Geant4 CMake: 10.2 to 10.3

Ben Morgan

THE UNIVERSITY OF
WARWICK

Core fixes/features since 10.1

- CMake version bump -> 2.8.12 (-> *3.3 for C++11*)
- Windows builds of DLL/Archives improved
- Default use of `@rpath` in OS X install names
- "Granular" mode of system CLHEP can be used if requested
- G4ENSDFSTATEDATA exported to environment by default

- Version bump mostly towards C++11 and modularisation (see later)

- Windows clients can now build DLLs/Archives together, and issues with some aspects of symbol visibility resolved.

- See the Kitware blogpost on @rpath for a good overview of why this is a better system than full, or no, install names.
  - http://www.kitware.com/blog/home/post/510

- "Granular" mode of CLHEP was requested by ATLAS, though is not intended for general use (handling N different library structures is hard from a configuration/build perspective).

- Brings us to 10 data libraries, and consequently 10 different environment variables…See separate slide on discussing data libraries later

Improved Build Product Layout

- When building Geant4, libraries etc are output to same directory structure as they will be installed in:

```
+- YourBuildDir/
   +- <buildscripts>
   +- BuildProducts/
      +- <MODE>/
         +- bin/
         +- lib/
            +- libG4global
```

Present for Xcode, VS etc

- <MODE> layer only present if build tool supports multiple build modes (Release, Debug etc).

- Subdirectories match the paths given to CMake install dir variables like CMAKE_INSTALL_{BINDIR,LIBDIR} etc.

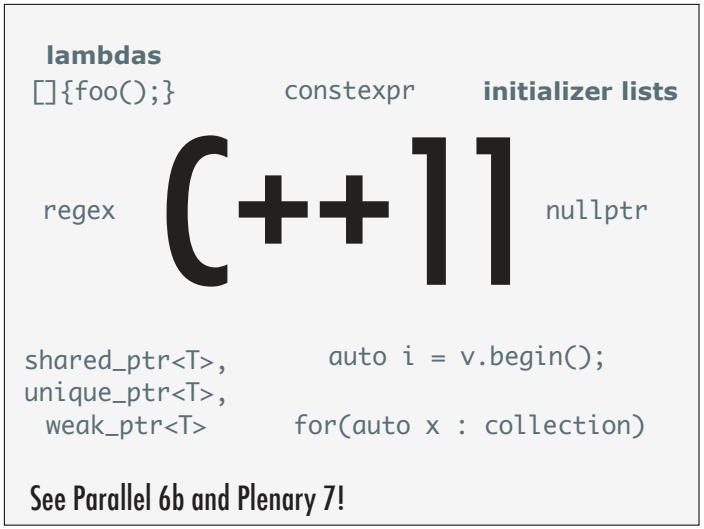
- Cleaner layout with better support for multi-mode IDEs like Xcode.

Install Guide Updates

- `cmd/Powershell` guide for Windows to be added
  - *Addresses Bugzilla #1729 and makes install process of the toolkit a bit easier and close to identical to UNIX.*
- Will retain Visual Studio GUI guide for user applications
- Will not provide guides for MinGW or Cygwin

- MinGW and Cygwin should, in use of CMake terms, behave much like the UNIX command line builds.

- Moreover, we don't have the resource to test and support these.



- On the CMake side, this addresses checking the compiler can support language features, and the standard library implements needed functions/objects.

- Parallel 6b will provide the overview on supported compilers, how to set these up and provide a demo of how CMake C++ standard support works

- Suffice to say, things are in good shape.

Modularization: G4processes

- **Reached symbol table limit on Windows VS2015**
  - See, e.g., http://cdash.cern.ch/viewBuildError.php?buildid=163810
- **Plan, for discussion, to break into three libs for 10.2:**
  - "G4processes-{hadronic,electromagnetic,general}"
- *No changes needed on your or users part, CMake will transform any compile/link paths/dependencies as needed*

- Different schemes for splitting G4processes can be considered, but the above is the simplest and minimally invasive.

- Key point is that you do not see this change. If you link to "G4processes" you still do so. The CMake system will be updated to recognise this as a synonym for the three split libs.

- The other thing is that this now forces us to address modularization in the next release…

Modularization: After 10.2

- Geant4 == >3500 .hh files over 145 modules
- "Granular" too small, "Global" both too big/small
- How then to organise code into libraries?
  - Let's start discussion this week on how to approach this!
  - There are CMake, Code organization and C++ API design issues here

- Ultimately it's G4processes that needs the most thought.

- Granular libs are too small in the sense that they create "dependency hell" by not keeping related functionality together.

- Global libs are too big in the sense of G4processes – symbol table limit reached, or users may not want all models (more specialised modules optional).

- Global libs are too small in the sense that core libs may not be integrated
- e.g. is G4intercoms really separate from G4global?

- As part of modularization, we should also look at the plugin architecture so that, for example, physics models, can be loaded at runtime. A proof of principle already exists for Vis drivers using the Poco C++ libraries ("ClassLoader" objects), though a third party API for plugins is not required other than from a convenience perspective.

- Shouldn't be afraid of merging or splitting modules at the code level if it makes sense!

- Should also consider "hiding" code that is implementation detail of a module. We currently install every single header in Geant4, but initial #inclusion studies suggest only 2/3rds are actually used outside their module of origin.

Modularization: Sketch of CMake API

```
# sources.cmake
include(Geant4CMakeAPI)

geant4_add_module(G4MyModule
  PUBLIC_HEADERS G4MyModule.hh
  SOURCES        G4MyModule.cc
  )

geant4_module_link_libraries(G4MyModule
  PUBLIC  G4globman G4csg
  PRIVATE G4intercoms ZLIB::ZLIB
  )

geant4_module_compile_definitions(G4MyModule
  PUBLIC $<$<CONFIG:Debug>:DEBUGMYMODULE>
  )
```

Assumed that we retain basic include/src structure, and also note that this is a sketch only.

- Fewer lines, but more function calls ("god functions" can't be omniscient, don't map to underlying CMake calls well, and are difficult to implement)
- Gives you more control over the parts of the build that you need to be concerned with (linkage, interfaces)

- `geant4_add_module` declares the module name and source composition. Note split of headers/sources. There is good evidence that modules have a large number of headers for implementation details - this is good, but we shouldn't be installing them!
    - Sources still have to be listed explicitly, but as ever this is required for a portable and simple development cycle (and to allow for all source usage patterns in Geant4 without awkward workarounds).
- `geant4_module_link_libraries` declares the modules we use publicly and/or privately. These may include externals, with imported targets used for convenience (and identical behaviour to in-project targets). Note that:
    - Only links to G4 modules (read, granular libraries) need to be declared. The lower level system will decide how modules are composed into physical libraries and what the physical link will be to. This helps to decouple the two decisions and allow the modularisation scheme to evolve independently.
    - NO include_directories are needed. Modern CMake handles these for us.
    - Links can be PUBLIC, where your module exposes symbols from the library, or PRIVATE where they are not.
- Additional functions are provided to handle additional tasks such as compile defintions. Note the intent to support CMake's powerful "generator expressions".

- Ultimately the API is a layer on top of the modern CMake "target" based commands. We need the layer because the ultimate CMake targets are/will be composed of 1 to N "Geant4 modules".

Managing Data Libraries

- We need to look at how data libraries are read/located
- Environment variable based setup awkward with too many potential points of failure
- Access via lots of small files has/is causing issues for MT performance?
- Let's use this week to discuss ideas
  - Slide notes include some of mine

"Geant4 Data Library" : In filesystem terms, a file or directory holding 2-N files
Ben's idea for data location : layered configuration

IF "G4<NAMEOFDATA>DATA" environment variable set:
    Look for <NAMEOFDATA> library at that location.
  - NB: Note use of consistent naming scheme
    - This allows a developer to override and test a new version of the library
ELSE IF "G4DATADIR" environment variable set:
    Look for all libraries under that path, i.e "G4DATADIR/NameOfDataLib"
    - As above, allows simple override to use different location
  - Can also help users - only 1 variable to set rather than 10!
ELSE:
    Look for all libraries under an API path "hard coded" into G4global:
    - "hard coded" may just mean "a relative path from G4global to data", so we'd still have full relocatability overall (e.g. use binreloc on Unix to introspect library for location).
    - for example, the default install layout puts libraries in <prefix>/lib, data in <prefix>/share/geant4/data, so "hard coded" path is just "../share/geant4/data"
  - This is intended to help users so that NO environment setup is required by default.
    - May also provide an API to override this path in an application, e.g.

```
G4DataLibrary::Initialise(); //expected to be needed for self location to work
G4DataLibrary::prependSearchPath("some/abs/or/rel/path); // use this first
```

What I haven't thought about yet…
- Versioning of libraries (and checking versions)
- If G4global handles data library location, how are libraries added/registered? Data libs may be used by more than one physics model.
- Library format (flat files, sqlite3 other?), needs to balance ease of use vs performance. However, can always provide flat files and compile into efficient form.

"Geant4GMake.gmk"

- A new pure-Make implementation of configuration for applications still using old GMake build system
  - See branch `cmake-feature-gmakesupport`
- Behaviour is a "superset" of environment configuration
  - Now fails if option requested which is not available
  - Options enabled by environment/command line/make variables, with reporting of option source

- Promotes environment variables to true Make variables
- Additional logic to prevent compilation with an option that is not available
- More flexible interface for configuring builds:

```
$ export G4LIB_USE_GDML
$ make
```

or

```
$ make G4LIB_USE_GDML=1
```

or

```
G4LIB_USE_GDML=1 (in makefile)
…
make
```

- Aim is still to fully retire "Geant4Make" at some point, but this Make-based configuration is more robust and will be easier to maintain if the lifetime is pushed out.

Geant4GMake.gmk

For users, a very simple one line replacement, with usage:

```
$ make
…
$ make VERBOSE=1
…
$ make geant4_help
```

```
name := exampleB1
G4TARGET := $(name)
G4EXLIB := true

# geant4-config gives make fragment path
include $(shell geant4-config --gmake-file)

.PHONY: all
all: lib bin

include $(G4INSTALL)/config/binmake.gmk

visclean:
    rm -f g4*.prim g4*.eps g4*.wrl
    rm -f .DAWN_*

# Optional geant4_help target(s)
include $(G4GMAKE_TARGETS)
```

- For a demo, just come and ask!
- Testers are welcome, as there are still some teething issues with variable export and target ordering
- Once again, note that aim is still to retire “Geant4 Make” in favour of CMake plus geant4-config.
- Geant4GMake is intended to simplify the management should “life extension” be required. It’s the environment setup that’s the most awkward and error prone part of Geant4.

Summary

- CMake system/documentation in good shape for 10.2
- Please come to Parallel 6b and Plenary 7 for C++11!
- Two major Software Management tasks for 10.3:
  - *Library modularisation*
  - *Data library location and format*
- Let's start the discussion on these during this week!