Dipping your toes into the buildsystem and CMake

or how to be not completely lost





7/29/2013 | Page 2



- Many do not dare take on a build system bug
- Having a rough idea helps being not completely lost
- You can actually do quite a lot with it

7/29/2013 | Page 3



- We use Cmake
 - Cross platform, compiler independent config files
 - Generates native config files for your compiler
 - You can easily configure by switching stuff on/off

CMake 2.8.5 - Y:/bin/bug/MITK-superbuild/MITK-build	
File Tools Options Help	
Where is the source code: C:/home/goch/mitk/sources/woco/MITK	Browse Source
where is the source code: C:/nome/goci/imitk/sources/woco/imitik	blowse source
Where to build the binaries: Y:/bin/bug/MITK-superbuild/MITK-build	Browse Build
Search:	🕅 Grouped 📝 Advanced 🔂 Add Entry
Name	Value
BLUEBERRY BUILD ALL PLUGINS	
BLUEBERRY_BUILD_TESTING	
BLUEBERRY_BUILD_org.blueberry.compat	
BLUEBERRY_BUILD_org.blueberry.core.commands	
BLUEBERRY_BUILD_org.blueberry.core.expressions	
BLUEBERRY_BUILD_org.blueberry.core.jobs	
BLUEBERRY_BUILD_org.blueberry.core.runtime	
BLUEBERRY_BUILD_org.blueberry.osgi BLUEBERRY_BUILD_org.blueberry.solstice.common	
BLUEBERRY BUILD org.blueberry.test	
BLUEBERRY_BUILD_org.blueberry.ui	
BLUEBERRY BUILD org.blueberry.ui.gt	
BLUEBERRY_BUILD_org.blueberry.ui.qt.help	
BLUEBERRY_BUILD_org.blueberry.ui.qt.log	
BLUEBERRY_BUILD_org.blueberry.ui.qt.objectinspector	a 🖬 a shekara ka sh
BLUEBERRY_BUILD_org.blueberry.uitest	
BLUEBERRY_DEBUG_SMARTPOINTER	
BLUEBERRY_DOCS_OUTPUT_DIR	Y:/bin/bug/MITK-superbuild/MITK-build/BlueBerry/Documentation
BLUEBERRY_DOC_TOOLS_DIR BLUEBERRY_ENABLE_GUI_TESTING	
BLUEBERRY_STATIC	
BLUEBERRY USE OT HELP	
BOOST ROOT	Y:/bin/bug/MITK-superbuild/Boost-src
BUILD_SHARED_LIBS	
BUILD_TESTING	✓
Press Configure to update and display new values in red, then press Generate to generate selected build files.	
Configure Generate Current Generator: Visual Studio 9 2008 Win64	

7/29/2013 | Page 4 Basic CMake I



- **SET**(VAR [VALUE])
- MESSAGE("Value of VAR: \${VAR}")
- Simple lists: SET(VAR \${VAR} "AnotherValue")
- **ADD_EXECUTABLE**(MyApplication source1.cpp source2.cpp)
- ADD_LIBRARY(MyLibrary libsource1.cpp ...)
- TARGET_LINK_LIBRARIES(MyApplication MyLibrary AnotherLibrary)
- **INCLUDE_DIRECTORIES**(<path to c++ headers>)





 Defined by MACRO(NAME parameter1 parameter2 ...)

ENDMACRO()

. . .

```
FUNCTION(NAME parameters)
ENDFUNCTION()
```

• In MITK there is

MITK_CREATE_MODULE() MITK_CREATE_PLUGIN()

Defined in mitk/CMake/*.cmake







- Each directory has a main file "CMakeLists.txt"
- **INCLUDE**(secondCMakeScript.cmake): direct parsing of the file in-place (as in C++)
- ADD_SUBDIRECTORY(directory): "directory" has to include another CMakeLists.txt. Used for sub-projects.



- Most interesting directories:
 - /CMake/ /BlueBerry/CMake/

These contain functions and macros used in the rest of MITK

/CMakeExternals/

If it is due to some external toolkit

- If it concerns installers
 - Grep ,CPack'
- It it concerns the applications
 - /Applications/<YourNameHere>/CMakeLists.txt
 - /Applications/<YourNameHere>/*.cmake

7/29/2013 | Page 8



Easy example:

```
foreach(mitk app ${MITK APPS})
 # extract target dir and option name
 string(REPLACE "^^" "\\;" target info ${mitk app})
 set(target info list ${target info})
 list(GET target_info_list 0 target dir)
 list(GET target_info_list 1 option_name)
 # check if the application is enabled
 if (${option name} OR MITK BUILD ALL APPS)
   # check whether application specific configuration files will be used
   if (use project cpack)
     # use files if they exist
     if (EXISTS "${CMAKE CURRENT SOURCE DIR}/Applications/${target dir}/CPackOptions.cmake")
       include ("${CMAKE CURRENT SOURCE DIR}/Applications/${target dir}/CPackOptions.cmake")
     endif()
     if (EXISTS "${PROJECT SOURCE DIR}/Applications/${target dir}/CPackConfig.cmake.in")
       set(CPACK PROJECT CONFIG FILE "${PROJECT BINARY DIR}/Applications/${target dir}/CPackConfig.cmake")
       configure file(${PROJECT_SOURCE_DIR}/Applications/${target_dir}/CPackConfig.cmake.in
                      ${CPACK PROJECT CONFIG FILE} @ONLY)
       set(use default config OFF)
     endif()
   endif()
  # add link to the list
 list(APPEND CPACK_CREATE_DESKTOP_LINKS "${target dir}")
 endif()
endforeach()
```

7/29/2013 | Page 9 | Someth



- The most important tools:
 - grep [Bash]
 - To find where the macro is called/where a variable is filled/where the error message originates
 - message() [CMake]
 - To debug the content of variables, will show during cmake run
- Common Sense:
 - Just because you can now tinker with the build system does not always mean you should