A CMake-based build and configuration framework

M. Clemencic, P. Mato

CERN - LHCb

CHEP 2012 - New York
Outline

Introduction

Requirements

Design

Implementation

Conclusions
Outline

Introduction

Requirements

Design

Implementation

Conclusions
Building LHCb Software

- Software projects need build tools
  - make, autotools, ant, jam, ...
- HEP software has special requirements
  - reproducibility and control
- CMT: a Configuration Management Tool
  - manages concurrent versions of projects and packages
  - dynamic runtime environment
Building LHCb Software

• Software projects need build tools
  • make, autotools, ant, jam, . . .
• HEP software has special requirements
  • reproducibility and control
• CMT: a Configuration Management Tool
  • manages concurrent versions of projects and packages
  • dynamic runtime environment
Building LHCb Software

- Software projects need build tools
  - make, autotools, ant, jam, …
- HEP software has special requirements
  - reproducibility and control
- CMT: a Configuration Management Tool
  - manages concurrent versions of projects and packages
  - dynamic runtime environment
CMT Projects Layout

- Code organized in packages
  - configuration file
    - use other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - container package
  - policy package
- Coexisting versions
CMT Projects Layout

- Code organized in packages
  - configuration file
    - use other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - container package
  - policy package
- Coexisting versions
CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - *container* package
  - *policy* package
- Coexisting versions
CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
    - *container* package
    - *policy* package
- Coexisting versions
CMT Projects Layout

- Code organized in packages
  - configuration file
    - use other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
    - container package
    - policy package
- Coexisting versions
CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - *container* package
  - *policy* package
- Coexisting versions
- Code organized in packages
  - configuration file
    - use other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
    - container package
    - policy package
- Coexisting versions
CMT Projects Layout

- Code organized in packages
  - configuration file
    - \textit{use} other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
    - \textit{container} package
  - \textit{policy} package
- Coexisting versions
LHCb Software & CMT

• Projects Layout
  • enforced by CMT

• Customizations
  • patterns (functions)
  • makefile fragments

• Optimizations
  • improved dependency computation
  • wrapper for parallel build

• Extensions
  • special command to prepare the environment

• Many functionalities not used
LHCb Software & CMT

• Projects Layout
  • enforced by CMT

• Customizations
  • patterns (functions)
  • makefile fragments

• Optimizations
  • improved dependency computation
  • wrapper for parallel build

• Extensions
  • special command to prepare the environment

• Many functionalities not used
LHCb Software & CMT

- Projects Layout
  - enforced by CMT
- Customizations
  - patterns (functions)
  - makefile fragments
- Optimizations
  - improved dependency computation
  - wrapper for parallel build
- Extensions
  - special command to prepare the environment
- Many functionalities not used
LHCb Software & CMT

- Projects Layout
  - enforced by CMT
- Customizations
  - patterns (functions)
  - makefile fragments
- Optimizations
  - improved dependency computation
  - wrapper for parallel build
- Extensions
  - special command to prepare the environment
- Many functionalities not used
LHCb Software & CMT

- Projects Layout
  - enforced by CMT

- Customizations
  - patterns (functions)
  - makefile fragments

- Optimizations
  - improved dependency computation
  - wrapper for parallel build

- Extensions
  - special command to prepare the environment

- Many functionalities not used
Overriding Packages

• Special feature of CMT
  • a project can override packages from projects it uses

• Extremely used in LHCb
  • Pick up bugfixes before releases
  • Lightweight development environment
Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment

Diagram:

- BaseProject
  - PackA v4r0
  - PackB v2r4
  - PackC v3r1

- Derived v5r3
  - PackD v1r3
  - PackE v2r2

(use arrow from BaseProject to Derived v5r3)
Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment

![Diagram showing BaseProject using PackB v2r4 and PackC v3r1 and Derived v5r3 using PackD v1r3 and PackE v2r2]
Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment
Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment
Overriding Packages

- **Special feature of CMT**
  - a project can override packages from projects it uses
- **Extremely used in LHCb**
  - Pick up bugfixes before releases
  - Lightweight development environment

![Diagram](image-url)

- Release
  - PackA v4r0
  - PackB v2r4
  - PackC v3r1
- LocalDevel
Overriding Packages

• Special feature of CMT
  • a project can override packages from projects it uses
• Extremely used in LHCb
  • Pick up bugfixes before releases
  • Lightweight development environment
Why to change?

• CMT has got limitations
  • OK on small projects, but very slow on big projects
  • limited logic of configuration language

• We know what we need
  • we do not use all the features in CMT
  • we extended and customized it to fit better our needs

• New products on the market
• Good time to investigate something new
Why to change?

- CMT has got limitations
  - OK on small projects, but very slow on big projects
  - limited logic of configuration language
- We know what we need
  - we do not use all the features in CMT
  - we extended and customized it to fit better our needs
- New products on the market
- Good time to investigate something new
Why to change?

• CMT has got limitations
  • OK on small projects, but very slow on big projects
  • limited logic of configuration language
• We know what we need
  • we do not use all the features in CMT
  • we extended and customized it to fit better our needs
• New products on the market
  • Good time to investigate something new
Why to change?

• CMT has got limitations
  • OK on small projects, but very slow on big projects
  • limited logic of configuration language
• We know what we need
  • we do not use all the features in CMT
  • we extended and customized it to fit better our needs
• New products on the market
• Good time to investigate something new
<table>
<thead>
<tr>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>Requirements</td>
</tr>
<tr>
<td>Design</td>
</tr>
<tr>
<td>Implementation</td>
</tr>
<tr>
<td>Conclusions</td>
</tr>
</tbody>
</table>
Requirements

- **Flexibility**
  - organize code in projects and packages
  - easily add/remove/move packages

- **Override packages**
  - derived projects can override packages

- **Simplicity**
  - minimalistic language (no details)

- **Runtime environment**
  - easy set-up for any version of any project

- **Smooth migration**
  - allow for adiabatic adoption of new framework
Requirements

• Flexibility
  • organize code in projects and packages
  • easily add/remove/move packages

• Override packages
  • derived projects can override packages

• Simplicity
  • minimalistic language (no details)

• Runtime environment
  • easy set-up for any version of any project

• Smooth migration
  • allow for adiabatic adoption of new framework
Requirements

- **Flexibility**
  - organize code in projects and packages
  - easily add/remove/move packages
- **Override packages**
  - derived projects can override packages
- **Simplicity**
  - minimalistic language (no details)
- **Runtime environment**
  - easy set-up for any version of any project
- **Smooth migration**
  - allow for adiabatic adoption of new framework
Requirements

• Flexibility
  • organize code in projects and packages
  • easily add/remove/move packages
• Override packages
  • derived projects can override packages
• Simplicity
  • minimalistic language (no details)
• Runtime environment
  • easy set-up for any version of any project
• Smooth migration
  • allow for adiabatic adoption of new framework
Requirements

- **Flexibility**
  - organize code in projects and packages
  - easily add/remove/move packages
- **Override packages**
  - derived projects can override packages
- **Simplicity**
  - minimalistic language (no details)
- **Runtime environment**
  - easy set-up for any version of any project
- **Smooth migration**
  - allow for adiabatic adoption of new framework
What to use?

- New products with respect to 10 years ago
- Many tools are too specific
  - wrong language (we need C++ and Python)
  - non portable (Unix only)
  - only specific type of projects
- Few are generic and flexible
  - CMake, SCons, …
- **CMake** is powerful and widely used (e.g. KDE)
What to use?

- New products with respect to 10 years ago
- Many tools are too specific
  - wrong language (we need C++ and Python)
  - non portable (Unix only)
  - only specific type of projects
- Few are generic and flexible
  - CMake, SCons, ...
- CMake is powerful and widely used (e.g. KDE)
What to use?

- New products with respect to 10 years ago
- Many tools are too specific
  - wrong language (we need C++ and Python)
  - non portable (Unix only)
  - only specific type of projects
- Few are generic and flexible
  - CMake, SCons, ...
- **CMake** is powerful and widely used (e.g. KDE)
What to use?

- New products with respect to 10 years ago
- Many tools are too specific
  - wrong language (we need C++ and Python)
  - non portable (Unix only)
  - only specific type of projects
- Few are generic and flexible
  - CMake, SCons, ...
- **CMake** is powerful and widely used (e.g. KDE)
Does CMake fit?

• Pros
  • projects and subdirectories
  • very powerful (complete) language
  • library of modules for configuration
  • extensible with functions and macros
  • properties

• Cons
  • no support for runtime environment
  • cannot override targets
  • transitivity of libraries, but not of includes

Something just fit, something not, but the language and the features are powerful enough to outweigh the limitations.
Does CMake fit?

• Pros
  • projects and subdirectories
  • very powerful (complete) language
  • library of modules for configuration
  • extensible with functions and macros
  • properties

• Cons
  • no support for runtime environment
  • cannot override targets
  • transitivity of libraries, but not of includes

Something just fit, something not, but the language and the features are powerful enough to outweigh the limitations.
Does CMake fit?

• Pros
  • projects and subdirectories
  • very powerful (complete) language
  • library of modules for configuration
  • extensible with functions and macros
  • properties

• Cons
  • no support for runtime environment
  • cannot override targets
  • transitivity of libraries, but not of includes

Something just fit, something not, but the language and the features are powerful enough to outweigh the limitations.
Outline

Introduction

Requirements

Design

Implementation

Conclusions
Main Elements

- **Projects**
  - entry point to the build configuration
  - coordinate the hosted subdirectories

- **Subdirectories**
  - equivalent to packages in CMT
  - describe the components to build/install

- **Toolchains**
  - replace the fixed set of external libraries
  - allow special settings (e.g. compiler)

- **Properties**
  - used to communicate between components

- **Exports**
  - communicate between projects
Main Elements

• Projects
  • entry point to the build configuration
  • coordinate the hosted subdirectories

• Subdirectories
  • equivalent to packages in CMT
  • describe the components to build/install

• Toolchains
  • replace the fixed set of external libraries
  • allow special settings (e.g. compiler)

• Properties
  • used to communicate between components

• Exports
  • communicate between projects
Main Elements

- **Projects**
  - entry point to the build configuration
  - coordinate the hosted subdirectories
- **Subdirectories**
  - equivalent to packages in CMT
  - describe the components to build/install
- **Toolchains**
  - replace the fixed set of external libraries
  - allow special settings (e.g. compiler)
- **Properties**
  - used to communicate between components
- **Exports**
  - communicate between projects
Main Elements

• Projects
  • entry point to the build configuration
  • coordinate the hosted subdirectories

• Subdirectories
  • equivalent to packages in CMT
  • describe the components to build/install

• Toolchains
  • replace the fixed set of external libraries
  • allow special settings (e.g. compiler)

• Properties
  • used to communicate between components

• Exports
  • communicate between projects
Main Elements

- Projects
  - entry point to the build configuration
  - coordinate the hosted subdirectories
- Subdirectories
  - equivalent to packages in CMT
  - describe the components to build/install
- Toolchains
  - replace the fixed set of external libraries
  - allow special settings (e.g. compiler)
- Properties
  - used to communicate between components
- Exports
  - communicate between projects
Outline

Introduction
Requirements
Design
Implementation
Conclusions
Layout

- **Main CMake module**
  - core of the configuration framework
  - contains all the functions and extensions
- **Compile flags module**
  - module for compile/link flags and settings
- **Toolchains modules**
  - define search paths for custom build of external libraries
- **Contributed find modules**
  - `FindX.cmake` modules not provided by standard CMake
- **Custom tool for environment manipulation**
  - Python script to prepare the environment from simple configuration
• Main CMake module
  • core of the configuration framework
  • contains all the functions and extensions

• Compile flags module
  • module for compile/link flags and settings

• Toolchains modules
  • define search paths for custom build of external libraries

• Contributed *find* modules
  • FindX.cmake modules not provided by standard CMake

• Custom tool for environment manipulation
  • Python script to prepare the environment from simple configuration
Layout

• Main CMake module
  • core of the configuration framework
  • contains all the functions and extensions

• Compile flags module
  • module for compile/link flags and settings

• Toolchains modules
  • define search paths for custom build of external libraries

• Contributed *find* modules
  • *FindX.cmake* modules not provided by standard CMake

• Custom tool for environment manipulation
  • Python script to prepare the environment from simple configuration
Layout

• Main CMake module
  • core of the configuration framework
  • contains all the functions and extensions

• Compile flags module
  • module for compile/link flags and settings

• Toolchains modules
  • define search paths for custom build of external libraries

• Contributed *find* modules
  • *FindX.cmake* modules not provided by standard CMake

• Custom tool for environment manipulation
  • Python script to prepare the environment from simple configuration
Layout

• Main CMake module
  • core of the configuration framework
  • contains all the functions and extensions
• Compile flags module
  • module for compile/link flags and settings
• Toolchains modules
  • define search paths for custom build of external libraries
• Contributed *find* modules
  • *FindX.cmake* modules not provided by standard CMake
• Custom tool for environment manipulation
  • Python script to prepare the environment from simple configuration
Configuration Files

Project Configuration

- Declare Project

- configuration file

- command
Configuration Files

- Declare Project
- Discover Subdirectories
- Import Projects
- Add Subdirectories
- Declare Global Products
- Export Project Infos

Configuration file
command
internal operation
call
information flow
Configuration Files

Project Configuration

- Declare Project
- Discover Subdirectories
- Import Projects
- Add Subdirectories
- Declare Global Products
- Export Project Infos

configuration file
command
internal operation
call
information flow
Configuration Files

- Project Configuration
  - Declare Project
  - Discover Subdirectories
  - Import Projects
  - Add Subdirectories
  - Declare Global Products
  - Export Project Infos

- Subdirectory Config.
  - Declare Dependencies
  - Require Ext. Libraries
  - Declare Products
  - Install Files
  - Declare Tests
  - Declare Environment

configuration file
command
internal operation
call
information flow
Configuration Files

Project Configuration

Declare Project

Discover Subdirectories

Import Projects

Add Subdirectories

Declare Global Products

Export Project Infos

Subdirectory Config.

Declare Dependencies

Require Ext. Libraries

Declare Products

Install Files

Declare Tests

Declare Environment

---

configuration file
command
internal operation
call
information flow
**Configuration Files**

- **Project Configuration**
  - Declare Project
  - Discover Subdirectories
  - Import Projects
  - Add Subdirectories
  - Declare Global Products
  - Export Project Infos

- **Subdirectory Config.**
  - Declare Dependencies
  - Require Ext. Libraries
  - Declare Products
  - Install Files
  - Declare Tests
  - Declare Environment

- **Toolchain Config.**
  - Declare LCG AA Projects
  - Declare Custom Compilers
  - Declare External Projects
  - Produce Search Paths

Diagram icons:
- Configuration file
- Command
- Internal operation
- Call
- Information flow
Configuration Files

- Project Configuration
  - Declare Project
  - Discover Subdirectories
  - Import Projects
  - Add Subdirectories
  - Declare Global Products
  - Export Project Infos
- Subdirectory Config.
  - Declare Dependencies
  - Require Ext. Libraries
  - Declare Products
  - Install Files
  - Declare Tests
  - Declare Environment
- Toolchain Config.
  - Declare LCG AA Projects
  - Declare Custom Compilers
  - Declare External Projects
  - Produce Search Paths

configuration file
command
internal operation
call
information flow
From CMT to CMake

- **Compatibility**
  - produce same filesystem hierarchy as CMT
  - preserve CMT configuration files

- **Translation tool**
  - Python script to analyze and translate configurations
  - used only for the first translation

- **Migration plan**
  - Validate framework (Summer)
  - Migrate project by project (LHC shutdown)
  - Phase out CMT (still used in old versions)
From CMT to CMake

• Compatibility
  • produce same filesystem hierarchy as CMT
  • preserve CMT configuration files

• Translation tool
  • Python script to analyze and translate configurations
  • used only for the first translation

• Migration plan
  • Validate framework (Summer)
  • Migrate project by project (LHC shutdown)
  • Phase out CMT (still used in old versions)
From CMT to CMake

• Compatibility
  • produce same filesystem hierarchy as CMT
  • preserve CMT configuration files
• Translation tool
  • Python script to analyze and translate configurations
  • used only for the first translation
• Migration plan
  • Validate framework (Summer)
  • Migrate project by project (LHC shutdown)
  • Phase out CMT (still used in old versions)
From CMT to CMake (2)

# package GaudiUtils
version v4r0
branches GaudiUtils src cmt doc

# dependencies
use GaudiKernel *
use ROOT * LCG_Interfaces
use AIDA * LCG_Interfaces -no_auto_imports
use Boost * LCG_Interfaces -no_auto_imports
use Reflex * LCG_Interfaces -no_auto_imports
use uuid * LCG_Interfaces -no_auto_imports
use XercesC * LCG_Interfaces -no_auto_imports

# own includes
apply_pattern install_more_includes more=GaudiUtils

# constituents
library GaudiUtilsLib Lib/*.cpp 
   -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib

library GaudiUtils component/*.cpp 
   -import=Boost -import=Reflex 
   -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils

# local settings
private
   macro_append ROOT_linkopts " -lHist -lXMLIO 
   macro_append Boost_linkopts " $(Boost_linkopts_date_time) 
end_private
# From CMT to CMake (2)

```bash
package GaudiUtils
version v4r0
# ====== structure ===============
branches GaudiUtils src cmt doc
# ====== dependencies ============
use GaudiKernel *
use ROOT     * LCG_Interfaces
use AIDA     * LCG_Interfaces -no_auto_imports
use Boost    * LCG_Interfaces -no_auto_imports
use Reflex   * LCG_Interfaces -no_auto_imports
use uuid     * LCG_Interfaces -no_auto_imports
use XercesC  * LCG_Interfaces -no_auto_imports
# ====== own includes ===========
apply_pattern install_more_includes more=GaudiUtils
# ====== constituents ===========
library GaudiUtilsLib Lib/*.cpp 
   -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib
# ====== constituents ===========
library GaudiUtils component/*.cpp 
   -import=Boost -import=Reflex 
   -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils
# ====== local settings =========
private
  macro_append ROOT_linkopts " -lHist -lXMLIO "
  macro_append Boost_linkopts " $(Boost_linkopts_date_time) "
end_private
```

```bash
gaudi_subdir(GaudiUtils v4r0)
```
From CMT to CMake (2)

# package GaudiUtils
version v4r0
# structure
branches GaudiUtils src cmt doc
# dependencies
use GaudiKernel *
use AIDA * LCG_Interfaces -no_auto_imports
use Boost * LCG_Interfaces -no_auto_imports
use Reflex * LCG_Interfaces -no_auto_imports
use uuid * LCG_Interfaces -no_auto_imports
use XercesC * LCG_Interfaces -no_auto_imports
# own includes
apply_pattern install_more_includes more=GaudiUtils
# constituents
library GaudiUtilsLib Lib/*.cpp \  
   -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib
# constituents
library GaudiUtils component/*.cpp \  
   -import=Boost -import=Reflex \  
   -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils
# local settings
private
  macro_append ROOT_linkopts " -lHist -lXMLIO "
  macro_append Boost_linkopts " $(Boost_linkopts_date_time) "
end_private

gaudi_subdir(GaudiUtils v4r0)

# depends_on_subdirs(GaudiKernel)
# From CMT to CMake (2)

```plaintext
# -------------------------------------
package GaudiUtils
version v4r0
# ===== structure ================
branches GaudiUtils src cmt doc
# ===== dependencies ===========
use GaudiKernel *
use ROOT   * LCG_Interfaces
use AIDA   * LCG_Interfaces -no_auto_imports
use Boost  * LCG_Interfaces -no_auto_imports
use Reflex * LCG_Interfaces -no_auto_imports
use uuid   * LCG_Interfaces -no_auto_imports
use XercesC* LCG_Interfaces -no_auto_imports
# ===== own includes ===========
apply_pattern install_more_includes more=GaudiUtils
# ===== constituents =========
library GaudiUtilsLib Lib/*.cpp \
   -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib
# ===== constituents =========
library GaudiUtils component/*.cpp \
   -import=Boost -import=Reflex \
   -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils
# ===== local settings =========
private
  macro_append ROOT_linkopts " -lHist -lXMLIO "
  macro_append Boost_linkopts " $(Boost_linkopts_date_time) "
end_private
```
# From CMT to CMake (2)

```plaintext
# package GaudiUtils
version v4r0

# branches GaudiUtils src cmt doc

# use dependencies
use GaudiKernel *
use ROOT * LCG_Interfaces
use AIDA * LCG_Interfaces -no_auto_imports
use Boost * LCG_Interfaces -no_auto_imports
use Reflex * LCG_Interfaces -no_auto_imports
use uuid * LCG_Interfaces -no_auto_imports
use XercesC * LCG_Interfaces -no_auto_imports

# apply_pattern install_more_includes more=GaudiUtils

# library GaudiUtilsLib Lib/*.cpp
# -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib

# library GaudiUtils component/*.cpp
# -import=Boost -import=Reflex 
# -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils

# local settings
private
macro_append ROOT_linkopts " -lHist -lXMLIO 
macro_append Boost_linkopts " $(Boost_linkopts_date_time) 
end_private

# gaudi_subdir(GaudiUtils v4r0)

# depends_on_subdirs(GaudiKernel)

find_package(ROOT COMPONENTS RIO Hist XMLIO)
find_package(AIDA)
find_package(Boost COMPONENTS date_time)
find_package(uuid)
find_package(XercesC)

# libraries

gaudi_add_library(GaudiUtilsLib Lib/*.cpp
  LINK_LIBRARIES GaudiKernel Boost ROOT
  INCLUDE_DIRS AIDA Boost ROOT
  PUBLIC_HEADERS GaudiUtils)

# gaudi_add_module(
  GaudiUtils component/*.cpp
  LINK_LIBRARIES GaudiUtilsLib uuid XercesC
  INCLUDE_DIRS uuid XercesC)
```
From CMT to CMake (2)

```plaintext
# Package GaudiUtils
version v4r0

# Structure branches GaudiUtils src cmt doc

# Dependencies
use GaudiKernel *
use ROOT * LCG_Interfaces
use AIDA * LCG_Interfaces -no_auto_imports
use Boost * LCG_Interfaces -no_auto_imports
use Reflex * LCG_Interfaces -no_auto_imports
use uuid * LCG_Interfaces -no_auto_imports
use XercesC * LCG_Interfaces -no_auto_imports

# Own Includes
apply_pattern install_more_includes more=GaudiUtils

# Constituents
library GaudiUtilsLib Lib/*.cpp
   -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib

library GaudiUtils component/*.cpp
   -import=Boost -import=Reflex
   -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils

# Local Settings
private
   macro_append ROOT_linkopts " -lHist -lXMLIO ">
   macro_append Boost_linkopts " $(Boost_linkopts_date_time) ">
end_private

# Subdir
gaudi_subdir(GaudiUtils v4r0)

depends_on_subdirs(GaudiKernel)

find_package(ROOT COMPONENTS RIO Hist XMLIO)
find_package(AIDA)
find_package(Boost COMPONENTS date_time)
find_package(uuid)
find_package(XercesC)

# Libraries
gaudi_add_library(GaudiUtilsLib Lib/*/cpp
   LINK_LIBRARIES GaudiKernel Boost ROOT
   INCLUDE_DIRS AIDA Boost ROOT
   PUBLIC_HEADERS GaudiUtils
   gaudi_add_module(GaudiUtils component/*.cpp
      LINK_LIBRARIES GaudiUtilsLib uuid XercesC
      INCLUDE_DIRS uuid XercesC)
```

# Introduction

# Requirements

# Design

# Implementation

# Conclusions
From CMT to CMake (2)

# package GaudiUtils
version v4r0

# branches GaudiUtils src cmt doc

# use GaudiKernel *
use ROOT * LCG_Interfaces
use AIDA * LCG_Interfaces -no_auto_imports
use Reflex * LCG_Interfaces -no_auto_imports
use uuid * LCG_Interfaces -no_auto_imports
use XercesC * LCG_Interfaces -no_auto_imports

# apply_pattern install_more_includes more=GaudiUtils

# library GaudiUtilsLib Lib/*.cpp\
  -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib

# gaudi_add_library(GaudiUtilsLib Lib/*.cpp LINK_LIBRARIES Boost ROOT INCLUDE_DIRS AIDA Boost ROOT PUBLIC_HEADERS GaudiUtils)

# gaudi_add_module(GaudiUtils component/*.cpp LINK_LIBRARIES uuid XercesC INCLUDE_DIRS uuid XercesC)
Outline

Introduction

Requirements

Design

Implementation

Conclusions
Conclusions

- CMT is a valid product, but with limits
  - CMake is not meant to address our use case…
  - … but it is powerful enough to be adapted
  - Developed a CMake-based build framework
    - can replace CMT in LHCb use
    - better performance
    - will be adopted by after some more validation
Conclusions

- CMT is a valid product, but with limits
- CMake is not meant to address our use case...
  - ... but it is powerful enough to be adapted
- Developed a CMake-based build framework
  - can replace CMT in LHCb use
  - better performance
  - will be adopted by after some more validation
Conclusions

- CMT is a valid product, but with limits
- CMake is not meant to address our use case...
- ... but it is powerful enough to be adapted
- Developed a CMake-based build framework
  - can replace CMT in LHCb use
  - better performance
  - will be adopted by after some more validation
Conclusions

• CMT is a valid product, but with limits
• CMake is not meant to address our use case…
• … but it is powerful enough to be adapted
• Developed a CMake-based build framework
  • can replace CMT in LHCb use
  • better performance
  • will be adopted by after some more validation