CMake Tutorial

- 1 – Introduction to CMake
- 2 – Using CMake for the ILC Software
- 3 – ILCInstall with CMake

Jan Engels

DESY

20th September 2007

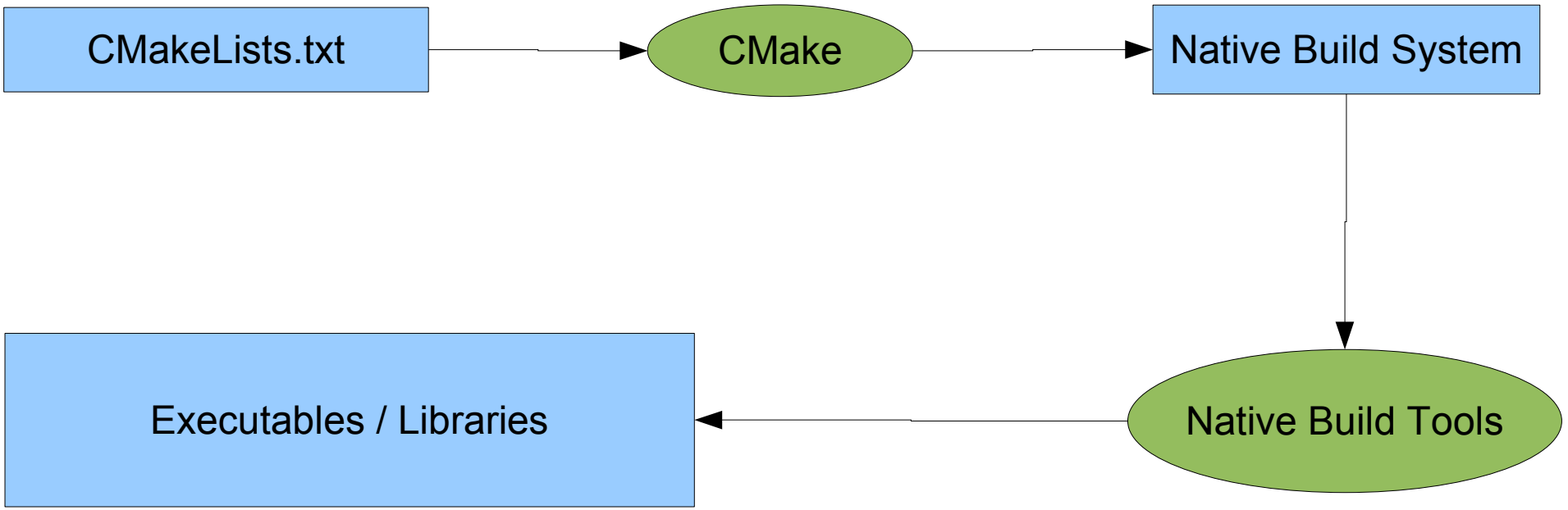
What is CMake

- CMake:
  - Generates native build environments
    - UNIX/Linux -> Makefiles
    - Windows -> VS Projects/Workspaces
    - Apple -> Xcode
  - Open-Source :)
  - Cross-Platform

CMake Features

- CMake has a lot of nice features:
  - Manage complex, large build environments (KDE4)
  - Very Flexible & Extensible
    - Support for Macros
    - Modules for finding/configuring software (bunch of modules already available)
    - Extend CMake for new platforms and languages
    - Create custom targets/commands
    - Run external programs
  - Very simple, intuitive syntax
  - Support for regular expressions (*nix style)
  - Support for “In-Source” and “Out-of-Source” builds
  - Cross Compiling
  - Integrated Testing & Packaging (Ctest, CPack)

Build-System Generator



CMake Basic Concepts

- **CmakeLists.txt**
  - Input text files that contain the project parameters and describe the flow control of the build process in simple CMake language.
- **CMake Modules**
  - Special cmake file written for the purpose of finding a certain piece of software and to set it's libraries, include files and definitions into appropriate variables so that they can be used in the build process of another project. (e.g. FindJava.cmake, FindZLIB.cmake, FindQt4.cmake)

CMake Basic Concepts

- The **Source Tree** contains:
  - CMake input files (CmakeLists.txt)
  - Program source files (hello.cc)
  - Program header files (hello.h)
- The **Binary Tree** contains:
  - Native build system files (*Makefiles*)
  - Output from build process:
    - Libraries
    - Executables
    - Any other build generated file
- Source and Binary trees may be:
  - In the same directory (**in-source** build)
  - In different directories (**out-of-source** build)

CMake Basic Concepts

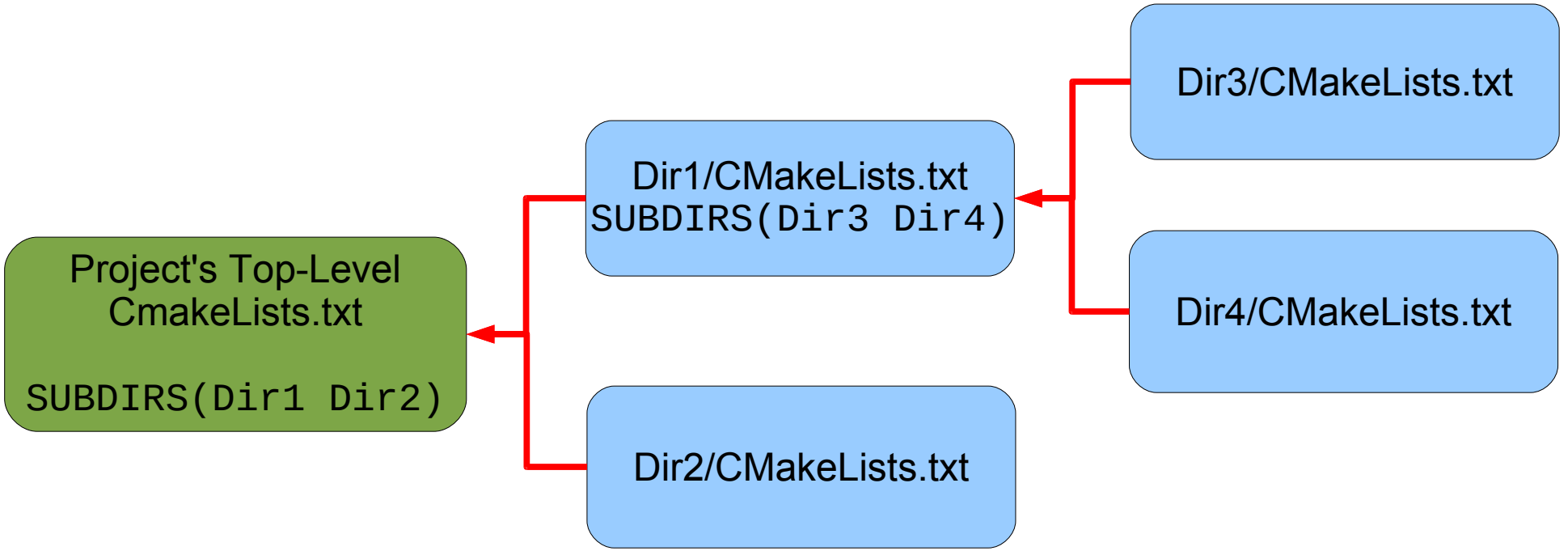
- CMAKE_MODULE_PATH
  - Path to where the CMake modules are located
- CMAKE_INSTALL_PREFIX
  - Where to put files when calling 'make install'
- CMAKE_BUILD_TYPE
  - Type of build (Debug, Release, ...)
- BUILD_SHARED_LIBS
  - Switch between shared and static libraries

- Variables can be changed directly in the build files (CmakeLists.txt) or through the command line by prefixing a variable's name with '-D':
  - cmake -DBUILD_SHARED_LIBS=OFF
- GUI also available: ccmake

CMake Cache

- Created in the build tree (*CMakeCache.txt*)
- Contains Entries VAR:TYPE=VALUE
- Populated/Updated during configuration phase
- Speeds up build process
- Can be initialized with *cmake -C <file>*
- GUI can be used to change values
- There should be no need to edit it manually!!

Source Tree Structure



- Subdirectories added with SUBDIRS/ADD_SUBDIRECTORY
- Child inherits from parent (feature that is lacking in traditional Makefiles)
- Order of processing: Dir1;Dir3;Dir4;Dir2 (When CMake finds a SUBDIR command it stops processing the current file immediately and goes down the tree branch)

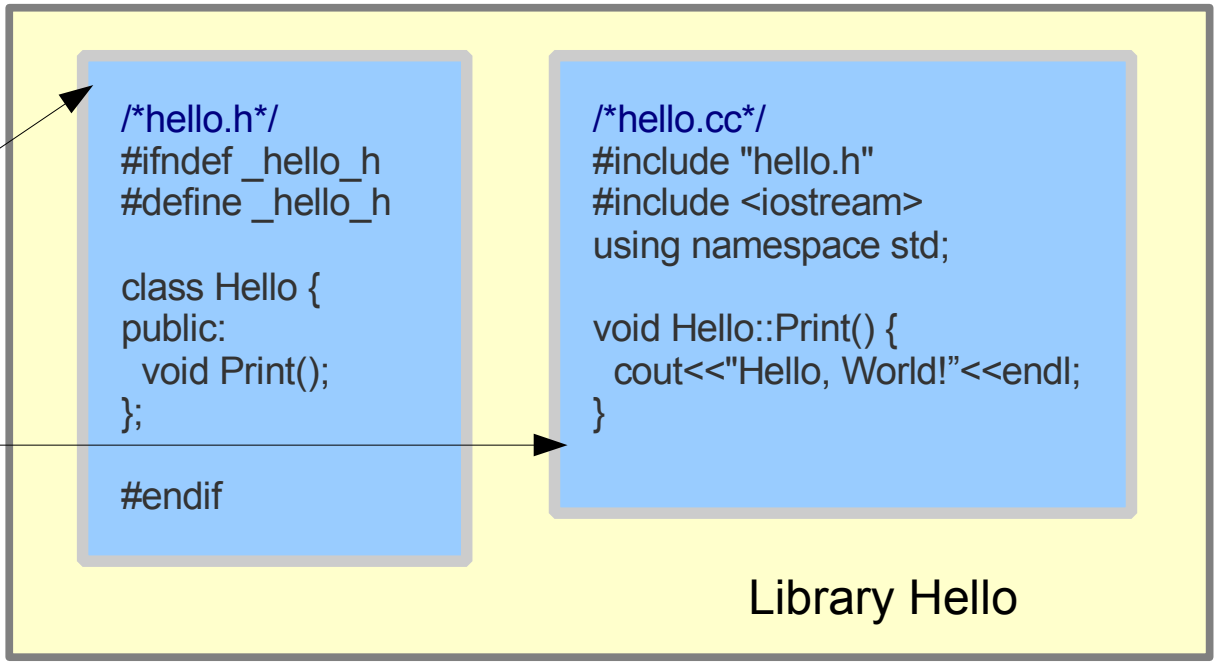
Using CMake

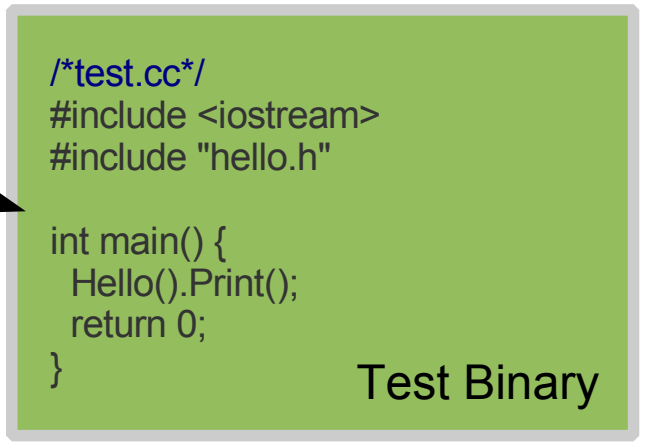
- **Create a build directory** (“out-of-source-build” concept)
  - mkdir build ; cd build
- **Configure** the package for your system:
  - cmake [options] <source_tree>
- **Build** the package:
  - make
- **Install** it:
  - make install
- The last 2 steps can be merged into one (just “make install”)

Similar to Auto Tools

Hello World for CMake

- Top-level project directory:
  - CMakeLists.txt
  - *Sub-directory* Hello:
    - CMakeLists.txt
    - hello.h
    - hello.cc
  - *Sub-directory* Test:
    - CMakeLists.txt
    - test.cc





Hello World for CMake

```
# Top-Level CmakeLists.txt

PROJECT( HELLO )

ADD_SUBDIRECTORY( Hello )
ADD_SUBDIRECTORY( Test )
```

```
# CmakeLists.txt in Hello dir

# Adds a library called Hello (libHello.a under Linux) from the source file hello.cc
ADD_LIBRARY( Hello hello )
```

```
# CmakeLists.txt in Test dir

# Make sure the compiler can find include files from our Hello library.
INCLUDE_DIRECTORIES(${HELLO_SOURCE_DIR}/Hello)

# Add binary called "helloWorld" that is built from the source file "test.cc".
# The extension is automatically found.
ADD_EXECUTABLE(helloWorld test)

# Link the executable to the Hello library.
TARGET_LINK_LIBRARIES(helloWorld Hello)
```

CmakeLists.txt Files

- Very simple syntax:
  - # This is a comment
  - Commands syntax: COMMAND(arg1 arg2 ...)
  - Lists A;B;C # semi-colon separated values
  - Variables ${VAR}
  - Conditional constructs
    - IF() ... ELSE()/ELSEIF() ... ENDIF()
      - Very useful: IF(APPLE); IF(UNIX); IF(WIN32)
    - WHILE() ... ENDWHILE()
    - FOREACH() ... ENDFOREACH()
  - Regular expressions (check CMake FAQ for details...)

CmakeLists.txt Files

- INCLUDE_DIRECTORIES(“dir1” “dir2” ...)
- AUX_SOURCE_DIRECTORY(“source”)
- ADD_EXECUTABLE
- ADD_LIBRARY
- ADD_CUSTOM_TARGET
- ADD_DEPENDENCIES(target1 t2 t3) target1 depends on t2 and t3
- ADD_DEFINITIONS(“-Wall -ansi -pedantic”)
- TARGET_LINK_LIBRARIES(target-name lib1 lib2 ...) Individual settings for each target
- LINK_LIBRARIES(lib1 lib2 ...) All targets link with the same set of libs
- SET_TARGET_PROPERTIES(...) lots of properties... OUTPUT_NAME, VERSION, ...
- MESSAGE(STATUS|FATAL_ERROR “message”)
- INSTALL(FILES “f1” “f2” “f3” DESTINATION .)
  - DESTINATION relative to ${CMAKE_INSTALL_PREFIX}

Check www.cmake.org -> Documentation

CmakeLists.txt Files

- SET(VAR value [CACHE TYPE DOCSTRING [FORCE]])
- LIST(APPEND|INSERT|LENGTH|GET|REMOVE_ITEM|REMOVE_AT|SORT ...)
- STRING(TOUPPER|TOLOWER|LENGTH|SUBSTRING|REPLACE|REGEX ...)
- SEPARATE_ARGUMENTS(VAR) convert space separated string to list
- FILE(WRITE|READ|APPEND|GLOB|GLOB_RECURSE|REMOVE|MAKE_DIRECTORY ...)
- FIND_FILE
- FIND_LIBRARY
- FIND_PROGRAM
- FIND_PACKAGE
- EXEC_PROGRAM(bin [work_dir] ARGS <..> [OUTPUT_VARIABLE var] [RETURN_VALUE var])
- OPTION(OPTION_VAR "description string" [initial value])

Check www.cmake.org -> Documentation

CMake Tutorial

- 2 – Using CMake for the ILC Software

CMake for the ILC Software

- IMPORTANT:
  - CMake files for the ILC Software were designed, written and tested exclusively for out-of-source builds, therefore we strongly disencourage in-source builds!!
  - A package should be installed first (with 'make install') before it can be used by other packages, thus we also strongly disencourage trying to pass the binary-tree from one package as the "installation directory" to other packages.
- Packages with CMake (build) support:
  - Marlin, MarlinUtil, MarlinReco, CEDViewer, CED, LCIO, GEAR, LCCD, RAIDA, PandoraPFA, LCFIVertex, SiliconDigi, Eutelescope
- CMake modules written for external packages:
  - CLHEP, CERNLIB, CondDBMySQL, GSL, ROOT, JAVA, AIDAJNI

Special variables

- BUILD_WITH=”CLHEP GSL”
  - Tell package to use the libraries, include files and definitions from these packages
- <PKG>_HOME
  - Variable for defining the home path from a pkg
    - Standard CMake Find modules differ slightly from ILC Find modules
      - ILC Find modules require PKG_HOME variable set
        - Enforce version consistency (get rid of setting global environment variables for defining local dependencies)
      - Could instead be called Config<PKG>.cmake

Macros

- MacroLoadPackage.cmake
  - To be able to use a package by using a "Find<PKG>.cmake" module or by using a "<PKG>Config.cmake" file
  - Assumes the PKG_HOME variable is properly set
- MacroCheckDeps.cmake
  - Uses MacroLoadPackage.cmake to check dependencies

Find<PKG>.cmake Modules

- Do the same as <PKG>Config.cmake generated by the cmake build
- Returns variables for using the package
  - **<PKG>_INCLUDE_DIRS**
  - **<PKG>_LIBRARIES**
  - **<PKG>_DEFINITIONS**
- Using the MacroLoadPackage this is automatically done for you

BuildSetup.cmake

- Script for pre-caching variables/options
  - SET(VAR “value” CACHE TYPE “description” FORCE)
- Easy way to change build parameters without having to pass the every time on the cmd line
- Use simple steps to build a package:
  - **mkdir build ; cd build**
  - **cmake -C ../BuildSetup.cmake ..**
  - **make install**
- Still possible to override options on the cmd line

BuildSetup.cmake

- Can use more than one -C option:
  - cmake -C ../BuildSetup.cmake -C ~/ILCSoft.cmake
  - Next file overwrites values from previous file
  - Useful for overwriting paths defined in a 'more global' file
    - CMake just ignores redundant variables from global file
    - ILCInstall generates a global file called ILCSoft.cmake
- Check /afs/desy.de/group/it/ilcsoft/v01-01/ILCSoft.cmake as an example

Adapting your processor to CMake

- Copy from $Marlin/examples/mymarlin
  - **CmakeLists.txt**
    - change the project name and add missing dependencies (default are Marlin;LCIO)
  - **mymarlinConfig.cmake.in**
    - rename to <MyProcessor>Config.cmake
  - **BuildSetup.cmake**
    - change this according to your system setup
  - **cmake_uninstall.cmake.in**
    - Script for generating a 'make uninstall' target
      - No changes needed!

CmakeLists.txt template

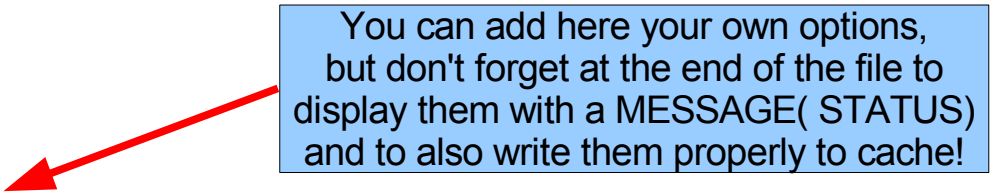
```
# cmake file for building Marlin example Package

# CMake compatibility issues: don't modify this, please!
CMAKE_MINIMUM_REQUIRED( VERSION 2.4.6 )
MARK_AS_ADVANCED(CMAKE_BACKWARDS_COMPATIBILITY)
# allow more human readable "if then else" constructs
SET( CMAKE_ALLOW_LOOSE_LOOP_CONSTRUCTS TRUE )

# User section
PROJECT( mymarlin )

# project version
SET( ${PROJECT_NAME}_MAJOR_VERSION 0 )
SET( ${PROJECT_NAME}_MINOR_VERSION 1 )
SET( ${PROJECT_NAME}_PATCH_LEVEL 0 )

# project options
OPTION( BUILD_SHARED_LIBS "Set to OFF to build static libraries" ON )
OPTION( INSTALL_DOC "Set to OFF to skip build/install Documentation" ON )
# project dependencies e.g. SET( ${PROJECT_NAME}_DEPENDS "Marlin MarlinUtil LCIO GEAR CLHEP GSL" )
SET( ${PROJECT_NAME}_DEPENDS "Marlin LCIO" )
# set default cmake build type to RelWithDebInfo (None Debug Release RelWithDebInfo MinSizeRel)
IF( NOT CMAKE_BUILD_TYPE )
   SET( CMAKE_BUILD_TYPE "RelWithDebInfo" )
ENDIF()
# set default install prefix to project root directory
IF( CMAKE_INSTALL_PREFIX STREQUAL "/usr/local" )
   SET( CMAKE_INSTALL_PREFIX "${PROJECT_SOURCE_DIR}" )
ENDIF()
```



CmakeLists.txt template

```
#include directories
INCLUDE_DIRECTORIES( "${PROJECT_SOURCE_DIR}/include" )
# install include files
INSTALL( DIRECTORY "${PROJECT_SOURCE_DIR}/include"
       DESTINATION . PATTERN "*~" EXCLUDE PATTERN "*CVS*" EXCLUDE )

# require proper c++
ADD_DEFINITIONS( "-Wall -ansi -pedantic" )
# add debug definitions
#IF( CMAKE_BUILD_TYPE STREQUAL "Debug" OR
#    CMAKE_BUILD_TYPE STREQUAL "RelWithDebInfo" )
#    ADD_DEFINITIONS( "-DDEBUG" )
#ENDIF()

# get list of all source files
AUX_SOURCE_DIRECTORY( src library_sources )

( .... )

# DEPENDENCIES: this code has to be placed before adding any library or
# executable so that these are linked properly against the dependencies
IF( DEFINED ${PROJECT_NAME}_DEPENDS OR DEFINED BUILD_WITH OR DEFINED LINK_WITH )
   # load macro
   IF( NOT EXISTS "${CMAKE_MODULE_PATH}/MacroCheckDeps.cmake" )
      MESSAGE( FATAL_ERROR
         "\nSorry, could not find MacroCheckDeps.cmake...\n"
         "Please set CMAKE_MODULE_PATH correctly with: "
         "cmake -DCMAKE_MODULE_PATH=<path_to_cmake_modules>" )
   ENDIF()
   INCLUDE( "${CMAKE_MODULE_PATH}/MacroCheckDeps.cmake" )
   CHECK_DEPS()
ENDIF()
```

Include directories here

Add your Debug definitions here!

If you have more sources you should add them here (see for ex. LCFIVertex CMakeLists.txt)

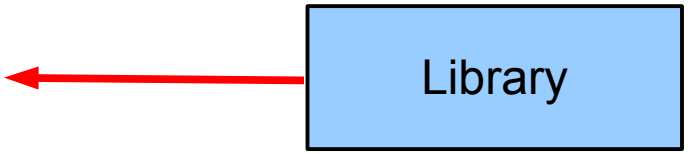
Dependencies are checked here!

CmakeLists.txt template

```
# LIBRARY
ADD_LIBRARY( lib_${PROJECT_NAME} ${library_sources} )
# create symbolic lib target for calling target lib_XXX
ADD_CUSTOM_TARGET( lib DEPENDS lib_${PROJECT_NAME} )
# change lib_target properties
SET_TARGET_PROPERTIES( lib_${PROJECT_NAME} PROPERTIES
   # create *nix style library versions + symbolic links
   VERSION ${${PROJECT_NAME}_VERSION}
   SOVERSION ${${PROJECT_NAME}_SOVERSION}
   # allow creating static and shared libs without conflicts
   CLEAN_DIRECT_OUTPUT 1
   # avoid conflicts between library and binary target names
   OUTPUT_NAME ${PROJECT_NAME} )

# install library
INSTALL( TARGETS lib_${PROJECT_NAME} DESTINATION lib PERMISSIONS
      OWNER_READ OWNER_WRITE OWNER_EXECUTE
      GROUP_READ GROUP_EXECUTE
      WORLD_READ WORLD_EXECUTE )

# create uninstall configuration file
CONFIGURE_FILE( "${PROJECT_SOURCE_DIR}/cmake_uninstall.cmake.in"
            "${PROJECT_BINARY_DIR}/cmake_uninstall.cmake"
            IMMEDIATE @ONLY )
# create uninstall target
ADD_CUSTOM_TARGET( uninstall
  "${CMAKE_COMMAND}" -P "${PROJECT_BINARY_DIR}/cmake_uninstall.cmake" )
# create configuration file from .in file
CONFIGURE_FILE( "${PROJECT_SOURCE_DIR}/${PROJECT_NAME}Config.cmake.in"
            "${PROJECT_BINARY_DIR}/${PROJECT_NAME}Config.cmake" @ONLY )
# install configuration file
INSTALL( FILES "${PROJECT_BINARY_DIR}/${PROJECT_NAME}Config.cmake" DESTINATION . )
```



CmakeLists.txt template

```
# display status message for important variables
MESSAGE( STATUS )
MESSAGE( STATUS "-------------------------------------------------------------------------------" )
MESSAGE( STATUS "BUILD_SHARED_LIBS = ${BUILD_SHARED_LIBS}" )
MESSAGE( STATUS "CMAKE_INSTALL_PREFIX = ${CMAKE_INSTALL_PREFIX}" )
MESSAGE( STATUS "CMAKE_BUILD_TYPE = ${CMAKE_BUILD_TYPE}" )
MESSAGE( STATUS "CMAKE_MODULE_PATH = ${CMAKE_MODULE_PATH}" )
MESSAGE( STATUS "${PROJECT_NAME}_DEPENDS = \"${${PROJECT_NAME}_DEPENDS}\"" )
MESSAGE( STATUS "BUILD_WITH = \"${BUILD_WITH}\"" )
MESSAGE( STATUS "INSTALL_DOC = ${INSTALL_DOC}" )
MESSAGE( STATUS "Change a value with: cmake -D<Variable>=<Value>" )
MESSAGE( STATUS "-------------------------------------------------------------------------------" )
MESSAGE( STATUS )

# force some variables that could be defined in the command line to be written to cache
SET( BUILD_SHARED_LIBS "${BUILD_SHARED_LIBS}" CACHE BOOL
   "Set to OFF to build static libraries" FORCE )
SET( CMAKE_INSTALL_PREFIX "${CMAKE_INSTALL_PREFIX}" CACHE PATH
   "Where to install ${PROJECT_NAME}" FORCE )
SET( CMAKE_BUILD_TYPE "${CMAKE_BUILD_TYPE}" CACHE STRING
   "Choose the type of build, options are: None Debug Release RelWithDebInfo MinSizeRel." FORCE )
SET( CMAKE_MODULE_PATH "${CMAKE_MODULE_PATH}" CACHE PATH
   "Path to custom CMake Modules" FORCE )
SET( INSTALL_DOC "${INSTALL_DOC}" CACHE BOOL
   "Set to OFF to skip build/install Documentation" FORCE )

# export build settings
INCLUDE( CMakeExportBuildSettings )
CMAKE_EXPORT_BUILD_SETTINGS( "${PROJECT_NAME}BuildSettings.cmake" )
# export library dependencies (keep this as the last line in the file)
EXPORT_LIBRARY_DEPENDENCIES( "${PROJECT_NAME}LibDeps.cmake" )
```

Here you can display your own project options

And here you should also add your own project options to be properly written to cache!

Loading Processors in Marlin

- **MARLIN_DLL** environment variable
  - $ export MARLIN_DLL=”/path1/lib1.so:/path2/lib2.so:$MARLIN_DLL”
  - $ ./Marlin steer.xml
  - Using ILCInstall this information is already added to the generated file **build_env.sh** for the processors found in the config file
- Linking Marlin with other shared libraries
  - Add to your Marlin **BuildSetup.cmake**
    - SET(**LINK_WITH** "MarlinReco CEDViewer” CACHE STRING “Link Marlin with these optional packages" FORCE)
  - Or pass it on the command line:
    - $ cmake -C ../BuildSetup.cmake
      - -DLINK_WITH="mymarlin PandoraPFA"
      - -Dmymarlin_HOME="path_to_mymarlin
      - -DPandoraPFA_HOME="path_to_pandora" ..

Loading Processors in Marlin

- Linking static libraries (Only works under linux!)
  - -D**LINK_STATIC_WHOLE_LIBS**="path_to_library/libMyprocessor.a"
  - Library gets fully included into the Marlin binary
  - For more than one library:
    - -DLINK_STATIC_WHOLE_LIBS="/path1/lib1.a;/path2/lib2.a"

CMake Tutorial

- 3 – ILCInstall with cmake

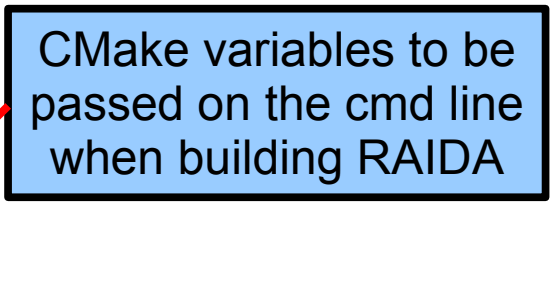
ILCInstall

```
ilcsoft = ILCSoft("/data/ilcsoft")
ilcsoft.useCMake = True

# python variable for referring the ILC Home directory
ilcPath = "/afs/desy.de/group/it/ilcsoft/"

# install RAIDA v01-03
ilcsoft.install( RAIDA( "v01-03" ))
# example for setting cmake variables (“ON”/”OFF” is equivalent to 1/0)
ilcsoft.module( “RAIDA” ).envcmake[“BUILD_RAIDA_EXAMPLE”] = “ON”
ilcsoft.module( “RAIDA” ).envcmake[“RAIDA_DEBUG_VERBOSE_FACTORY”] = 1

# use ROOT at: /afs/desy.de/group/it/ilcsoft/root/5.08.00
ilcsoft.link( ROOT( ilcPath + "root/5.08.00" ))
# use CMakeModules at: /afs/desy.de/group/it/ilcsoft/CMakeModules/v01-00
ilcsoft.use( CMakeModules( ilcPath + "CMakeModules/v01-00" ))
# use CMake at: /afs/desy.de/group/it/ilcsoft/CMake/2.4.6
ilcsoft.use( CMake( ilcPath + "CMake/2.4.6" ))
# End of configuration file
```



ILCInstall

- After creating the file call:
  - ilcsoft-install RAIDA.cfg (display summary)
  - ilcsoft-install RAIDA.cfg -i (install RAIDA)
- ILCSoft.cmake is generated by installation script
  - Placed in the root directory of installation
  - Contains paths for all packages defined in cfg file
    - Only the ones that are supported by the cmake modules!

ILCInstall (Dev)

- Under the directory "releases" you find the AFS reference-installation configuration files
  - Copy one of them:
    - Only install packages you want to work on
      - ilcsoft.module("RAIDA").download.type="ccvssh"
      - ilcsoft.module("RAIDA").download.username="engels"
    - Change the package dependencies install -> link
      - ilcsoft.link(ROOT("/data/myILCSoftware/root/5.08.00"))
    - Set needed options
      - ilcsoft.module("RAIDA").envcmake["BUILD_RAIDA_EXAMPLE"] = 1
      - ilcsoft.module("RAIDA").envcmake["RAIDA_DEBUG_VERBOSE_FACTORY"] = 1

References

- http://ilcsoft.desy.de -> General Documentation -> “How to use the CMake building tool for the ILC Software”
- http://www.cmake.org
  - Documentation
  - FAQ
- **Mastering CMake**
  - Ken Martin, Bill Hoffman
  - Published by Kitware, Inc.
  - ISBN: 1-930934-16-5
- This talk: http://ilcsoft.desy.de -> General Documentation

Thank you!