : bin)
  - Installation directory for Geant4 executables. It can be supplied as a path relative to CMAKE_INSTALL_PREFIX or as an absolute path.

- CMAKE_INSTALL_INCLUDEDIR : (DEFAULT : include)
  - Installation directory for Geant4 C/C++ headers. It can be supplied as a path relative to CMAKE_INSTALL_PREFIX or as an absolute path.

- CMAKE_INSTALL_LIBDIR : (DEFAULT : lib(+?SUFFIX))
  - Installation directory for object code libraries It can be supplied as a path relative to CMAKE_INSTALL_PREFIX, or an absolute path. By default, SUFFIX will be set to 64 on 64bit Linux platforms apart from Debian systems.

- CMAKE_INSTALL_DATAROOTDIR : (DEFAULT : share)
  - Installation directory for read-only architecture-independent data files. It can be supplied as a path relative to CMAKE_INSTALL_PREFIX, or an absolute path.

- XERCESC_INCLUDE_DIR
  - If CMake cannot locate your Xerces-C++ installation, set this to the directory containing the Xerces-C ++ headers (e.g. if you have /foobar/xercesc/util/XercesVersion.hpp, then set this to /foobar).

- XERCESC_LIBRARY
  - If CMake cannot locate your Xerces-C++ installation, set this to the full path to the Xerces-C++ library, e.g. /usr/lib/libxerces-c.so

- MOTIF_INCLUDE_DIR
  - If CMake cannot locate your Motif installation, set this to the directory containing the Motif headers (e.g. if you have /foobar/Xm/Xm.h, then set this to /foobar).

- MOTIF_LIBRARIES
  - If CMake cannot locate your Motif installation, set this to the full path to the Motif library, e.g. /usr/lib/libXm.so

- `INVENTOR_INCLUDE_DIR`
  - If CMake cannot locate your OpenInventor installation, set this to the directory containing the Inventor headers (e.g. if you have `/foobar/Inventor/So.h`, then set this to `/foobar`).
- `INVENTOR_LIBRARY_RELEASE`
  - If CMake cannot locate your Inventor installation, set this to the full path to the Inventor Release library, e.g. `/usr/lib/libCoin.so`
- `INVENTOR_SOWIN_LIBRARY (`*`Windows only`*`)`
  - If CMake cannot locate your Inventor installation, set this to the full path to the Inventor SoWin binding library, e.g. `c:\Program Files\Coin\sowin.dll`.
- `INVENTOR_SOXT_LIBRARY (`*`Unix only`*`)`
  - If CMake cannot locate your Inventor installation, set this to the full path to the Inventor SoXt binding library, e.g. `/usr/lib/libSoXt.so`.
- `CLHEP_INCLUDE_DIR (`*`If GEANT4_USE_SYSTEM_CLHEP selected`*`)`
  - If CMake cannot locate your external CLHEP installation, set this to the directory containing the CLHEP headers (e.g. if you have `/foobar/CLHEP/Vector/defs.h`, then set this to `/foobar`).
- `CLHEP_LIBRARY (`*`If GEANT4_USE_SYSTEM_CLHEP selected`*`)`
  - If CMake cannot locate your CLHEP installation, set this to the full path to the CLHEP library, e.g. `/usr/lib/libCLHEP.so`
- `GEANT4_BUILD_STORE_TRAJECTORY : (DEFAULT : ON)`
  - If set to `ON`, store trajectories in event processing. It can be switched to `OFF` to give a degree of performance improvement, but you will *not* be able to visualize events.
- `GEANT4_BUILD_VERBOSE_CODE : (DEFAULT : ON)`
  - If set to `ON`, build Geant4 libraries with extra verbosity. It can be switched to `OFF` to give a degree of performance improvement, but you will not have as mush information output should you run into problems or need to debug.
- `GEANT4_BUILD_GRANULAR_LIBRARIES : (DEFAULT : OFF)`
  - If set to `ON`, build Geant4 libraries in granular mode. *WARNING:* this option is for Geant4 developers only and does not provide a fully usable Geant4 install. *No support is, or will be, provided for user applications built with granular libraries.*
- `GEANT4_BUILD_EXAMPLES : (DEFAULT : OFF)`
  - If set to `ON`, build all Geant4 example applications using current Geant4 build. *WARNING:* this option is for Geant4 system testing only and is not intended for use by users studying and working on the examples. *No support is, or will be, provided for user builds of the examples using this option.*
- `GEANT4_ENABLE_TESTING : (DEFAULT : OFF)`
  - If set to `ON`, build and run Geant4 testing suites. *WARNING:* this option is for Geant4 system testing only and is not intended for use by users. *No support is, or will be, provided for user builds with this option.*

Chapter 3. Setting Up and Using an Install of Geant4

3.1. Geant4 Installation Locations

If you choose the default installation paths, then your install of Geant4 is completely contained under the directory you chose for CMAKE_INSTALL_PATH, with the structure

```
+- CMAKE_INSTALL_PREFIX
   +- bin/
   |  +- geant4-config   (UNIX ONLY)
   |  +- geant4.csh      (UNIX ONLY)
   |  +- geant4.sh       (UNIX ONLY)
   |  +- G4global.dll    (WINDOWS ONLY)
   |  +- ...
   +- include/
   |  +- Geant4/
   |     +- G4global.hh
   |     +- ...
   |     +- CLHEP/       (WITH INTERNAL CLHEP ONLY)
   |     +- tools/
   +- lib/               (MAY BE lib64 on LINUX)
   |  +- libG4global.so  (AND/OR .a, OR G4Global.lib ON WINDOWS)
   |  +- ...
   |  +- Geant4-9.5.0/
   |     +- Geant4Config.cmake
   |     +- Geant4ConfigVersion.cmake
   |     +- Geant4LibraryDepends.cmake
   |     +- Geant4LibraryDepends-Release.cmake
   |     +- UseGeant4.cmake
   |     +- Linux-g++    (OR Darwin-g++ UNIX ONLY SOFTLINK -> ..)
   +- share
      +- Geant4-9.5.0
         +- data/        (IF GEANT4_INSTALL_DATA WAS SET)
         +- geant4make/
            +- geant4make.csh
            +- geant4make.sh
            +- config/
```

To make the Geant4 binaries and libraries available on your PATH and library path (LD_LIBRARY_PATH on Linux, DYLD_LIBRARY_PATH on Mac OS X), you should source the relevant script in the CMAKE_INSTALL_PREFIX

On bourne shells (e.g. bash), do (assuming you are in CMAKE_INSTALL_PREFIX):

```
$ . bin/geant4.sh
```

On C shells, do (assuming you are in CMAKE_INSTALL_PREFIX):

```
$ source bin/geant4.csh
```

On Windows, you should add the directory containing the Geant4 dll files to your PATH environment variable. On Windows 7/XP, this can be done via the Control Panel as follows

- Step 1:

  Open the Windows Control Panel.

- Step 2:

  Open the *System* item in the Control Panel.

- Step 3:

  Click on the *Advanced system settings* link on the System window (on *Windows XP*, click on the *Advanced* tab).

- Step 4:

  Click on the *Environment Variables* button in the System Properties window.

- Step 5:

  Select the PATH entry in the *User variables* list, and click the *Edit* button. If PATH is not present, click on the *New* and create it.

- Step 6:

  In the popup *Edit User Variable* window, append the directory in which the Geant4 dlls are installed to the Variable value entry of the PATH variable (Note that on Windows, path entries are separated by semicolons). It's very important to append the Geant4 dll path if you have an existing PATH, otherwise other programs may stop working correctly! If the Variable value entry of the PATH variable is empty, or you've just created it, you can simply set the value to the directory in which the Geant4 dlls are installed. Once you have edited, click *OK*.

3.2. Building Applications with Geant4

Geant4 comes with three options for configuring your environment and application build to use the installed headers and libraries.

3.2.1. Geant4Make

Geant4Make is the Geant4 GNU Make toolchain formerly used to build the toolkit and applications. It is installed for backwards compatibility with the Geant4 Examples and your existing applications which use a GNUmakefile and the Geant4Make `binmake.gmk` file.

Geant4Make is configured by environment variables, so to use this toolchain to build applications you need to source the setup script `CMAKE_INSTALL_PREFIX/share/Geant4-9.5.0/geant4make/geant4make.(c)sh` as appropriate for your shell.

The usage of Geant4Make and syntax of the needed `GNUMakefile` is described in the FAQ and Appendices of the Geant4 User's Guide for Application Developers.

Please note that this toolchain is provided purely for conveniance and backwards compatibility. We encourage you to use, or migrate your applications to, the new CMake and geant4-config tools described below.

3.2.2. Geant4 CMake Module

Geant4 provides a CMake module, Geant4Config.cmake, located in

```
+- CMAKE_INSTALL_PREFIX
    +- lib/
       +- Geant4-9.5.0/
          +- Geant4Config.cmake
```

for use by CMake's find_package command.

This allows you to use CMake to easily build your own Geant4 application, and can be used on all platforms transparently. To do this, your CMake script needs to call `find_package`

```
find_package(Geant4)
```

and you need to point CMake to the location of the `Geant4Config.cmake` file via the CMake option `Geant4_DIR`.

The usage of CMake to build a Geant4 application using this module is described in more detail in the Appendices of the Geant4 User's Guide for Application Developers. We welcome feedback, suggestions for improvement and bug reports on `Geant4Config.cmake`.

3.2.3. Unix `geant4-config` Shell Script

On Unix platforms only, a simple config script is provided in

```
+- CMAKE_INSTALL_PREFIX
    +- bin/
       +- geant4-config
```

This script may be run and passed arguments to obtain the installation location of Geant4, the compile and link flags needed to build an application against Geant4, and the features the install of Geant4 provides. You can pass the `--help` argument to obtain the usage of the script:

```
$ geant4-config --help
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:
 clhep[yes]
 expat[no]
 gdml[no]
 qt[no]
 motif[no]
 raytracer-x11[no]
 opengl-x11[no]
 openinventor[no]

Help options:
  -?, --help              show this help message
  --usage                 display brief usage message
```

If you are on Unix and do not wish to use CMake to build applications, we recommend the use of `geant4-config`. We welcome feedback, suggestions for improvement and bug reports on `geant4-config`.

3.3. Note on Geant4 Datasets

If you built and installed Geant4 configured with the option `GEANT4_INSTALL_DATA` set, then the Geant4 dataset libraries will also be downloaded and installed.

In this case, the `geant4.(c)sh` and `geant4make.(c)sh` scripts will set up the needed environment variables required by Geant4 to locate these datasets. On Windows, you will need to set the needed environment variables listed below by hand. This can be done by opening the *Environment Variables* dialog, as described in Section 3.1, clicking on the *New* button, and entering the *Variable name* and *Variable value* using the names and paths described below.

If you choose not to install the datasets, but require them later, you will need to download and unpack them by hand to a location of your choice and set the needed environment variables:

- `G4ABLADATA`

  Set to the path to the G4ABLA3.0 dataset.

- `G4LEDATA`

  Set to the path to the G4EMLOW6.23 dataset.

- `G4LEVELGAMMADATA`

  Set to the path to the PhotonEvaporation2.2 dataset.

- `G4NEUTRONHPDATA`

  Set to the path to the G4NDL4.0 dataset.

- `G4NEUTRONXSDATA`

  Set to the path to the G4NEUTRONXS1.1 dataset.

- `G4PIIDATA`

  Set to the path to the G4PII1.3 dataset.

- `G4RADIOACTIVEDATA`

  Set to the path to the RadioactiveDecay3.4 dataset.

- `G4REALSURFACEDATA`

  Set to the path to the RealSurface1.0 dataset.

Chapter 4. CMake and Build Tools For Geant4 Developers

Geant4 developers can make use of several powerful features of CMake to help with their work. The key concept and working practice is the separation of the *source directory*, which is where the sources you edit reside, and the *build directory*, where the buildscripts and compiled products reside.

4.1. Developing Geant4 using Make, Xcode, Visual Studio and Eclipse

CMake is a buildsystem that generates scripts for buildtools including Make, Xcode, Visual Studio and Eclipse, among others. The resulting scripts can then be used within the build tool to perform the actual build, install and packaging.

Whilst we only support Make and Visual Studio for Unix and Windows user builds respectively, Geant4 developers are welcome, and encouraged, to use their build tool of choice. Buildscripts for Geant4 using these other buildtools can be generated by choosing the CMake generator when running CMake for the first time.

On the command line, one can select the tool using the `-G` argument of CMake. For example, to generate an Xcode project for Geant4 using the example from Section 2.1:

```
$ mkdir -p /path/to/geant4.9.5-build-xcode
$ cd /path/to/geant4.9.5-build-xcode
$ cmake -G Xcode -DCMAKE_INSTALL_PREFIX=/path/to/geant4.9.5-install /path/to/geant4.9.5
```

The resulting `/path/to/geant4.9.5-build-xcode/Geant4.xcodeproj` project may be opened with Xcode.

In the CMake GUI, the generator will be asked for the first time you click on *Configure* button (see Section 2.2), where it can be selected from a drop down list.

Note that in all cases, you can only have one buildtool configuration in a given build directory (e.g. you cannot have Unix Makefiles and an Xcode project).

Support for these buildtools is still preliminary, so feedback is welcome, whether bug reports, guides or general comments.

4.2. Command Line Help with Make

If you develop using the command line and Make, you can get information on the targets available to make by making the `help` target in your build directory:

```
$ make help
The following are some of the valid targets for this Makefile:
... all (the default if no target is provided)
... clean
... depend
.furthertargets.
```

You may build any target individually, and it will be built together with all of its dependencies. CMake's generated makefiles only output minimal information, but if you need to see the full commands for debugging or just info, you can run make as

```
$ make VERBOSE=1
```

to output all commands.

If you want to very quickly check that your target compiles, *without* checking any of its dependencies, you can append `/fast` to the target name, e.g.

```
$ make G4run/fast
```

This will finish with an error if any dependents of the target do not exist, but can be useful to quickly check that just the sources you are working on compile.

4.3. Building Quickly and Efficiently with Multiple Build Directories

The many ways in which Geant4 can be configured with optional components can make compilation and testing under different configurations time consuming.

As the CMake configured buildscripts live in a *build directory* isolated from the *source directory*, one can create several build directories configured against the same source directory. Each directory can have a different configuration, for example

```
$ cd /path/to
$ ls
geant4.9.5
$ mkdir geant4.9.5-build-release geant4.9.5-build-debug
$ cd geant4.9.5-build-release
$ cmake -DCMAKE_BUILD_TYPE=Release ../geant4.9.5

...output...

$ make -j8

...output...

$ cd ../geant4-build-debug
$ cmake -DCMAKE_BUILD_TYPE=Debug ../geant4.9.5

...output...

$ make -j8

...output...
```

The above example uses Unix, but the same technique works on all platforms. It may not seem to have gained you much, but when you now edit and develop code that is living under `/path/to/geant4.9.5`, you only need to rebuild in each directory:

```
... work on code ...
$ cd /path/to/geant4.9.5-build-release
$ make -j8

... incremental build ...

$ cd /path/geant4.9.5-build-debug
$ make -j8

... incremental build ...
```

The builds pick up the changes you make to the source and build separately *without needing reconfiguration*. This is particularly powerful if the different configurations you need to test (for example, different versions of an external package) would require significant recompilation if the configuration were changed. Naturally, this power comes at the cost of some disk space, so may not be ideal in all cases.

Note that whilst this technique works on all platforms and buildtools, if you are using an IDE like Xcode or Visual Studio and simply want to build in the Release, Debug and other build modes, the IDE will handle this for you. These tools essentially use the above pattern behind the scenes.

4.4. Building Test Applications Against Your Development Build

A key feature of developing Geant4 with CMake is that you *do not* need to install your current build to be able to use it. A typical use case here is that you have a simple application which you want to build against your current development build of Geant4 for testing.

Versions of the `geant4cmake.(c)sh` (described in Section 3.2.1), `Geant4Config.cmake` (described in Section 3.2.2) and `geant4-config` (described in Section 3.2.3) scripts are created in the build directory. Each of these are configured to use the libraries as they exist in the build directory, and headers from the source directory, without installation.

You can therefore use these scripts as described earlier in Chapter 3 to build your test applications *against a specific build tree*. You therefore don't need to install Geant4 everytime you make a small update.

Chapter 5. Help And Support

5.1. Getting Help

Whilst every effort has been made to make the build of Geant4 robust and reliable, the multitude of platforms and system configurations mean we cannot guarantee that problems will not be encountered on platforms other than those listed in Section 1.1.

In case of issues with building and installing Geant4, we welcome questions as well as feedback via our HyperNews Forum. To help us deal with your problem as quickly as possible, please include as much detail as possible. At minimum, you should let us know the platform and operating system version, your C++ compiler type and version, CMake version and any error messages. It also helps to have the sequence of commands used to reproduce the issue.

Please note that as discussed earlier we can only support user installs on Unix via CMake and Unix Makefiles, and on Windows via CMake and Visual Studio. Developers however are welcome to try CMake and other buildtools like CMake and Eclipse, and we welcome your feedback here.

If you feel you have found a genuine bug in the Geant4 CMake build, please report in to the CMake category on our Bugzilla. We also welcome feature requests and feedback.

5.2. Further Information

- CMake Documentation
- CMake Wiki
- CMake Tutorial

Chapter 6. Manual GNUmake Installation Procedures on Unix

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.

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 by means of the `CMake` system.

In this section, a short tutorial on how to -manually- install the toolkit's kernel libraries is given.

6.1. 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 .

6.1.1. Required Environment Variables

`G4SYSTEM:`
set to one of the flavors listed below:

```
Linux   - Scientific Linux CERN, SLC5
          g++ gcc 4.1.2                          G4SYSTEM: Linux-g++
MacOSX  - MacOSX Darwin 10.7
          g++ gcc 4.2.1                          G4SYSTEM: Darwin-g++
Windows - Windows 7 and Cygwin32
          MSVC++ 10.0, Visual Studio 2010        G4SYSTEM: WIN32-VC
```

`G4INSTALL:`
path where the Geant4 toolkit tree is installed (ex. `$HOME/geant4`)

`CLHEP_BASE_DIR:`
path to the CLHEP installation

6.1.2. 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 global" and "make" 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.

6.2. 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 7. Tips for manually Installing on Windows with GNUMake

7.1. Windows with the Cygwin Toolset and Microsoft Visual C++

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

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

Several tools provided by the Cygwin toolset are used:

- `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 explicitely 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 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 with GNUMake on Windows:

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, an external CLHEP installation is used and 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 G4LEDATA=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.

7.2. Building Kernel Libraries DLLs with GNUMake

DLLs (Dynamic Link Libraries) on Windows are supported for MS-VC++ and can be built for the `compound` kernel libraries of Geant4 (see Section 6.1.2 of the Installation Procedure of this Guide for a dissertation on `global/compound` libraries).

The libraries can be built issuing the command:

```
make dll
```

from the directory `$G4INSTALL/source`.

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.