Coding Rules & Guidelines

I. Hrivnacova, IPN Orsay

16th Geant4 Collaboration Meeting, 19 - 23 September, SLAC

Outline

- Introduction
- Coding Guidelines for Geant4 developers
  - Their check-ability
  - Following the guidelines in the Geant4 code
- New N&E Examples Coding Guidelines
  - Motivations

Introduction

- From Geant4 Coding Guideline Web page:
- "For a world-wide collaboration like GEANT4, it is therefore important not to impose rigid rules or style-conventions, but to maintain flexible and adequate guidelines for programming and coding styles."
- "C++ was designed to support data abstraction and object-oriented programming in addition to traditional C programming techniques. It was not meant to force any particular programming style upon all users" (B.Stroustrup).

- *Set of Geant4 Coding Guidelines is very "light"*
- *http://geant4.web.cern.ch/geant4/collaboration/coding_guidelines.sh*

Geant4 Coding Guidelines Sets

The Geant4 coding guidelines are grouped in the following sets:

- Programming guidelines
  - *Aid adherence to the object-oriented paradigm (data-hiding, encapsulation, etc ...), and promote performance and portability*
- Coding-style guidelines
  - *Promote code maintainability and readability*
- Guidelines for ISO/ANSI Compliant Code
  - *Facilitate the porting of Geant4 to different systems/architectures*

The guidelines listed on the following slides are preceded with

- if their automatic checking seems to be possible
- If their automatic checking does not seem to be possible or
- if checking would require precising the checked rule

Geant4 Coding Guidelines

Programming

The following guidelines for using the features of the programming language aid adherence to the object-oriented paradigm (data-hiding, encapsulation, etc ...), and promote performance and portability:

- Every class should have at least one constructor and a destructor.
- Each class should have an assignment operator and a copy constructor
- Data members should be made private/protected and in case accessed by inline functions.
- The use of global variables or functions should be avoided.
- The use of friend classes should be avoided where possible.
- The use of type casting, especially from const* or const, should be avoided.
- Hard-coded numbers within the code must be avoided; const should be used instead.
- Instead of the raw C types, G4 types should be used.

Geant4 Coding Guidelines

Coding-style

The following coding-style guidelines promote code maintainability and readability

- The public, protected and private keywords must be used explicitly in the class declaration.
- Self-explanatory, English names for constants, variables and functions should be used.
- The code should be properly indented.
- Each GEANT4 class name should begin with G4.
- Each header file should contain only one or related class declarations.
- The implementation code for a class should be contained in a single source file.
- Each header file must be protected from multiple inclusions.
- Each file should contain a short header about the author, updates (CVS) and date.

Geant4 Coding Guidelines
for ISO/ANSI Compliant Code

Using the following style guidelines will facilitate the porting of Geant4 to different systems/architectures:

- Where possible, consider using the defined type G4String instead of STL std::string in the code.
- Use the following keywords defined in global: G4cout, G4cerr, G4cin, G4endl instead of: cout, cerr, cin, endl.
- In every other case prepend std::. For example, for stream manipulators: std::ws, std::setprecision, std::setw.....
- Prepend std:: for every STL type declaration. Example: std::vector myvector;

Looking at the code

- Inspecting geant4-09-04-ref07 source
- ALICE Code Checker
  - Developed by Paolo Tonella and Surafel Lemma Abebe, FBK Trentino
  - Requires:
    - 1. Java
    - 2. srcML toolkit (,which includes src2srcml) from: http://www.sdml.info/projects/srcml/
  - Directly applicable to 5 rules (after renaming classes to have extensions .h and .cxx)
- Find + grep for types to be avoided
  - Be careful: not to search 'double' (would count G4double as well) but ' double '; not to count comment lines ...

Are The Guidelines Followed ?

No Geant4 automatic checking tool
=> one can find many of "to be avoided" in the code

- Constructor/destructor
- No friend classes
- Class prefix G4

- Casting
- Hard coded numbers
- Proper indented code
- Author, date
- ...

?

- Assignment operator & copy constructor missing
- Public data members - in 90 classes !
- Global variables – in 47 classes
- Raw C types (double, ...)
- Multiple inclusion protection missing in – in 4 classes
- Std::string instead of G4String
- cout, cerr, endl instead of G4...

New N&E Examples
Coding Guidelines

- In addition to Geant4 guidelines
- The new rules were discussed and approved at a WG phone meeting in June
  - with finalizing last rules by votes on Doodle poll
- Grouped in 5 sets:
  - Naming conventions, Coding rules, Style rules, Documentation, Application guidelines
- More rigid than Geant4 guidelines
- To improve the code
  - Quality
  - Readability
  - Understandability

- http://geant4.web.cern.ch/geant4/collaboration/working_groups/novi

N&E Examples Coding Guidelines

Naming Conventions

- 1.1. Classes in common (the common place with sharable classes used by more extended examples) use names starting with the prefix 'ExG4'
  - If there are more than one classes of the same name, then all classes names are followed by a number: (00), 01, 02, 03 ...
  - The number 00 should be used for helper classes which are not supposed to be directly usable in user application, eg. physics list with geantino only.
- Motivation : The generic naming scheme make the classes directly usable in a user application (no clash with existing user application classes names) and it facilitates the procedure of importing classes in concrete examples

Naming Conventions

- 1.2. Classes in extended examples (features) use names with a prefix, specific to the demonstrated feature and different from the one reserved for classes in common, or names without a prefix.
  - Motivation: Unique class names facilitate referencing the classes from other code. (Accepting also the names without a prefix was a compromise with EM standard physics developers.)
- 1.3. Class member functions start with an upper case letter.
- 1.4. Class data members start with a prefix "f" followed with an upper case letter.
- 1.5. Local variables and functions argument names start with a lower case letter except for the names starting with known acronyms in capital case letters.
  - Motivation: These rules makes easier to understand the code.

Coding rules

- 2.1. Re-declare virtual functions in the header files with keyword virtual.
  - Motivation: It makes easier to see which functions are prescribed in the base class (and usually called by kernel)
- 2.2. Provide the initialization list of the class data members (and base class if present) in all class constructors.
  - Motivation: This prevents from use of uninitialized values.
- 2.3. Do not introduce dummy functions or classes if they have no use in the example.
  - Motivation: Let's not complicate the code more than needed.
- 2.4. All commands implemented in messengers should be demonstrated in a run macro (can be done also via commented lines).
  - Motivation: Makes them easier to test.

Style rules

- 3.1. Avoid using long lines (more than 80 characters) where possible, avoid using tabulators.
- 3.2. Each function implementation in the .cc file should be preceded by the agreed separator line:

  //....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....oooOO0OOooo.....
- 3.3. Avoid using more than one empty lines or personalized separators in the code.
  - Motivation: Improve readability of the code.

Documentation

- 4.1. Each example is provided with a README text file and its modified version .README for automatic generation of the Web documentation with Doxygen.
  - The latter differs from the former only by the modifications needed for a correct representation of the file on the Web.
- 4.2. The files with C++ code start with a standard header including Geant4 copyright, the SVN Id keyword and a file description.
  - Example of a file description:

```
/// \file N1DetectorConstruction.hh
/// \brief Definition of the N1DetectorConstruction class
```

- 4.3. Each class contains a description of the class functionality placed just before a class definition in the class header file (.hh)
  - The comment lines with a class description start with /// (instead of standard //) in order to be recognized by Doxygen.

Application conventions

- 5.1. It is recommended to define materials with using NIST manager unless there is a specific reason for explicit material definition.
  - Motivation: NIST manager guaranties the correct values for material properties and users should be encouraged to get familiar with it.
- 5.2. It is recommended to use the physics list classes and physics builders provided in Geant4 unless there is a specific reason for using an explicitly defined physics list.
  - Motivation: The physics lists and builders classes were defined by the experts, they also allow the users to provide a reference of their physics setup

Conclusion

- Geant4 has quite a light set of coding guidelines
- Their following by developers is on volunteer basis
  - Most of the rules are check-able, but no automatic checking is performed
  - The code is not always compliant with the rules
- New set of coding guidelines was agreed for new N&E examples
- Introducing a tool for automatic checking can improve the Geant4 code quality