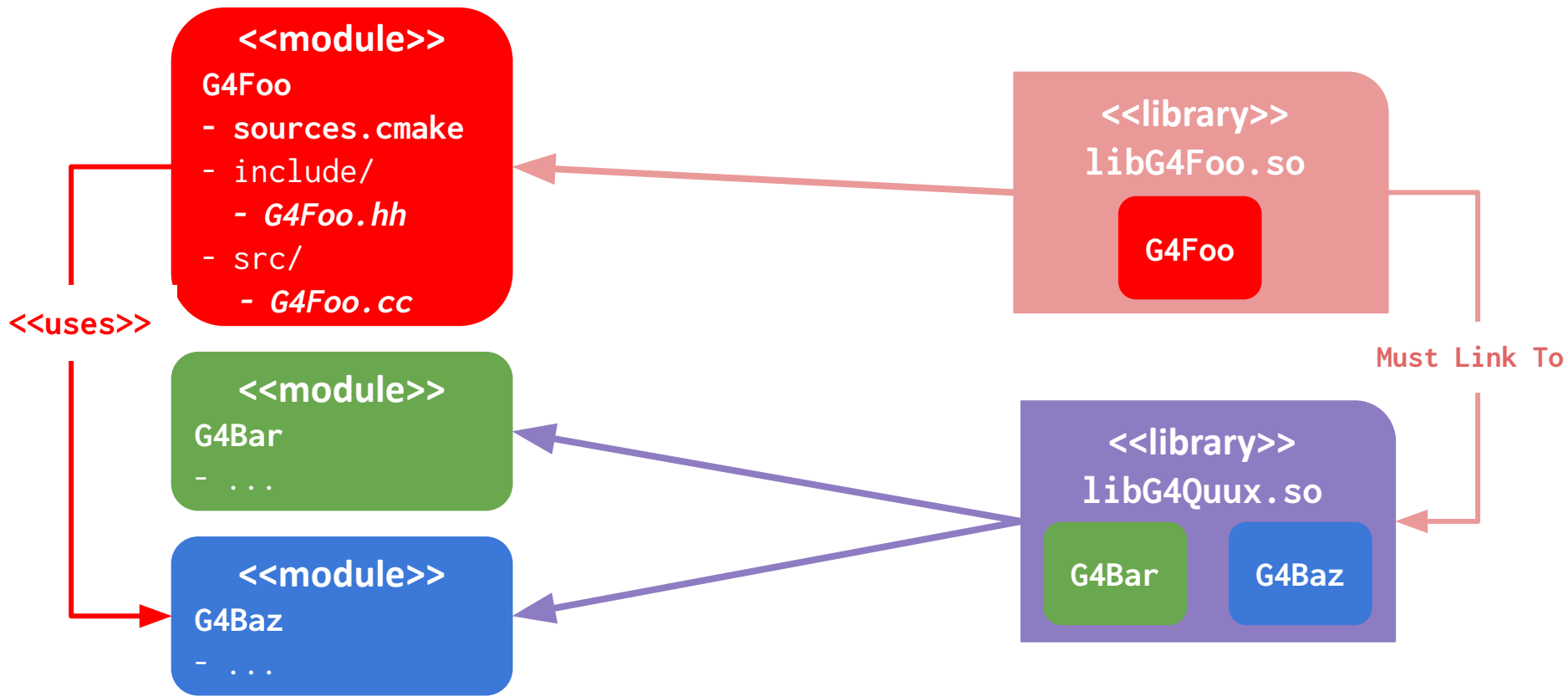




Geant4 Library Modularization Status and Plans

Ben Morgan





Modularization?

How we group source code “modules” into toolkit libraries

Module

cmake > sources.cmake

```
1 # - G4baz module build definition
2
3 # What sources I'm built from
4 geant4_add_module(G4Baz
5   PUBLIC_HEADERS # What interfaces I expose (more later)
6     G4Baz.hh
7   SOURCES        # What files to compile
8     G4Bar.cc)
9
10 # What modules, external libraries I use...
11 geant4_module_link_libraries(G4Baz
12   PUBLIC # What interfaces appear in mine (e.g. G4String)
13     G4globman
14     ${ZLIB_LIBRARIES}
15   PRIVATE # Interfaces I only need for implementation
16     G4leptons)
17
18 # geant4_module_compile_definitions
19 # - As link_libraries, but for -D flags
20 # geant4_module_include_directories
21 # - As link_libraries but for -I flags
```

Internal links
are to *modules*,
not *libraries*

Library

cmake > CMakeLists.txt

```
1 # - G4quux category build
2
3 # Windows DLL annoyance for allocators
4 # - (extern statics inline...)
5 add_definitions(-DG4QUUX_ALLOC_EXPORT)
6
7 # Compose Library from Module sources.cmake
8 geant4_global_library_target(NAME G4quux
9   COMPONENTS
10     bar/sources.cmake
11     baz/sources.cmake )
12
13 # Possible interface once modularization scheme clear
14 # and recursion into deep subdirectories confirmed not
15 # to slow down config/build
16 #
17 # geant4_add_library(G4quux MODULES G4bar G4baz)
18
19 # NB: NO _link_libraries etc versions. Only modules
20 # can define build properties
```

CMake commands for Developers/Coordinators

Follows CMake “[target](#)” commands
and “[usage requirements](#)” as far as
possible.

```
CMakeLists.txt — geant4-dev.git [SSH: 192.168.1.68]
M CMakeLists.txt x
source > M CMakeLists.txt
27 add_subdirectory(intercoms)
28 add_subdirectory(interfaces)
29 add_subdirectory(materials)
30 add_subdirectory(parameterisations)
31 add_subdirectory(particles)
32 add_subdirectory(persistency)
33 add_subdirectory(physics_lists)
34 add_subdirectory(processes)
35 add_subdirectory(readout)
36 add_subdirectory(run)
37 add_subdirectory(tasking)
38 add_subdirectory(track)
39 add_subdirectory(tracking)
40 add_subdirectory(visualization)
41
42 # - Compose libs
43 geant4_compose_targets()
44
```

- Recurse into subdirs to add modules, libraries
- Heavy lifting function
 - **Must** be here because we can't compose until all modules/libraries are defined
- *Checks:*
 - No orphan Modules
 - Module not composed into >1 Library
- *Calculates:*
 - Needed CMake add_library calls
 - Needed CMake target_link_library etc calls, e.g.. works out that "libG4Foo" needs to link to "libG4Quux"
- *Tests*
 - No Module-Module cycles (at CTest time)
 - CMake checks Library-Library cycles

Under The Hood...


Composition still done at Category level (one dir down) as before

Changes for 11.0

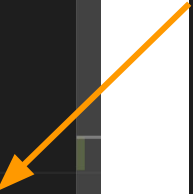
- No changes to user code or build scripts(*) for 11.0
 - *An internal build change, classic “global” libraries still generated*
 - *Changes to library composition can only be made after final retirement of GNUMake system for users(*)*
- Future changes to library composition would break link-interface, and thus ABI compatibility
 - *Probably implies a new major version*
 - *Makes no change to API*
 - *We don't (AFAIK) require/enforce ABI compatibility between minor releases*
 - *On the other hand, studying the library composition naturally involves ABI, so perhaps this checking can be used to report both API and ABI changes!*
 - *Not saying we enforce compatibility, but useful to know/report changes!*

Beyond 11.0: How to compose libs, or, why change?

- Neither “global” (current) or “granular” (1 Module to 1 Library) optimal
 - *“Granular” composition far too small (~145 libraries...)*
 - *“Global” composition too big (G4processes), too small (“kernel” functionality split across many libs)*
- Criteria for “just right” coming up, but some up front caveats:
 - ***DON’T*** want CMake option(s) to choose (go ahead, calculate the permutations (and number of CI jobs needed) with 145 libraries!
 - ***DO*** want pattern to be easily changed when needed
 - ***... but don’t these conflict?***
 - ***... Configurable Composition file provides customization point for users (if they really need it), and to enable studies with no source changes***

```
CMakeLists.txt — geant4-dev.git [SSH: 192.168.1.68]
M CMakeLists.txt M ●
source > M CMakeLists.txt
27 add_subdirectory(intercoms)
28 add_subdirectory(interfaces)
29 add_subdirectory(materials)
30 add_subdirectory(parameterisations)
31 add_subdirectory(particles)
32 add_subdirectory(persistency)
33 add_subdirectory(physics_lists)
34 add_subdirectory(processes)
35 add_subdirectory(readout)
36 add_subdirectory(run)
37 add_subdirectory(tasking)
38 add_subdirectory(track)
39 add_subdirectory(tracking)
40 add_subdirectory(visualization)
41
42  Load composition file - could be modified by users
43 include(${CMAKE_CURRENT_SOURCE_DIR}/G4LibComp.cmake)
44
45 # - Compose libs
46 geant4_compose_targets()
47
```

Recurse into these, loading all sources.cmake



```
G4LibComp.cmake U X
source > G4LibComp.cmake
1 # - Compose Libraries from Modules
2 # NB: All Modules must be defined by this point
3 geant4_add_library(G4global
4     MODULES
5         G4globman
6         G4hepgeometry
7         G4hepnumerics
8         G4heprandom)
9
10 # Order shouldn't matter as long as lib/module names unique
11 # NB: Module/library names in different namespaces
12 geant4_add_library(G4run MODULES G4run)
13
14 geant4_add_library(G4event MODULES G4event)
15
16 # This form needed if deep recursion hurts build performance
17 geant4_add_library(G4processes
18     MODULE_SOURCES
19         processes/management/sources.cmake
20         # ... and so on)
21
```

Configurable Composition

In progress: Issues/limitations from -D flags for Allocators, CMake target definitions, export to Geant4Config.cmake

Criteria for composition (for discussion)

- **Optional modules (e.g. GDML) should always be in their own Library**
 - *Library existence => availability, easier to compile/package*
- **Compose Libraries from modules at similar depth in module DAG?**
 - *Categories may be a mix of modules across many levels*
 - *Hints of “kernel” vs “implementation” layers during migration process*
 - *E.g. (abstract) base classes, core/internal algorithms in “libG4kernel”, specific implementations, e.g. physics models, at higher level*
 - *Need to balance against too big, or new “variant”, libraries*
- **Don't compromise performance**
 - *Mostly for shared libraries, current “global” composition is the baseline*
 - [Guilherme's presentation from Tuesday](#): *can we reduce interlibrary calls, even hide truly internal symbols?*

Test Cases

1. **Split out G4gdml module from G4persistency library -> libG4gdml.so**
 - a. *Canonical case, even likely in 11.0 if GNUmake retired*
 - b. *Longer term would be UI/Vis on basis on external lib used, but a lot more work and tied in with out aspects like compiled in vs plugin drivers.*
2. **Find “layers/generations” in Module DAG (i.e. topological sorting)**
 - a. *Use this as a guide to prepare different “G4LibComp.cmake” files (NB No changes to source code or category organization needed)*
 - b. *Build/Profile against “global” baseline, feedback results*
 - c. *Identify candidate compositions and discuss/decide on results with all WGs*
 - d. *Roll out agreed new composition in next appropriate minor/major Release*
3. **Related task for all: identify Public/Private APIs of your module(s)**

```
G4Foo.hh U
source > global > management > include > G4Foo.hh > ...
1  #ifndef G4F00_HH
2  #define G4F00_HH
3
4  #include "globals.hh"
5
6  // This is G4Foo's only Public Header/API
7  // Note that use of G4DeepThought is hidden
8  class G4Foo
9  {
10 |   G4String TheAnswer(const G4String& question);
11 };
12
13 #endif

G4Foo.cc 4, U
source > global > management > src > G4Foo.cc > ...
1  #include "G4Foo.hh"
2  // Only #included here, never intended for use elsewhere!
3  #include "G4DeepThought.hh"
4
5  G4String G4Foo::TheAnswer(const G4String& question)
6  {
7      // Use of G4DeepThought is an implementation detail of G4Foo
8      // .. later on it can change to G4Siri, G4Alexa...
9      G4DeepThought x;
10     x.ReadQuestion(question);
11     G4String answer = x.GetTheAnswer();
12
13     if(answer == "42")

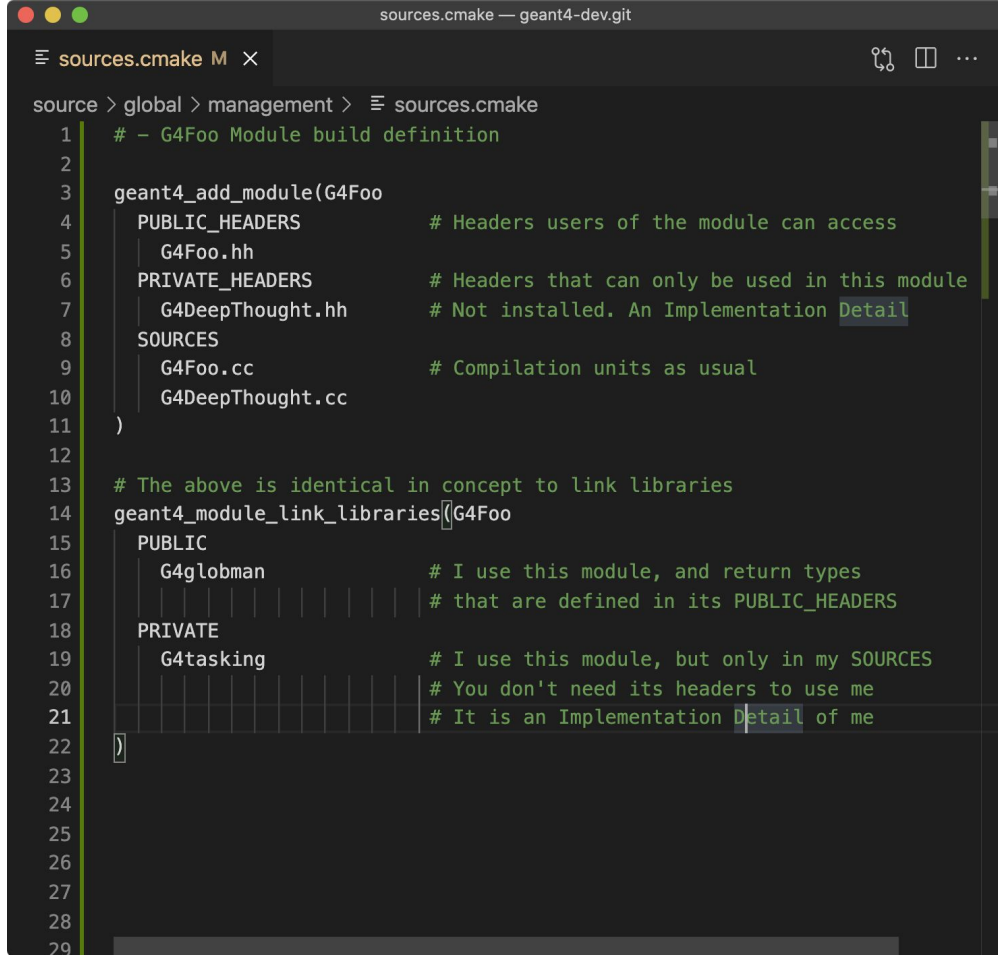
G4DeepThought.hh U
source > global > management > include > G4DeepThought.hh > ...
1  #ifndef G4DEEPTHOUGHT_HH
2  #define G4DEEPTHOUGHT_HH
3
4  // The "Private Header"
5  #include "globals.hh"
6
7  class G4DeepThought
8  {
9      void ReadQuestion(const G4String&);
10     G4String GetTheAnswer();
11 };
12
13 #endif
```

Identifying Public/Private Module APIs

3784 header files, i.e. interfaces, in Geant4. Which of these are intended for use outside their module?

Why does this help?

- **Documentation:** Clear which headers/classes are part of Geant4 API, which are implementation details
- Reduce installed size of toolkit
 - *Trivial? but...*
- **Private API \approx Hidden library symbol**
 - *Hiding symbols reduces **library size***
 - [May improve performance of shared libraries via removal of trampolines \(PLT lookups\)](#)
- `geant4_add_module` will provide an interface for declaring private headers after GNUmake removal completed



```
sources.cmake M X
source > global > management > sources.cmake
1  # - G4Foo Module build definition
2
3  geant4_add_module(G4Foo
4      PUBLIC_HEADERS      # Headers users of the module can access
5      G4Foo.hh
6      PRIVATE_HEADERS     # Headers that can only be used in this module
7      G4DeepThought.hh   # Not installed. An Implementation Detail
8      SOURCES
9      G4Foo.cc            # Compilation units as usual
10     G4DeepThought.cc
11 )
12
13 # The above is identical in concept to link libraries
14 geant4_module_link_libraries(G4Foo
15     PUBLIC
16     G4globman            # I use this module, and return types
17                          # that are defined in its PUBLIC_HEADERS
18     PRIVATE
19     G4tasking            # I use this module, but only in my SOURCES
20                          # You don't need its headers to use me
21                          # It is an Implementation Detail of me
22 )
23
24
25
26
27
28
29
```

Summary and Discussion

1. CMake tooling for modularization in production with final tweaks incoming
2. Some criteria for modularization and workflow defined - are these reasonable? What others should be considered?
 - a. *Informs decisions on how to proceed with noted Test Cases*
3. You should start identifying Public/Private APIs of your modules to see if we can reduce installed size of Geant4 or increase performance with symbol hiding