Modernizing Gaudi

M. Clemencic

September 21, 2016

CERN - LHCb

Introduction

- Gaudi initially developed with C++98
- Main development guideline: *backward compatibility*
- After almost 20 years
  - old style of coding
  - a lot of legacy/unused code

Table of contents

C++

A new programming language

a new programming language in town

A new programming language

a new programming language in town

C++14

A new programming language

a new programming language in town

C++14

- more powerful
- more robust
- easier to use

What's wrong with Gaudi?

- Old fashion, for example
  - a lot of `std::vector<T*>`
  - `for(std::vector::iterator it = …`
  - `new` + `delete`

What's wrong with Gaudi?

- Old fashion, for example
  - a lot of `std::vector<T*>`
  - `for(std::vector::iterator it = …`
  - `new` + `delete`
- New C++ standards have better ways
  - `std::vector<T>` can be very efficient
  - `for(auto element: container) …`
  - handles / smart pointers

What's wrong with Gaudi?

- Old fashion, for example
  - a lot of `std::vector<T*>`
  - `for(std::vector::iterator it = …`
  - `new` + `delete`
- New C++ standards have better ways
  - `std::vector<T>` can be very efficient
  - `for(auto element: container) …`
  - handles / smart pointers
- But there's worse
  - we are mixing the two styles

What can we do?

- Replace old style code with new style
- Follow “C++ Core Guidelines”
- But it requires changing interfaces

What can we do?

- Replace old style code with new style
- Follow “C++ Core Guidelines”
- But it requires changing interfaces

what about backward compatibility?

Legacy code

Origins

Backward compatibility is useful,

Origins

Backward compatibility is useful, but not for too long.

- improvements in the framework should not break current code
- existing code should be adapted to new features

Origins

Backward compatibility is useful, but not for too long.

- improvements in the framework should not break current code
- existing code should be adapted to new features

It’s inconceivable that we cannot remove a method used
by an unused class written 20 years ago.

What to do?

C++14 provide a standard way to mark deprecated methods/classes.

What to do?

C++14 provide a standard way to mark deprecated methods/classes.

- Mark old code as `[[deprecated("...")]]`
- Provide useful messages
  - when it will be removed
  - what to use instead
  - possibly a link to a web page describing the change
- Remove the code when promised

What to do?

C++14 provide a standard way to mark deprecated methods/classes.

- Mark old code as `[[deprecated("...")]]`
- Provide useful messages
  - when it will be removed
  - what to use instead
  - possibly a link to a web page describing the change
- Remove the code when promised

Some developers may not like having to maintain their code (?), but, usually, the changes aim to simplify user code.

Summary

Summary

- Gaudi is not dead, but it's very old
- Using the new C++ will make it more maintainable
- Legacy code should go away
- It's also a good moment to review some concepts
  - properties (merge request 182)
  - reference counting (GAUDI-1114)
  - ...