

CMake tutorial

and its friends CPack, CTest and CDash

Eric NOULARD - eric.noulard@gmail.com



<http://www.cmake.org>

February, 8th 2012

This presentation is licensed



<http://creativecommons.org/licenses/by-sa/3.0/us/>



Thanks to...

- Kitware for making a really nice set of tools and making them open-source
- the CMake mailing list for its friendliness and its more than valuable source of information
- CMake developers for their tolerance when I break the dashboard or mess-up with the git workflow,
- CPack users for their patience when things don't work as they should expect
- Alan, Alex, Bill, Brad, Clint, David, Eike, Julien, Mathieu, Michael & Michael, and many more...
- My son Louis for the nice CPack 3D logo done with Blender.
- and...Toulibre for hosting this presentation in Toulouse, France.



CMake tool sets

CMake

CMake is a cross-platform build systems generator which makes it easier to build software in a unified manner on a broad set of platforms:



, Windows, MacOS, AIX, IRIX,  , iOS ...

CMake has friends softwares that may be used on their own or together:

- CMake: build system generator
- CPack: package generator
- CTest: systematic test driver
- CDash: a dashboard collector



Outline of Part I: CMake

- 1 **Basic CMake usage**
- 2 **Discovering environment specificities**
 - Handling platform specificities
 - Working with external packages
- 3 **More CMake scripting**
 - Custom commands
 - Generated files
- 4 **Advanced CMake usage**
 - Cross-compiling with CMake
 - Export your project



Outline of Part II: CPack

- 5 CPack: Packaging made easy
- 6 CPack with CMake
- 7 Various package generators



Outline of Part III: CTest and CDash

- 8 **Systematic Testing**
- 9 **CTest submission to CDash**
- 10 **References**



Build what?

Software build system

A software build system is the usage of a [set of] tool[s] for building software applications.

Why do we need that?



Build what?

Software build system

A software build system is the usage of a [set of] tool[s] for building software applications.

Why do we need that?

- because most softwares consist in several parts that need some building to put them together,



Build what?

Software build system

A software build system is the usage of a [set of] tool[s] for building software applications.

Why do we need that?

- because most softwares consist in several parts that need some building to put them together,
- because softwares are written in various languages that may share the same building process,



Build what?

Software build system

A software build system is the usage of a [set of] tool[s] for building software applications.

Why do we need that?

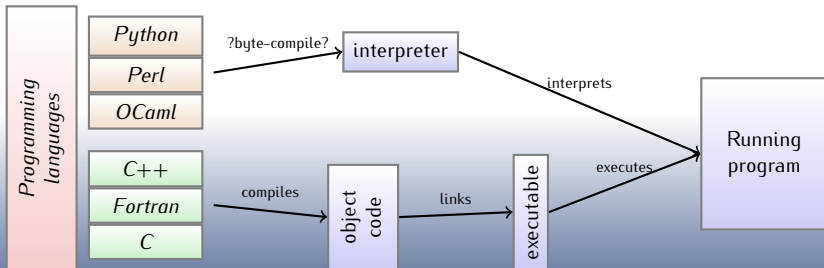
- because most softwares consist in several parts that need some building to put them together,
- because softwares are written in various languages that may share the same building process,
- because we want to build the same software for various computers (PC, Macintosh, Workstation, mobile phones and other PDA, embedded computers) and systems (Windows, Linux, *BSD, other Unices (many), Android, etc...)



Programming languages

Compiled vs interpreted or what?

Building an application requires the use of some programming language: Python, Java, C++, Fortran, C, Go, Tcl/Tk, Ruby, Perl, OCaml,...

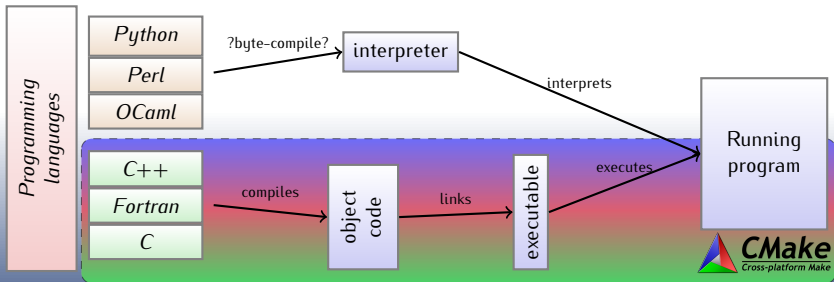




Programming languages

Compiled vs interpreted or what?

Building an application requires the use of some programming language: Python, Java, C++, Fortran, C, Go, Tcl/Tk, Ruby, Perl, OCaml,...





Build systems: several choices

Alternatives

CMake is not the only build system [generator]:

- (portable) hand-written Makefiles, depends on make tool.
- Apache ant <http://ant.apache.org/>, dedicated to Java (almost).
- Portable IDE: Eclipse, Code::Blocks, Geany, NetBeans, ...
- GNU Autotools: Autoconf, Automake, Libtool. Produce makefiles. Needs bourne shell (and M4 macro processor). see e.g. <http://www.gnu.org/software/autoconf/>
- SCons: <http://www.scons.org> only depends on python. Extensible with python.
- ...



Comparisons and [success] stories

Disclaimer

This presentation is biased. I mean totally.

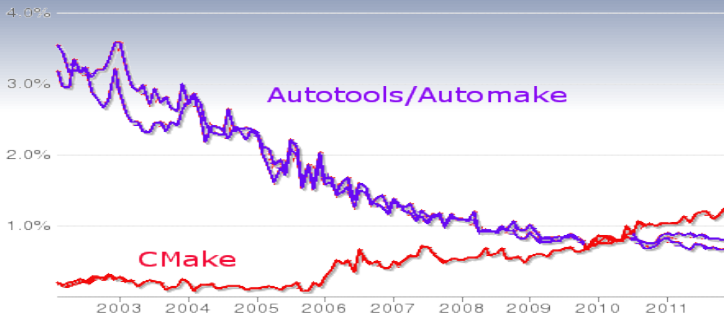
I am a big CMake fan, I'm contributing to CMake, thus I'm not impartial at all. But I will be ready to discuss why CMake is the greatest build system out there :-)

Go and forge your own opinion:

- Bare list: http://en.wikipedia.org/wiki/List_of_build_automation_software
- A comparison: <http://www.scons.org/wiki/SconsVsOtherBuildTools>
- KDE success story (2006): "Why the KDE project switched to CMake – and how" <http://lwn.net/Articles/188693/>



CMake/Auto[conf|make] on Ohloh

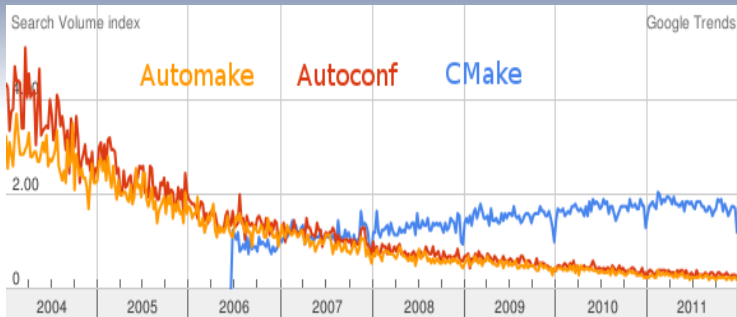


<https://www.ohloh.net/languages/compare>

Language comparison of CMake to automake and autoconf showing the percentage of developers commits that modify a source file of the respective language.



CMake/Auto[conf|make] on Google Trend



<http://www.google.com/trends>

Scale is based on the average worldwide traffic of cmake in all years.



Outline

- 1 **Basic CMake usage**
- 2 **Discovering environment specificities**
 - Handling platform specificities
 - Working with external packages
- 3 **More CMake scripting**
 - Custom commands
 - Generated files
- 4 **Advanced CMake usage**
 - Cross-compiling with CMake
 - Export your project



A build system generator

- CMake is a generator: it generates native build systems files (Makefile, IDE project files, ...),
- CMake scripting language (declarative) is used to describe the build,
- The developer edit `CMakeLists.txt`, invoke CMake but should never edit the generated files,
- CMake may be (automatically) re-invoked by the build system,



The CMake workflow

When do things take place?

CMake is a generator so it does not compile (i.e. build) the sources, the underlying build tool (make, XCode, Code::Blocks...) does.



The CMake workflow

- 1 CMake time: CMake is running & processing `CMakeLists.txt`

When do things take place?

CMake is a generator so it does not compile (i.e. build) the sources, the underlying build tool (make, XCode, Code::Blocks...) does.



The CMake workflow

- 1 CMake time: CMake is running & processing `CMakeLists.txt`
- 2 Build time: the build tool runs and invokes (at least) the compiler

When do things take place?

CMake is a generator so it does not compile (i.e. build) the sources, the underlying build tool (make, XCode, Code::Blocks...) does.



The CMake workflow

- 1 CMake time: CMake is running & processing `CMakeLists.txt`
- 2 Build time: the build tool runs and invokes (at least) the compiler
- 3 Install time: the compiled binaries are installed
i.e. from build area to an install location.

When do things take place?

CMake is a generator so it does not compile (i.e. build) the sources, the underlying build tool (make, XCode, Code::Blocks...) does.



The CMake workflow

- 1 CMake time: CMake is running & processing `CMakeLists.txt`
- 2 Build time: the build tool runs and invokes (at least) the compiler
- 3 Install time: the compiled binaries are installed
i.e. from build area to an install location.
- 4 CPack time: CPack is running for building package

When do things take place?

CMake is a generator so it does not compile (i.e. build) the sources, the underlying build tool (make, XCode, Code::Blocks...) does.



The CMake workflow

- 1 CMake time: CMake is running & processing `CMakeLists.txt`
- 2 Build time: the build tool runs and invokes (at least) the compiler
- 3 Install time: the compiled binaries are installed
i.e. from build area to an install location.
- 4 CPack time: CPack is running for building package
- 5 Package Install time: the package (from previous step) is installed

When do things take place?

CMake is a generator so it does not compile (i.e. build) the sources, the underlying build tool (make, XCode, Code::Blocks...) does.



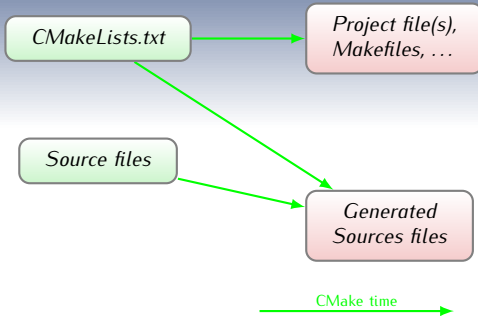
The CMake workflow (pictured)

CMakeLists.txt

Source files

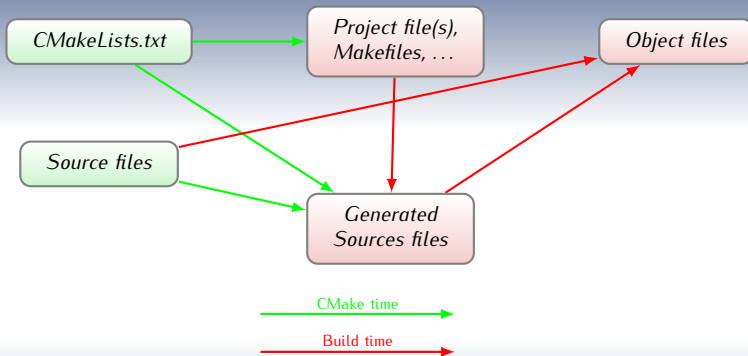


The CMake workflow (pictured)



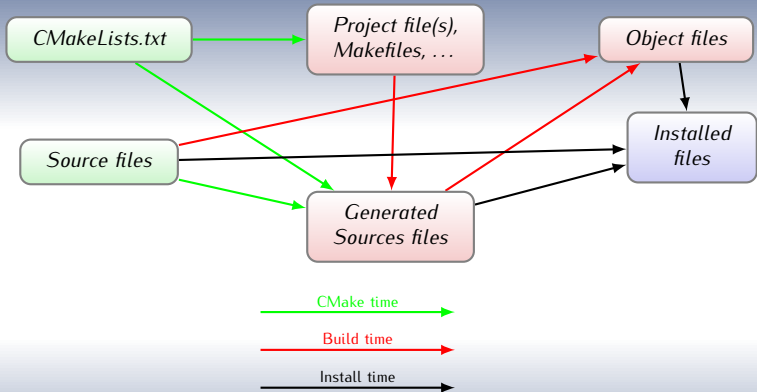


The CMake workflow (pictured)



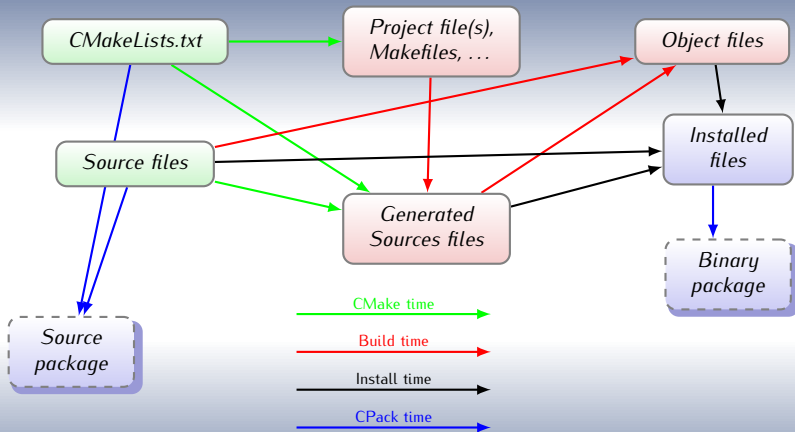


The CMake workflow (pictured)



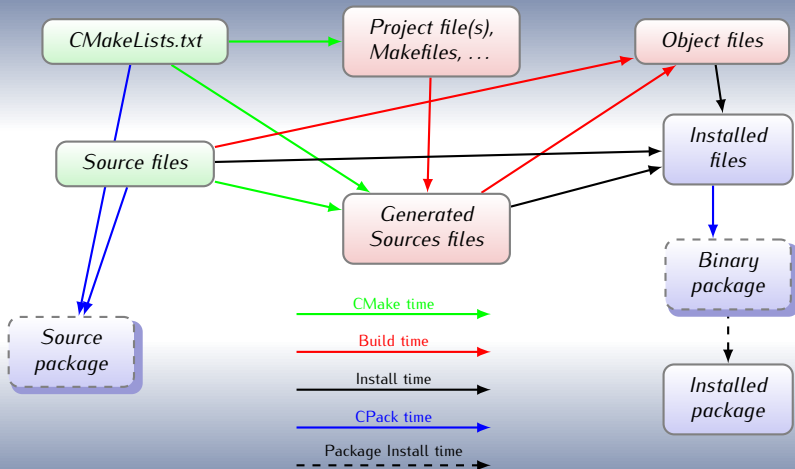


The CMake workflow (pictured)





The CMake workflow (pictured)





Building an executable

Listing 1: Building a simple program

```
1 cmake_minimum_required (VERSION 2.8)
2 # This project use C source code
3 project (TotallyFree C)
4 # build executable using specified
5 # list of source files
6 add_executable (Acrolibre acrolibre.c)
```

CMake scripting language is [mostly] declarative. It has commands which are documented from within CMake:

```
$ cmake --help-command-list | wc -l
96
$ cmake --help-command add_executable
...
add_executable
```

Add an executable to the project using the specified source files.



Builtin documentation

_____ CMake builtin doc for 'project' command _____

```
1 $ cmake --help-command project
2 cmake version 2.8.7.20120121-g751713-dirty
3   project
4     Set a name for the entire project.
5
6     project(<projectname> [languageName1 languageName2 ... ] )
7
8     Sets the name of the project.  Additionally this sets the variables
9     <projectName>_BINARY_DIR and <projectName>_SOURCE_DIR to the
10    respective values.
11
12    Optionally you can specify which languages your project supports.
13    Example languages are CXX (i.e. C++), C, Fortran, etc. By default C
14    and CXX are enabled. E.g. if you do not have a C++ compiler, you can
15    disable the check for it by explicitly listing the languages you want
16    to support, e.g. C. By using the special language "NONE" all checks
17    for any language can be disabled.
```




Generating & building

Building with CMake is easy:

```
----- CMake + Unix Makefile -----
1  $ ls totally-free
2  acrolibre.c CMakeLists.txt
3  $ mkdir build
4  $ cd build
5  $ cmake ../totally-free
6  -- The C compiler identification is GNU 4.6.2
7  -- Check for working C compiler: /usr/bin/gcc
8  -- Check for working C compiler: /usr/bin/gcc -- works
9  ...
10 $ make
11 Scanning dependencies of target Acrolibre
12 [100%] Building C object CMakeFiles/Acrolibre.dir/acrolibre.c.o
13 Linking C executable Acrolibre
14 [100%] Built target Acrolibre
15 $ ./Acrolibre toulibre
```

Source tree vs Build tree

Even the most simple project should never mix-up sources with generated files. CMake supports out-of-source build.



Always build out-of-source

Out-of-source is better

People are lazy (me too) and they think that because building in source is possible and authorizes less typing they can get away with it. In-source build is a BAD choice.

Out-of-source build is always better because:



Always build out-of-source

Out-of-source is better

People are lazy (me too) and they think that because building in source is possible and authorizes less typing they can get away with it. In-source build is a BAD choice.

Out-of-source build is always better because:

- 1 Generated files are separate from manually edited ones (thus you don't have to clutter you favorite VCS ignore files).



Always build out-of-source

Out-of-source is better

People are lazy (me too) and they think that because building in source is possible and authorizes less typing they can get away with it. In-source build is a BAD choice.

Out-of-source build is always better because:

- 1 Generated files are separate from manually edited ones (thus you don't have to clutter you favorite VCS ignore files).
- 2 You can have several build trees for the same source tree



Always build out-of-source

Out-of-source is better

People are lazy (me too) and they think that because building in source is possible and authorizes less typing they can get away with it. In-source build is a BAD choice.

Out-of-source build is always better because:

- 1 Generated files are separate from manually edited ones (thus you don't have to clutter you favorite VCS ignore files).
- 2 You can have several build trees for the same source tree
- 3 This way it's always safe to completely delete the build tree in order to do a clean build



Building program + autonomous library

We now have the following set of files in our source tree:

- `acrolibre.c`, the main C program
- `acrodect.h`, the Acrodect library header
- `acrodect.c`, the Acrodect library source
- `CMakeLists.txt`, the soon to be updated CMake entry file



Building program + autonomous library

Conditional build

We want to keep a version of our program that can be compiled and run without the new Acrodicst library and the new version which uses the library.

We now have the following set of files in our source tree:

- `acrolibre.c`, the main C program
- `acrodicst.h`, the Acrodicst library header
- `acrodicst.c`, the Acrodicst library source
- `CMakeLists.txt`, the soon to be updated CMake entry file

The main program source

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <strings.h>
4 #ifdef USE_ACRODICT
5 #include "acrodict.h"
6 #endif
7 int main(int argc, char* argv[]) {
8
9     const char * name;
10 #ifdef USE_ACRODICT
11     const acroltem_t* item;
12 #endif
13
14     if (argc < 2) {
15         fprintf(stderr, "%s: you need one
16             argument\n", argv[0]);
17         fprintf(stderr, "%s<name>\n", argv
18             [0]);
19         exit(EXIT_FAILURE);
20     }
21     name = argv[1];
22 #ifndef USE_ACRODICT
23     if (strcasecmp(name, "toulibre")==0) {
24         printf("Toulibre is a french
25             organization promoting FLOSS
26             .\n");
27     }
28 #else
29     item = acrodect_get(name);
30     if (NULL!=item) {
31         printf("%s: %s\n", item->name, item->
32             description);
33     } else if (item=acrodect_get_approx(
34         name)) {
35         printf("<%s> is unknown, may be you
36             mean:\n", name);
37         printf("%s: %s\n", item->name, item->
38             description);
39     }
40 #endif
41     else {
42         printf("Sorry, I don't know: <%s>\n
43             ", name);
44         return EXIT_FAILURE;
45     }
46     return EXIT_SUCCESS;
47 }
```

```
24
25 }
26 #else
27     item = acrodect_get(name);
28     if (NULL!=item) {
29         printf("%s: %s\n", item->name, item->
30             description);
31     } else if (item=acrodect_get_approx(
32         name)) {
33         printf("<%s> is unknown, may be you
34             mean:\n", name);
35         printf("%s: %s\n", item->name, item->
36             description);
37     }
38 #endif
39     else {
40         printf("Sorry, I don't know: <%s>\n
41             ", name);
42         return EXIT_FAILURE;
43     }
44     return EXIT_SUCCESS;
45 }
```


The library source

```
1 #ifndef ACRODICT_H
2 #define ACRODICT_H
3 typedef struct acroltem {
4     char* name;
5     char* description;
6 } acroltem_t;
7
8 const acroltem_t*
9 acrodic_get(const char* name);
10 #endif

1 #include <stdlib.h>
2 #include <string.h>
3 #include "acrodic.h"
4 static const acroltem_t acrodic[] = {
5     {"Toulibre", "Toulibre_iss_a_french_
6         organization_promoting_FLOSS"},
7     {"GNU", "GNU_is_Not_Unix"},
8     {"GPL", "GNU_general_Public_License"
9     },
10    {"BSD", "Berkeley_Software_
11        Distribution"},
12    {"CULTe", "Club_des_Utilisateurs_de_
13        Logiciels_libres_et_de_gnu/
14        linux_de_Toulouse_et_des_
15        environs"},
16    {"Lea", "Lea-Linux:_Linux_entre_ami(e
17        )s"},
18    {"RMLL", "Rencontres_Mondiales_du_
19        Logiciel_Libre"},
20    {"FLOSS", "Free_Libre_Open_Source_
21        Software"},
22    {"", ""}};
23 const acroltem_t*
24 acrodic_get(const char* name) {
25     int current = 0;
26     int found = 0;
27     while ((strlen(acrodic[current].name
28         )>0) && !found) {
29         if (strcmp(name, acrodic[
30             current].name)==0) {
31             found=1;
32         } else {
33             current++;
34         }
35     }
36     if (found) {
37         return &acrodic[current];
38     } else {
39         return NULL;
40     }
41 }
```



Building a library I

Listing 2: Building a simple program + shared library

```
1 cmake_minimum_required (VERSION 2.8)
2 project (TotallyFree C)
3 add_executable (Acrolibre acrolibre.c)
4 set (LIBSRC acrodicth.c acrodicth.h)
5 add_library (acrodicth ${LIBSRC})
6 add_executable (Acrodicthlibre acrolibre.c)
7 target_link_libraries (Acrodicthlibre acrodicth)
8 set_target_properties (Acrodicthlibre
9     PROPERTIES COMPILE_FLAGS "-DUSE_ACRODICT")
```



Building a library II

And it builds...

All in all CMake generates appropriate Unix makefiles which build all this smoothly.

CMake + Unix Makefile

```
1 $ make
2 [ 33%] Building C object CMakeFiles/acrodict.dir/acrodict.c.o
3 Linking C shared library libacrodict.so
4 [ 33%] Built target acrodict
5 [ 66%] Building C object CMakeFiles/Acrodictlibre.dir/acrolibre.c.o
6 Linking C executable Acrodictlibre
7 [ 66%] Built target Acrodictlibre
8 [100%] Building C object CMakeFiles/Acrolibre.dir/acrolibre.c.o
9 Linking C executable Acrolibre
10 [100%] Built target Acrolibre
11 $ ls -F
12 Acrodictlibre* CMakeCache.txt cmake_install.cmake Makefile
13 Acrolibre* CMakeFiles/ libacrodict.so*
```



Building a library III

And it works...

We get the two different variants of our program, with varying capabilities.

```
1 $ ./Acrolibre toulibre
2 Toulibre is a french organization promoting FLOSS.
3 $ ./Acrolibre FLOSS
4 Sorry, I don't know: <FLOSS>
5 $ ./Acrodictlibre FLOSS
6 FLOSS: Free Libre Open Source Software

$ make help
The following are some of the valid targets
  for this Makefile:
... all (the default if no target is provided)
... clean
... depend
... Acrodictlibre
... Acrolibre
... acrodict
...
```

Generated Makefiles has several builtin targets besides the expected ones:

- one per target (library or executable)
- clean, all
- more to come ...



User controlled build option

User controlled option

May be our users don't want the acronym dictionary support.
We can use CMake **OPTION** command.

Listing 3: User controlled build option

```
1 cmake_minimum_required (VERSION 2.8)
2 # This project use C source code
3 project (TotallyFree C)
4 # Build option with default value to ON
5 option(WITH_ACRODICT "Include acronym dictionary support" ON)
6 set(BUILD_SHARED_LIBS true)
7 # build executable using specified list of source files
8 add_executable(Acrolibre acrolibre.c)
9 if (WITH_ACRODICT)
10     set(LIBSRC acrodicth acrodicth.c)
11     add_library(acrodicth ${LIBSRC})
12     add_executable(Acrodicthlibre acrolibre.c)
13     target_link_libraries(Acrodicthlibre acrodicth)
14     set_target_properties(Acrodicthlibre PROPERTIES COMPILE_FLAGS "-DUSE_ACRODICT")
15 endif(WITH_ACRODICT)
```



Too much keyboard, time to click? I

CMake comes with several tools

A matter of choice / taste:

- a command line: `cmake`
- a curses-based TUI: `ccmake`
- a QT-based GUI: `cmake-gui`

Call convention

All tools expect to be called with a single argument which may be interpreted in 2 different ways.

- path to the source tree, e.g.: `cmake /path/to/source`
- path to an existing build tree, e.g.: `cmake-gui .`



Too much keyboard, time to click? II

ccmake : the curse-based TUI (demo)

```
Fichier Éditer Affichage Terminal Aller Aide
Page 1 of 1
CMAKE_BUILD_TYPE
CMAKE_INSTALL_PREFIX /usr/local
WITH_ACRODICT ON

CMAKE BUILD TYPE: Choose the type of build, options are: None(CMAKE CXX FLAGS or
Press [enter] to edit option      CMake Version 2.8.7.20120121-g751713-dirty
Press [c] to configure
Press [h] for help                Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```

Here we can choose to toggle the WITH_ACRONYM **OPTION**.



Too much keyboard, time to click? III

cmake-gui : the QT-based GUI (demo)

File Tools Options Help

Where is the source code:

Where to build the binaries:

Search: Grouped Advanced

| Name | Value |
|----------------------|-------------------------------------|
| ▼ Ungrouped Entries | |
| WITH_ACRODICT | <input checked="" type="checkbox"/> |
| ▼ CMAKE | |
| CMAKE_BUILD_TYPE | |
| CMAKE_INSTALL_PREFIX | /usr/local |

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Unix Makefiles

Configuring done

Again, we can choose to toggle the WITH_ACRONYM **OPTION**.



Remember CMake is a build **generator**?

The number of active generators depends on the platform we are running on Unix, **Apple**, **Windows**:

| | | | |
|----|----------------------------|----|---|
| 1 | Borland Makefiles | 17 | Visual Studio 9 2008 |
| 2 | MSYS Makefiles | 18 | Visual Studio 9 2008 IA64 |
| 3 | MinGW Makefiles | 19 | Visual Studio 9 2008 Win64 |
| 4 | NMake Makefiles | 20 | Watcom WMake |
| 5 | NMake Makefiles JOM | 21 | CodeBlocks - MinGW Makefiles |
| 6 | Unix Makefiles | 22 | CodeBlocks - NMake Makefiles |
| 7 | Visual Studio 10 | 23 | CodeBlocks - Unix Makefiles |
| 8 | Visual Studio 10 IA64 | 24 | Eclipse CDT4 - MinGW Makefiles |
| 9 | Visual Studio 10 Win64 | 25 | Eclipse CDT4 - NMake Makefiles |
| 10 | Visual Studio 11 | 26 | Eclipse CDT4 - Unix Makefiles |
| 11 | Visual Studio 11 Win64 | 27 | KDevelop3 |
| 12 | Visual Studio 6 | 28 | KDevelop3 - Unix Makefiles |
| 13 | Visual Studio 7 | 29 | XCode |
| 14 | Visual Studio 7 .NET 2003 | 30 | Ninja (in development) |
| 15 | Visual Studio 8 2005 | 31 | http://martine.github.com/ninja/ |
| 16 | Visual Studio 8 2005 Win64 | | |



Equally simple on other platforms

It is as easy for a windows build, however names for executables and libraries are computed in a **platform specific way**.

```
----- CMake + MinGW Makefile -----  
1  $ ls totally-free  
2  acrodict.h acrodict.c acrolibre.c CMakeLists.txt  
3  $ mkdir build-win32  
4  $ cd build-win32  
5  ...  
6  $ make  
7  Scanning dependencies of target acrodict  
8  [ 33%] Building C object CMakeFiles/acrodict.dir/acrodict.c.obj  
9  Linking C shared library libacrodict.dll  
10 Creating library file: libacrodict.dll.a  
11 [ 33%] Built target acrodict  
12 Scanning dependencies of target Acrodictlibre  
13 [ 66%] Building C object CMakeFiles/Acrodictlibre.dir/acrolibre.c.obj  
14 Linking C executable Acrodictlibre.exe  
15 [ 66%] Built target Acrodictlibre  
16 Scanning dependencies of target Acrolibre  
17 [100%] Building C object CMakeFiles/Acrolibre.dir/acrolibre.c.obj  
18 Linking C executable Acrolibre.exe  
19 [100%] Built target Acrolibre
```



Installing things

Install

Several parts of the software may need to be installed, this is controlled by the CMake `install` command.

Remember `cmake --help-command install!!`

Listing 4: install command examples

```
1  ...
2  add_executable(Acrolibre acrolibre.c)
3  install(TARGETS Acrolibre DESTINATION bin)
4  if (WITH_ACRODICT)
5  ...
6  install(TARGETS Acrodiclibre acrodiclibre
7  RUNTIME DESTINATION bin
8  LIBRARY DESTINATION lib
9  ARCHIVE DESTINATION lib/static)
10 install(FILES acrodiclibre.h DESTINATION include)
11 endif(WITH_ACRODICT)
```



Controlling installation destination

Use relative DESTINATION

One should always use relative installation DESTINATION unless you really want to use absolute path like /etc.

Then depending when you install:



Controlling installation destination

Use relative DESTINATION

One should always use relative installation DESTINATION unless you really want to use absolute path like /etc.

Then depending when you install:

- At CMake-time set `CMAKE_INSTALL_PREFIX` value

```
$ cmake --help-variable CMAKE_INSTALL_PREFIX
```



Controlling installation destination

Use relative DESTINATION

One should always use relative installation DESTINATION unless you really want to use absolute path like /etc.

Then depending when you install:

- At CMake-time set `CMAKE_INSTALL_PREFIX` value

```
$ cmake --help-variable CMAKE_INSTALL_PREFIX
```
- At Install-time use `DESTDIR` mechanism (Unix Makefiles)

```
$ make DESTDIR=/tmp/testinstall install
```



Controlling installation destination

Use relative DESTINATION

One should always use relative installation DESTINATION unless you really want to use absolute path like /etc.

Then depending when you install:

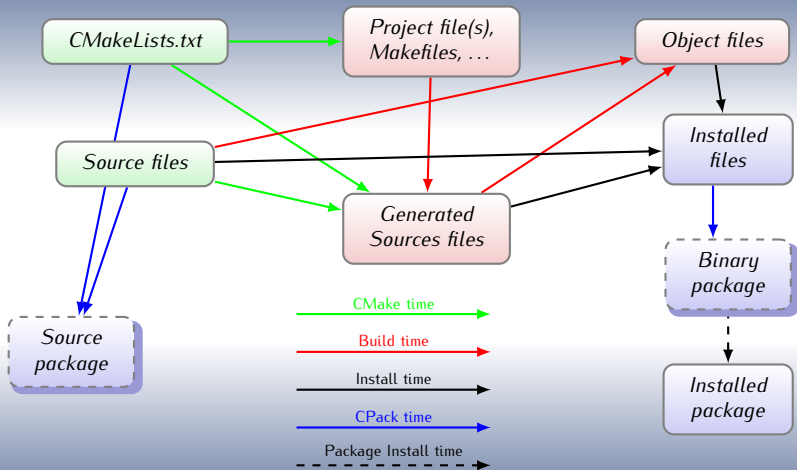
- At **CMake-time** set **CMAKE_INSTALL_PREFIX** value

```
$ cmake --help-variable CMAKE_INSTALL_PREFIX
```
- At Install-time use DESTDIR mechanism (Unix Makefiles)

```
$ make DESTDIR=/tmp/testinstall install
```
- At **CPack-time**, CPack what? ...be patient.
- At Package-install-time, we will see that later



The CMake workflow (pictured)





Using CMake variables

CMake variables

They are used by the user to simplify its `CMakeLists.txt`, but CMake uses many (~170+) of them to control/change its [default] behavior. Try: `cmake --help-variables-list`.

Inside a CMake script

```
set(CMAKE_INSTALL_PREFIX /home/eric/testinstall)  
$ cmake --help-command set
```

On the command line/TUI/GUI

Remember that (beside options) each CMake tool takes a single argument (source tree or **existing** build tree)

```
$ cmake -DCMAKE_INSTALL_PREFIX=/home/eric/testinstall .
```



The install target

Install target

The `install` target of the underlying build tool (in our case `make`) appears in the generated build system as soon as some `install` command are used in the `CMakeLists.txt`.

```
1 $ make DESTDIR=/tmp/testinstall install
2 [ 33%] Built target acrodict
3 [ 66%] Built target Acrodictlibre
4 [100%] Built target Acrolibre
5 Install the project...
6 -- Install configuration: ""
7 -- Installing: /tmp/testinstall/bin/Acrolibre
8 -- Installing: /tmp/testinstall/bin/Acrodictlibre
9 -- Removed runtime path from "/tmp/testinstall/bin/Acrodictlibre"
10 -- Installing: /tmp/testinstall/lib/libacrodict.so
11 -- Installing: /tmp/testinstall/include/acrodict.h
12 $
```



Package the whole thing

CPack

CPack is a CMake friend application (detailed later) which may be used to easily package your software.

Listing 5: add CPack support

```
1 ...
2 endif(WITH_ACRODICT)
3 ...
4 # Near the end of the CMakeLists.txt
5 # Chose your CPack generator
6 set(CPACK_GENERATOR "TGZ")
7 # Setup package version
8 set(CPACK_PACKAGE_VERSION_MAJOR 0)
9 set(CPACK_PACKAGE_VERSION_MINOR 1)
10 set(CPACK_PACKAGE_VERSION_PATCH 0)
11 # 'call' CPack
12 include(CPack)
```

```
$ make package
[ 33%] Built target acrodict
[ 66%] Built target Acrodictlibre
[100%] Built target Acrolibre
Run CPack packaging tool...
CPack: Create package using TGZ
CPack: Install projects
CPack: - Run preinstall target for: TotallyFree
CPack: - Install project: TotallyFree
CPack: Create package
CPack: - package: <build-tree>/...
      TotallyFree-0.1.0-Linux.tar.gz generated.
$ tar ztvf TotallyFree-0.1.0-Linux.tar.gz
... TotallyFree-0.1.0-Linux/include/acrodict.h
... TotallyFree-0.1.0-Linux/bin/Acrolibre
... TotallyFree-0.1.0-Linux/bin/Acrodictlibre
... TotallyFree-0.1.0-Linux/lib/libacrodict.so
```



CPack the packaging friend

CPack is a standalone generator

As we will see later on, CPack is standalone application, which as CMake is a generator.

```
$ cpack -G ZIP
```

```
CPack: Create package using ZIP
CPack: Install projects
CPack: - Run preinstall target for: TotallyFree
CPack: - Install project: TotallyFree
CPack: Create package
CPack: - package: <build-tree>/...
      TotallyFree-0.1.0-Linux.zip generated.
```

```
$ unzip -t TotallyFree-0.1.0-Linux.zip
Archive:  TotallyFree-0.1.0-Linux.zip
  testing: To.../include/acrodict.h    OK
  testing: To.../bin/Acrolibre         OK
  testing: To.../bin/Acrodictlibre    OK
  testing: To.../lib/libacrodict.so    OK
No errors detected in compressed
data of TotallyFree-0.1.0-Linux.zip.
```

```
$ cpack -G RPM
```

```
CPack: Create package using RPM
CPack: Install projects
CPack: - Run preinstall target for: TotallyFree
CPack: - Install project: TotallyFree
CPack: Create package
CPackRPM: Will use GENERATED spec file: <build-tree>/...
      _CPack_Packages/Linux/RPM/SPECS/totallyfree.spec
CPack: - package: <build-tree>/...
      TotallyFree-0.1.0-Linux.rpm generated.
$ rpm -qpl TotallyFree-0.1.0-Linux.rpm
/usr
/usr/bin
/usr/bin/Acrodictlibre
/usr/bin/Acrolibre
/usr/include
/usr/include/acrodict.h
/usr/lib
/usr/lib/libacrodict.so
```



Didn't you mentioned testing? I

CTest

CTest is a CMake friend application (detailed later) which may be used to easily test your software.

Listing 6: add CTest support

```
1  ...
2  endif(WITH_ACRODICT)
3  ...
4  enable_testing()
5  add_test(toulibre-builtin
6           Acrolibre "toulibre")
7  add_test(toulibre-dict
8           Acrodictlibre "toulibre")
9  add_test(FLOSS-dict
10          Acrodictlibre "FLOSS")
11 add_test(FLOSS-fail
12          Acrolibre "FLOSS")
```

```
$ make test
Running tests...
Test project <buildtree-prefix>/build
  Start 1: toulibre-builtin
1/4 Test #1: toulibre-builtin .... Passed 0.00 sec
  Start 2: toulibre-dict
2/4 Test #2: toulibre-dict..... Passed 0.00 sec
  Start 3: FLOSS-dict
3/4 Test #3: FLOSS-dict ..... Passed 0.00 sec
  Start 4: FLOSS-fail
4/4 Test #4: FLOSS-fail .....***Failed 0.00 sec

75% tests passed, 1 tests failed out of 4

Total Test time (real) = 0.01 sec
```

The following tests FAILED:
4 - FLOSS-fail (Failed)



Didn't you mentioned testing? II

Tailor success rule

CTest uses the return code in order to get success/failure status, but one can tailor the success/fail rule.

Listing 7: add CTest support

```
1  ...
2  endif(WITH_ACRODICT)
3  ...
4  enable_testing()
5  add_test(toulibre-builtin
6           Acrolibre "toulibre")
7  add_test(toulibre-dict
8           Acrodictlibre "toulibre")
9  add_test(FLOSS-dict
10          Acrodictlibre "FLOSS")
11 add_test(FLOSS-fail
12          Acrolibre "FLOSS")
13 set_tests_properties (FLOSS-fail
14                       PROPERTIES
15                       PASS_REGULAR_EXPRESSION
16                       "Sorry, I don't know:.*FLOSS")
```

```
$ make test
Running tests...
Test project <buildtree-prefix>/build
  Start 1: toulibre-builtin
1/4 Test #1: toulibre-builtin .... Passed 0.00 sec
  Start 2: toulibre-dict
2/4 Test #2: toulibre-dict..... Passed 0.00 sec
  Start 3: FLOSS-dict
3/4 Test #3: FLOSS-dict ..... Passed 0.00 sec
  Start 4: FLOSS-fail
4/4 Test #4: FLOSS-fail ..... Passed 0.00 sec

100% tests passed, 0 tests failed out of 4

Total Test time (real) = 0.01 sec
```



CTest the testing friend

CTest is a standalone generic test driver

As we will see later on, CTest is standalone application, which can run a set of test programs.

```
$ ctest -R toulibre-
Test project <build-tree>/build
  Start 1: toulibre-builtin
1/2 Test #1: toulibre-builtin .. Passed 0.00 sec
  Start 2: toulibre-dict
2/2 Test #2: toulibre-dict ..... Passed 0.00 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) = 0.01 sec
```

```
$ ctest -R FLOSS-fail -V
Test project <build-tree>
Constructing a list of tests
Done constructing a list of tests
Checking test dependency graph...
Checking test dependency graph end
test 4
  Start 4: FLOSS-fail
4: Test command: <build-tree>/Acrolibre "FLOSS"
4: Test timeout computed to be: 9.99988e+06
4: Sorry, I don't know: <FLOSS>
1/1 Test #4: FLOSS-fail .....***Failed 0.00 sec

0% tests passed, 1 tests failed out of 1
Total Test time (real) = 0.00 sec

The following tests FAILED:
 4 - FLOSS-fail (Failed)
Errors while running CTest
```



CDash the test results publishing

Dashboard

CTest may help publishing the results of the tests on a CDash dashboard (<http://www.cdash.org/>) for easing collective regression testing. More on this later...

<http://www.orfeo-toolbox.org/~http://dash.orfeo-toolbox.org/>

OTB Dashboard

DASHBOARD | CALENDAR | PREVIOUS | CURRENT | PROJECT

No update data as of Friday, February 03 2012 19:00:00 CET [Help](#)

[Show Filters](#)

Continuous | Experimental | Nightly Applications | Continuous Applications | Experimental Applications | Coverage | Dynamic Analysis

| Site | Build Name | Update | | Configure | | | Build | | | Test | | | Build Time | Labels | |
|---|---|--------|-----|-----------|------|-----|-------|------|------|--------|------------------|--------------------|------------|-------------------------|--------|
| | | Files | Min | Error | Warn | Min | Error | Warn | Min | NotRun | Fail | Pass | | | Min |
| pc-christophe.cst.cnes.fr | ArchLinux-64bits-Release | 6 | 0.1 | 0 | 1 | 1.1 | 0 | 6 | 126 | 0 | 60 ¹⁰ | 2410 ¹⁰ | 130.6 | 2012-02-03T20:11:37 CET | (none) |
| pc-gricorin@elm.cst.cnes.fr | Basic-Liberty10_04-64bits-Release | 0 | 0.1 | 0 | 2 | 0.7 | 0 | 0 | 23 | 0 | 67 ¹⁰ | 2346 ¹⁰ | 25.2 | 2012-02-03T23:37:34 CET | (none) |
| titon | Deb50-64bits-Release | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2012-02-03T20:15:19 CET | (none) |
| leod.c-s.fr | MacOSX10_5-Release-macport | 6 | 0.1 | 0 | 0 | 1.9 | 0 | 0 | 72.9 | 0 | 62 ¹⁰ | 2396 ¹⁰ | 46.1 | 2012-02-03T20:42:25 CET | (none) |



Summary

CMake basics

Using CMake basics we can already do a lot a things with minimal writing.

- Write simple build specification file: `CMakeLists.txt`
- Discover compilers (C, C++, Fortran)
- Build executable and library (shared or static) in a cross-platform manner
- Package the resulting binaries with CPack
- Runs systematic test with CTest and publish them with CDash



Seeking more information or help

There are several places you can go by yourself:

- 1 Read the FAQ:

http://www.cmake.org/Wiki/CMake_FAQ

- 2 Read the Wiki:

<http://www.cmake.org/Wiki/CMake>

- 3 Ask on the Mailing List:

<http://www.cmake.org/cmake/help/mailing.html>

- 4 Browse the built-in help:

```
cmake --help-xxxxx
```



Outline

- 1 Basic CMake usage
- 2 Discovering environment specificities**
 - Handling platform specificities
 - Working with external packages
- 3 More CMake scripting
 - Custom commands
 - Generated files
- 4 Advanced CMake usage
 - Cross-compiling with CMake
 - Export your project



Outline

- 1 Basic CMake usage
- 2 Discovering environment specificities**
 - Handling platform specificities
 - Working with external packages
- 3 More CMake scripting
 - Custom commands
 - Generated files
- 4 Advanced CMake usage
 - Cross-compiling with CMake
 - Export your project



How to discover system

System/compiler specific variables

Right after the **project** command CMake has set up a bunch of variables which can be used to tailor the build in a platform specific way.

- system specific
 - **WIN32** True on windows systems, including win64.
 - **UNIX** True for UNIX and UNIX like operating systems.
 - **APPLE** True if running on Mac OSX.
 - **CYGWIN** True for cygwin.
- compiler specific
 - **MSVC** True when using Microsoft Visual C
 - **CMAKE_COMPILER_IS_GNU<LANG>** True if the <LANG> compiler is GNU.
 - **MINGW** true if the compiler is MinGW.



Handle system specific code

Some functions like `strcasestr` (lines 6 and 7) may not be available on all platforms.

Listing 8: excerpt from `acrodic.c`

```
1  const acroltem_t* acrodic_get_approx(const char* name) {
2      int current =0;
3      int found =0;
4      #ifndef GUESS_NAME
5          while ((strlen(acrodic[current].name)>0) && !found) {
6              if ((strcasestr(name,acrodic[current].name)!=0) ||
7                  (strcasestr(acrodic[current].name,name)!=0)) {
8                  found=1;
9              } else {
10                 current++;
11             }
12         }
13         if (found) {
14             return &(acrodic[current]);
15         } else
16         #endif
17         {
18             return NULL;
19         }
20     }
```



Use system specific option

```
1 # Build option with default value to ON
2 option(WITH_ACRODICT "Include␣acronym␣dictionary␣support" ON)
3 if(NOT WIN32)
4   option(WITH_GUESS_NAME "Guess␣acronym␣name" ON)
5 endif(NOT WIN32)
6 ...
7 if(WITH_ACRODICT)
8   # list of sources in our library
9   set(LIBSRC acrodict.h acrodict.c)
10  if(WITH_GUESS_NAME)
11    set_source_files_properties(acrodict.c PROPERTIES COMPILE_FLAGS "-DGUESS_NAME")
12  endif(WITH_GUESS_NAME)
13  add_library(acrodict ${LIBSRC})
14  ...
```

Line 4 defines a CMake option, but not on **WIN32** system. Then on line 11, if the option is set then we pass a source specific compile flags.

```
cmake --help-command set_source_files_properties
```



System specific in real life

Real [numeric] life project

Real projects (i.e. not the toy of this tutorial) have many parts of their `CMakeLists.txt` which deals with system/compiler specific option/feature.

- MuseScore : <http://musescore.org>

<http://mscore.svn.sourceforge.net/viewvc/mscore/trunk/mscore/mscore/>

Display `CMakeLists.txt` from MuseScore

- CERTI : <https://savannah.nongnu.org/projects/certi/>

<http://cvs.savannah.gnu.org/viewvc/certi/?root=certi>

- CMake (of course): <http://www.cmake.org>
- LLVM: <http://llvm.org/docs/CMake.html>
- a lot more ...



What about projectConfig.h file? I

Project config files

Sometimes it's easier to test for feature and then write a configuration file (`config.h`, `project_config.h`, ...). The CMake way to do that is to:

- 1 lookup system informations using CMake variable, functions, macros (built-in or imported) then set various variables,
- 2 use the defined variable in order to write a template configuration header file
- 3 then use **configure_file** in order to produce the actual config file from the template.



What about projectConfig.h file? II

Listing 9: Excerpt from CERTI project main CMakeLists.txt

```
1 # Load Checker macros
2 INCLUDE(CheckFunctionExists)
3
4 FIND_FILE(HAVE_STDINT_H NAMES stdint.h)
5 FIND_FILE(HAVE_SYS_SELECT_H NAMES select.h
6           PATH_SUFFIXES sys)
7 INCLUDE(CheckIncludeFile)
8 CHECK_INCLUDE_FILE(time.h HAVE_TIME_H)
9 FIND_LIBRARY(RT_LIBRARY rt)
10 if(RT_LIBRARY)
11     SET(CMAKE_REQUIRED_LIBRARIES ${CMAKE_REQUIRED_LIBRARIES} ${RT_LIBRARY})
12 endif(RT_LIBRARY)
13
14 CHECK_FUNCTION_EXISTS(clock_gettime HAVE_CLOCK_GETTIME)
15 CHECK_FUNCTION_EXISTS(clock_settime HAVE_CLOCK_SETTIME)
16 CHECK_FUNCTION_EXISTS(clock_getres HAVE_CLOCK_GETRES)
17 CHECK_FUNCTION_EXISTS(clock_nanosleep HAVE_CLOCK_NANOSLEEP)
18 IF (HAVE_CLOCK_GETTIME AND HAVE_CLOCK_SETTIME AND HAVE_CLOCK_GETRES)
19     SET(HAVE_POSIX_CLOCK 1)
20 ENDIF (HAVE_CLOCK_GETTIME AND HAVE_CLOCK_SETTIME AND HAVE_CLOCK_GETRES)
21 ...
22 CONFIGURE_FILE(${CMAKE_CURRENT_SOURCE_DIR}/config.h.cmake
23               ${CMAKE_CURRENT_BINARY_DIR}/config.h)
```



What about projectConfig.h file? III

Excerpt from CERTI config.h.cmake

```
1  /* define if the compiler has numeric_limits<T> */
2  #cmakedefine HAVE_NUMERIC_LIMITS
3
4  /* Define to 1 if you have the <stdint.h> header file. */
5  #cmakedefine HAVE_STDINT_H 1
6
7  /* Define to 1 if you have the <stdlib.h> header file. */
8  #cmakedefine HAVE_STDLIB_H 1
9
10 /* Define to 1 if you have the <strings.h> header file. */
11 #cmakedefine HAVE_STRINGS_H 1
12 ...
13 /* Name of package */
14 #cmakedefine PACKAGE "@PACKAGE_NAME@"
15
16 /* Define to the address where bug reports for this package should be sent. */
17 #cmakedefine PACKAGE_BUGREPORT "@PACKAGE_BUGREPORT@"
18
19 /* Define to the full name of this package. */
20 #cmakedefine PACKAGE_NAME "@PACKAGE_NAME@"
21
22 /* Define to the full name and version of this package. */
23 #cmakedefine PACKAGE_STRING "@PACKAGE_NAME@-@PACKAGE_VERSION@"
```



What about projectConfig.h file? IV

And you get something like:

```
_____ Excerpt from generated CERTI config.h _____  
1  /* define if the compiler has numeric_limits<T> */  
2  #define HAVE_NUMERIC_LIMITS  
3  
4  /* Define to 1 if you have the <stdint.h> header file. */  
5  #define HAVE_STDINT_H 1  
6  
7  /* Define to 1 if you have the <stdlib.h> header file. */  
8  #define HAVE_STDLIB_H 1  
9  
10 /* Define to 1 if you have the <strings.h> header file. */  
11 #define HAVE_STRINGS_H 1  
12 ...  
13 /* Name of package */  
14 /* #undef PACKAGE */  
15  
16 /* Define to the address where bug reports for this package should be sent. */  
17 #define PACKAGE_BUGREPORT "certi-devel@nongnu.org"  
18  
19 /* Define to the full name of this package. */  
20 #define PACKAGE_NAME "CERTI"  
21  
22 /* Define to the full name and version of this package. */  
23 /* #undef PACKAGE_STRING */
```



Outline

- 1 Basic CMake usage
- 2 Discovering environment specificities**
 - Handling platform specificities
 - Working with external packages
- 3 More CMake scripting
 - Custom commands
 - Generated files
- 4 Advanced CMake usage
 - Cross-compiling with CMake
 - Export your project



The `find_package` command I

Finding external package

Project may be using external libraries, program, files etc... Those can be found using the `find_package` command.

Listing 10: using libxml2

```
1 find_package (LibXml2)
2 if (LIBXML2_FOUND)
3     add_definitions(-DHAVE_XML ${LIBXML2_DEFINITIONS})
4     include_directories (${LIBXML2_INCLUDE_DIR})
5 else (LIBXML2_FOUND)
6     set (LIBXML2_LIBRARIES "")
7 endif (LIBXML2_FOUND)
8 ...
9 target_link_libraries (MyTarget ${LIBXML2_LIBRARIES})
```



The `find_package` command II

- Find modules usually defines standard variables (for module XXX)
 - ① `XXX_FOUND`: Set to false, or undefined, if we haven't found, or don't want to use XXX.
 - ② `XXX_INCLUDE_DIRS`: The final set of include directories listed in one variable for use by client code.
 - ③ `XXX_LIBRARIES`: The libraries to link against to use XXX. These should include full paths.
 - ④ `XXX_DEFINITIONS`: Definitions to use when compiling code that uses XXX.
 - ⑤ `XXX_EXECUTABLE`: Where to find the XXX tool.
 - ⑥ `XXX_LIBRARY_DIRS`: Optionally, the final set of library directories listed in one variable for use by client code.
- See `doc cmake --help-module FindLibXml2`
- Many modules are provided by CMake (130 as of CMake 2.8.7)



The `find_package` command III

- You may write your own:
http://www.cmake.org/Wiki/CMake:Module_Maintainers
- You may find/borrow modules from other projects which use CMake
 - KDE4:
<http://websvn.kde.org/trunk/KDE/kdelibs/cmake/modules/>
 - PlPlot: <http://plplot.svn.sourceforge.net/viewvc/plplot/trunk/cmake/modules/>
 - <http://cmake-modules.googlecode.com/svn/trunk/Modules/>
 - probably many more...
- A module may provide not only CMake variables but new CMake macros (we will see that later with the **MACRO**, **FUNCTION** CMake language commands)



The other `find_xxxx` commands I

The `find_xxx` command family

`find_package` is a high level module finding mechanism but there are lower-level CMake commands which may be used to write find modules or anything else inside `CMakeLists.txt`

- to find an executable program: `find_program`
- to find a library: `find_library`
- to find any kind of file: `find_file`
- to find a path where a file reside: `find_path`



The other `find_xxxx` commands II

```
1 # - Find Prelude compiler
2 # Find the Prelude synchronous language compiler with associated includes path.
3 # See http://www.lifl.fr/~forget/prelude.html
4 # This module defines
5 # PRELUDE_COMPILER, the prelude compiler
6 # PRELUDE_COMPILER_VERSION, the version of the prelude compiler
7 # PRELUDE_INCLUDE_DIR, where to find dword.h, etc.
8 # PRELUDE_FOUND, If false, Prelude was not found.
9 # On can set PRELUDE_PATH_HINT before using find_package(Prelude) and the
10 # module with use the PATH as a hint to find preludec.
11 ...
12 if(PRELUDE_PATH_HINT)
13     message(STATUS "FindPrelude: using PATH_HINT: ${PRELUDE_PATH_HINT}")
14 else()
15     set(PRELUDE_PATH_HINT)
16 endif()
17 # FIND_PROGRAM twice using NO_DEFAULT_PATH on first shot
18 find_program(PRELUDE_COMPILER NAMES preludec
19     PATHS ${PRELUDE_PATH_HINT} PATH_SUFFIXES bin
20     NO_DEFAULT_PATH
21     DOC "Path to the Prelude compiler command 'preludec'")
22 find_program(PRELUDE_COMPILER NAMES preludec
23     PATHS ${PRELUDE_PATH_HINT} PATH_SUFFIXES bin
24     DOC "Path to the Prelude compiler command 'preludec'")
```



The other `find_xxxx` commands III

```
25
26 if (PRELUDE_COMPILER)
27     # get the path where the prelude compiler was found
28     get_filename_component (PRELUDE_PATH ${PRELUDE_COMPILER} PATH)
29     # remove bin
30     get_filename_component (PRELUDE_PATH ${PRELUDE_PATH} PATH)
31     # add path to PRELUDE_PATH_HINT
32     list (APPEND PRELUDE_PATH_HINT ${PRELUDE_PATH})
33     execute_process (COMMAND ${PRELUDE_COMPILER} --version
34                     OUTPUT_VARIABLE PRELUDE_COMPILER_VERSION
35                     OUTPUT_STRIP_TRAILING_WHITESPACE)
36 endif (PRELUDE_COMPILER)
37
38 find_path (PRELUDE_INCLUDE_DIR NAMES dword.h
39           PATHS ${PRELUDE_PATH_HINT} PATH_SUFFIXES lib/prelude
40           DOC "The Prelude include headers")
41 ...
42 # handle the QUIETLY and REQUIRED arguments and set PRELUDE_FOUND to TRUE if
43 # all listed variables are TRUE
44 include (FindPackageHandleStandardArgs)
45 FIND_PACKAGE_HANDLE_STANDARD_ARGS (PRELUDE
46                                     REQUIRED_VARS PRELUDE_COMPILER PRELUDE_INCLUDE_DIR)
```



Advanced use of external package I

Installed External package

The previous examples suppose that you have the package you are looking for on your host.

- you did install the runtime libraries
- you did install eventual developer libraries, headers and tools

What if the external packages:

- are only available as source (tarball, VCS repositories, ...)
- use a build system (autotools or CMake or ...)



Advanced use of external package II

ExternalProject_Add

The `ExternalProject.cmake` CMake module defines a high-level macro which does just that:

- 1 download/checkout source
- 2 update/patch
- 3 configure
- 4 build
- 5 install (and test)

...an external project

```
$ cmake --help-module ExternalProject
```



Outline

- 1 Basic CMake usage
- 2 Discovering environment specificities
 - Handling platform specificities
 - Working with external packages
- 3 **More CMake scripting**
 - Custom commands
 - Generated files
- 4 Advanced CMake usage
 - Cross-compiling with CMake
 - Export your project



The different CMake “modes”

- Normal mode: the mode used when processing `CMakeLists.txt`
- Command mode: `cmake -E <command>`, command line mode which offers basic command in a portable way:
- Process scripting mode: `cmake -P <script>`, used to execute a CMake script which is not a `CMakeLists.txt`.
- Wizard mode: `cmake -i`, interactive equivalent of the Normal mode.



The different CMake “modes”

- Normal mode: the mode used when processing `CMakeLists.txt`
- Command mode: `cmake -E <command>`, command line mode which offers basic command in a portable way:
works on all supported CMake platforms. I.e. you don't want to rely on shell or native command interpreter capabilities.
- Process scripting mode: `cmake -P <script>`, used to execute a CMake script which is not a `CMakeLists.txt`.
- Wizard mode: `cmake -i`, interactive equivalent of the Normal mode.



The different CMake “modes”

- Normal mode: the mode used when processing CMakeLists.txt
- Command mode: `cmake -E <command>`, command line mode which offers basic command in a portable way:
works on all supported CMake platforms. I.e. you don't want to rely on shell or native command interpreter capabilities.
- Process scripting mode: `cmake -P <script>`, used to execute a CMake script which is not a CMakeLists.txt.
Not all CMake commands are scriptable!!
- Wizard mode: `cmake -i`, interactive equivalent of the Normal mode.



Command mode

Just try:

```
1  $ cmake -E
2  CMake Error: cmake version 2.8.7
3  Usage: cmake -E [command] [arguments ...]
4  Available commands:
5  chdir dir cmd [args]...  - run command in a given directory
6  compare_files file1 file2 - check if file1 is same as file2
7  copy file destination    - copy file to destination (either file or directory)
8  copy_directory source destination - copy directory 'source' content to directory 'destination'
9  copy_if_different in-file out-file - copy file if input has changed
10 echo [string]...         - displays arguments as text
11 echo_append [string]... - displays arguments as text but no new line
12 environment              - display the current environment
13 make_directory dir       - create a directory
14 md5sum file1 [...]       - compute md5sum of files
15 remove [-f] file1 file2 ... - remove the file(s), use -f to force it
16 remove_directory dir     - remove a directory and its contents
17 rename oldname newname   - rename a file or directory (on one volume)
18 tar [cxt][vzf][cvfj] file.tar file/dir1 file/dir2 ... - create a tar archive
19 time command [args] ...  - run command and return elapsed time
20 touch file                - touch a file.
21 touch_nocreate file       - touch a file but do not create it.
22 Available on UNIX only:
23 create_symlink old new    - create a symbolic link new -> old
```



CMake scripting

Overview of CMake language

CMake is a declarative language which contains 90+ commands. It contains general purpose constructs: **set**, **unset**, **if**, **elseif**, **else**, **endif**, **foreach**, **while**, **break**

Remember:

```
1 $ cmake --help-command-list
2 $ cmake --help-command <command-name>
3 $ cmake --help-command message
4 cmake version 2.8.7
5   message
6     Display a message to the user.
7     message([STATUS|WARNING|AUTHOR_WARNING|FATAL_ERROR|SEND_ERROR]
8             "message to display" ...)
9     The optional keyword determines the type of message:
10    (none)      = Important information
11    STATUS      = Incidental information
12    WARNING     = CMake Warning, continue processing
13    AUTHOR_WARNING = CMake Warning (dev), continue processing
14    SEND_ERROR  = CMake Error, continue but skip generation
15    FATAL_ERROR = CMake Error, stop all processing
```



Higher level commands as well

- file manipulation with **file** : **READ**, **WRITE**, **APPEND**, **RENAME**, **REMOVE**, **MAKE_DIRECTORY**
- advanced files operations: **GLOB**, **GLOB_RECURSE** file name in a path, **DOWNLOAD**, **UPLOAD**
- working with path: **file** (**TO_CMAKE_PATH** / **TO_NATIVE_PATH** ...), **get_filename_component**
- execute an external process (with stdout, stderr and return code retrieval): **execute_process**
- builtin list manipulation command: **list** with sub-commands **LENGTH**, **GET**, **APPEND**, **FIND**, **APPEND**, **INSERT**, **REMOVE_ITEM**, **REMOVE_AT**, **REMOVE_DUPLICATES** **REVERSE**, **SORT**
- string manipulation: **string**, upper/lower case conversion, length, comparison, substring, regular expression match, ...



Portable script for building CMake I

As an example of what can be done with pure CMake script (script mode) here is a script for building CMake package using a previously installed CMake.

```
1  # Simple cmake script which may be used to build
2  # cmake from automatically downloaded source
3  #
4  #   cd tmp/
5  #   cmake -P CMake-autobuild-v2.cmake
6  # you should end up with a
7  #   tmp/cmake-x.y.z source tree
8  #   tmp/cmake-x.y.z-build build tree
9  # configure and compiled tree, using the tarball found on Kitware.
10
11 cmake_minimum_required(VERSION 2.8)
12 set(CMAKE_VERSION "2.8.7")
13 set(CMAKE_FILE_PREFIX "cmake-${CMAKE_VERSION}")
14 set(CMAKE_REMOTE_PREFIX "http://www.cmake.org/files/v2.8/")
15 set(CMAKE_FILE_SUFFIX ".tar.gz")
16 set(CMAKE_BUILD_TYPE "Debug")
17 set(CPACK_GEN "TGZ")
18
```



Portable script for building CMake II

```
19 set(LOCAL_FILE "${CMAKE_FILE_PREFIX}${CMAKE_FILE_SUFFIX}")
20 set(REMOTE_FILE "${CMAKE_REMOTE_PREFIX}${CMAKE_FILE_PREFIX}${CMAKE_FILE_SUFFIX}")
21
22 message(STATUS "Trying to autoinstall CMake version ${CMAKE_VERSION} using ${
    REMOTE_FILE} file...")
23
24 message(STATUS "Downloading...")
25 if (EXISTS ${LOCAL_FILE})
26     message(STATUS "Already there: nothing to do")
27 else (EXISTS ${LOCAL_FILE})
28     message(STATUS "Not there, trying to download...")
29     file(DOWNLOAD ${REMOTE_FILE} ${LOCAL_FILE}
30          TIMEOUT 600
31          STATUS DL_STATUS
32          LOG DL_LOG
33          SHOW_PROGRESS)
34     list(GET DL_STATUS 0 DL_NOK)
35     if (${DL_LOG} MATCHES "404 Not Found")
36         set(DL_NOK 1)
37     endif (${DL_LOG} MATCHES "404 Not Found")
38     if (DL_NOK)
39         # we shall remove the file because it is created
40         # with an inappropriate content
41         file(REMOVE ${LOCAL_FILE})
42         message(SEND_ERROR "Download failed: ${DL_LOG}")
```



Portable script for building CMake III

```
43     else (DL_NOK)
44         message (STATUS "Download successful.")
45     endif (DL_NOK)
46 endif (EXISTS ${LOCAL_FILE})
47
48 message (STATUS "Unarchiving the file")
49 execute_process (COMMAND ${CMAKE_COMMAND} -E tar zxvf ${LOCAL_FILE}
50                 RESULT_VARIABLE UNTAR_RES
51                 OUTPUT_VARIABLE UNTAR_OUT
52                 ERROR_VARIABLE UNTAR_ERR
53                 )
54 message (STATUS "CMake version ${CMAKE_VERSION} has been unarchived in ${
55             CMAKE_CURRENT_SOURCE_DIR}/${CMAKE_FILE_PREFIX}." )
56 message (STATUS "Configuring with CMake (build type=${CMAKE_BUILD_TYPE})...")
57 file (MAKE_DIRECTORY ${CMAKE_FILE_PREFIX}-build)
58 execute_process (COMMAND ${CMAKE_COMMAND} -DCMAKE_BUILD_TYPE=${CMAKE_BUILD_TYPE} -
59                 DBUILD_QtDialog:BOOL=ON ../ ${CMAKE_FILE_PREFIX}
60                 WORKING_DIRECTORY ${CMAKE_FILE_PREFIX}-build
61                 RESULT_VARIABLE CONFIG_RES
62                 OUTPUT_VARIABLE CONFIG_OUT
63                 ERROR_VARIABLE CONFIG_ERR
64                 TIMEOUT 200)
65 message (STATUS "Building with cmake --build...")
```



Portable script for building CMake IV

```
66 execute_process(COMMAND ${CMAKE_COMMAND} --build .
67                 WORKING_DIRECTORY ${CMAKE_FILE_PREFIX}--build
68                 RESULT_VARIABLE CONFIG_RES
69                 OUTPUT_VARIABLE CONFIG_OUT
70                 ERROR_VARIABLE CONFIG_ERR)
71
72 message(STATUS "Create package_${CPACK_GEN} with CPack...")
73 execute_process(COMMAND ${CMAKE_CPACK_COMMAND} -G ${CPACK_GEN}
74                 WORKING_DIRECTORY ${CMAKE_FILE_PREFIX}--build
75                 RESULT_VARIABLE CONFIG_RES
76                 OUTPUT_VARIABLE CONFIG_OUT
77                 ERROR_VARIABLE CONFIG_ERR)
78 message(STATUS "CMake version_${CMAKE_VERSION} has been built in ${
79                 CMAKE_CURRENT_SOURCE_DIR}/${CMAKE_FILE_PREFIX}.")
80 string(REGEX MATCH "CPack: _package:(.*)generated" PACKAGES "${CONFIG_OUT}")
81 message(STATUS "CMake package(s) are: ${CMAKE_MATCH_1}")
```




Build specific commands

- create executable or library: `add_executable`, `add_library`
- add compiler/linker definitions/options: `add_definition`, `include_directories`, `target_link_libraries`
- powerful installation specification: `install`
- probing command: `try_compile`, `try_run`
- fine control of various properties: `set_target_properties`, `set_source_files_properties`, `set_directory_properties`, `set_tests_properties` : 190+ different properties may be used.

```
$ cmake --help-property-list
```

```
$ cmake --help-property COMPILER_FLAGS
```



Outline

- 1 Basic CMake usage
- 2 **Discovering environment specificities**
 - Handling platform specificities
 - Working with external packages
- 3 **More CMake scripting**
 - Custom commands
 - Generated files
- 4 **Advanced CMake usage**
 - Cross-compiling with CMake
 - Export your project



What are CMake targets?

CMake target

Many times in the documentation you may read about CMake target. A target is something that CMake should build (i.e. generate something enabling the building of the target). A CMake target has **dependencies** and **properties**.

- 1 Executable are targets: **add_executable**
- 2 Libraries are targets: **add_library**
- 3 There exist some builtin targets: `install`, `clean`, `package`, ...
- 4 You may create custom targets: **add_custom_target**



Target dependencies and properties I

A CMake target has **dependencies** and **properties**.

Dependencies

Most of the time, source dependencies are computed from target specifications using CMake builtin dependency scanner (C, C++, Fortran) whereas library dependencies are inferred via **target_link_libraries** specification.

If this is not enough then one can use **add_dependencies**, or some properties.



Target dependencies and properties II

Properties

Properties may be attached to either target or source file (or even test). They may be used to tailor prefix or suffix to be used for libraries, compile flags, link flags, linker language, shared libraries version, ...

see : [set_target_properties](#) or [set_source_files_properties](#)

Sources vs Targets

Properties set to a target like **COMPILE_FLAGS** are used for all sources of the concerned target. Properties set to a source are used for the source file itself (which may be involved in several targets).



Custom targets and commands

Custom

Custom targets and custom commands are a way to create target which may be used to execute arbitrary command at **Build-time**.

- for target : **add_custom_target**
- for command : **add_custom_command**, in order to add some custom build step to another (existing) target.

This is usually for: generating source files (Flex, Bison) or other files derived from source like embedded documentation (Doxygen),
...



Outline

- 1 Basic CMake usage
- 2 **Discovering environment specificities**
 - Handling platform specificities
 - Working with external packages
- 3 **More CMake scripting**
 - Custom commands
 - Generated files**
- 4 **Advanced CMake usage**
 - Cross-compiling with CMake
 - Export your project



Generated files

List all the sources

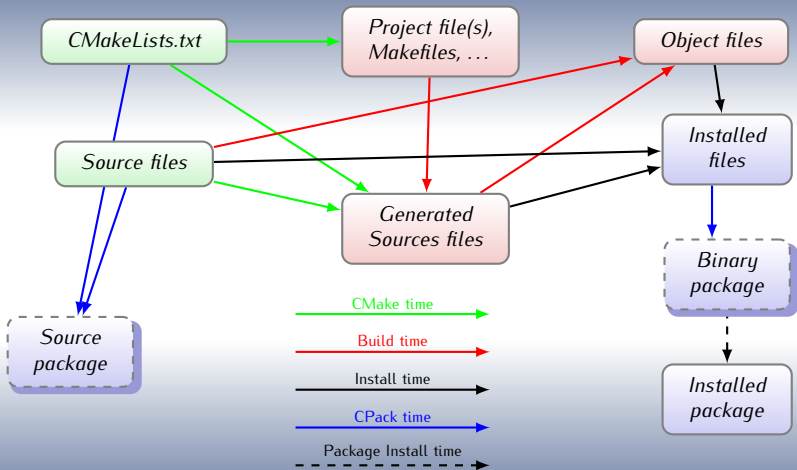
CMake advocates to specify all the source files explicitly (this do not use **file** (**GLOB** ...)) This is the only way to keep robust dependencies. Moreover you usually already need to do that when using a VCS (cvs, subversion, git, hg,...).

However some files may be generated during the build (using `add_custom_xxx`) in this case you must tell CMake that they are **GENERATED** files using:

```
1 set_source_files_properties (${SOME_GENERATED_FILES}  
2                               PROPERTIES GENERATED TRUE)
```




The CMake workflow (pictured)





Example 1

```
1  ### Handle Source generation for task file parser
2  include_directories (${CMAKE_CURRENT_SOURCE_DIR})
3  find_package (LexYacc)
4  set (YACC_SRC           ${CMAKE_CURRENT_SOURCE_DIR}/lsmc_taskfile_syntax.yy)
5  set (YACC_OUT_PREFIX   ${CMAKE_CURRENT_BINARY_DIR}/y.tab)
6  set (YACC_WANTED_OUT_PREFIX ${CMAKE_CURRENT_BINARY_DIR}/lsmc_taskfile_syntax)
7  set (LEX_SRC           ${CMAKE_CURRENT_SOURCE_DIR}/lsmc_taskfile_tokens.ll)
8  set (LEX_OUT_PREFIX   ${CMAKE_CURRENT_BINARY_DIR}/lsmc_taskfile_tokens.yy)
9  set (LEX_WANTED_OUT_PREFIX ${CMAKE_CURRENT_BINARY_DIR}/lsmc_taskfile_tokens)
10
11 #Exec Lex
12 add_custom_command(
13     OUTPUT  ${LEX_WANTED_OUT_PREFIX}.c
14     COMMAND ${LEX_PROGRAM} ARGS -l -o${LEX_WANTED_OUT_PREFIX}.c ${LEX_SRC}
15     DEPENDS ${LEX_SRC}
16 )
17 set (GENERATED_SRCS ${GENERATED_SRCS} ${LEX_WANTED_OUT_PREFIX}.c)
18 #Exec Yacc
19 add_custom_command(
20     OUTPUT  ${YACC_WANTED_OUT_PREFIX}.c ${YACC_WANTED_OUT_PREFIX}.h
21     COMMAND ${YACC_PROGRAM} ARGS ${YACC_COMPAT_ARG} -d ${YACC_SRC}
22     COMMAND ${CMAKE_COMMAND} -E copy ${YACC_OUT_PREFIX}.h ${YACC_WANTED_OUT_PREFIX}.h
23     COMMAND ${CMAKE_COMMAND} -E copy ${YACC_OUT_PREFIX}.c ${YACC_WANTED_OUT_PREFIX}.c
24     DEPENDS ${YACC_SRC}
```



Example II

```
25 )
26 set(GENERATED_SRCS ${GENERATED_SRCS}
27   ${YACC_WANTED_OUT_PREFIX}.c ${YACC_WANTED_OUT_PREFIX}.h)
28 # Tell CMake that some file are generated
29 set_source_files_properties(${GENERATED_SRCS} PROPERTIES GENERATED TRUE)
30
31 # Inhibit compiler warning for LEX|YACC generated files
32 # Note that the inhibition is COMPILER dependent ...
33 # GNU CC specific warning stop
34 if (CMAKE_COMPILER_IS_GNUCC)
35   message(STATUS "INHIBIT_compiler_warning_for_LEX|YACC_generated_files")
36   SET_SOURCE_FILES_PROPERTIES(${YACC_WANTED_OUT_PREFIX}.c ${YACC_WANTED_OUT_PREFIX}.h
37     PROPERTIES COMPILE_FLAGS "-w")
38
39   SET_SOURCE_FILES_PROPERTIES(${LEX_WANTED_OUT_PREFIX}.c
40     PROPERTIES COMPILE_FLAGS "-w")
41 endif(CMAKE_COMPILER_IS_GNUCC)
42 ...
43 set(LSCHED_SRC
44   lsmc_dependency.c lsmc_core.c lsmc_utils.c
45   lsmc_time.c lsmc_taskfile_parser.c
46   ${GENERATED_SRCS})
47 add_library(lsmc ${LSCHED_SRC})
```



Outline

- 1 Basic CMake usage
- 2 **Discovering environment specificities**
 - Handling platform specificities
 - Working with external packages
- 3 **More CMake scripting**
 - Custom commands
 - Generated files
- 4 **Advanced CMake usage**
 - Cross-compiling with CMake
 - Export your project



Outline

- 1 Basic CMake usage
- 2 **Discovering environment specificities**
 - Handling platform specificities
 - Working with external packages
- 3 **More CMake scripting**
 - Custom commands
 - Generated files
- 4 **Advanced CMake usage**
 - Cross-compiling with CMake
 - Export your project



Cross-compiling

Definition: Cross-compiling

Cross-compiling is when the host system, the one the compiler is running on, is not the same as the target system, the one the compiled program will be running on.

CMake can handle cross-compiling using Toolchain see http://www.cmake.org/Wiki/CMake_Cross_Compiling.

```
1 mkdir build-win32
2 cd build-win32
3 cmake -DCMAKE_TOOLCHAIN_FILE=../totally-free/Toolchain-cross-mingw32-linux.cmake ../totally-free/
```

Demo



Linux to Win32 Toolchain example

```
1 # the name of the target operating system
2 SET(CMAKE_SYSTEM_NAME Windows)
3 # Choose an appropriate compiler prefix
4 # for classical mingw32 see http://www.mingw.org/
5 #set(COMPILER_PREFIX "i586-mingw32msvc")
6 # for 32 or 64 bits mingw-w64 see http://mingw-w64.sourceforge.net/
7 set(COMPILER_PREFIX "i686-w64-mingw32")
8 #set(COMPILER_PREFIX "x86_64-w64-mingw32")
9
10 # which compilers to use for C and C++
11 find_program(CMAKE_RC_COMPILER NAMES ${COMPILER_PREFIX}-windres)
12 #SET(CMAKE_RC_COMPILER ${COMPILER_PREFIX}-windres)
13 find_program(CMAKE_C_COMPILER NAMES ${COMPILER_PREFIX}-gcc)
14 #SET(CMAKE_C_COMPILER ${COMPILER_PREFIX}-gcc)
15 find_program(CMAKE_CXX_COMPILER NAMES ${COMPILER_PREFIX}-g++)
16 #SET(CMAKE_CXX_COMPILER ${COMPILER_PREFIX}-g++)
17
18 # here is the target environment located
19 SET(USER_ROOT_PATH /home/erk/erk-win32-dev)
20 SET(CMAKE_FIND_ROOT_PATH /usr/${COMPILER_PREFIX} ${USER_ROOT_PATH})
21 # adjust the default behaviour of the FIND_XXX() commands:
22 # search headers and libraries in the target environment, search
23 # programs in the host environment
24 set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
25 set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
26 set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```



Outline

- 1 Basic CMake usage
- 2 **Discovering environment specificities**
 - Handling platform specificities
 - Working with external packages
- 3 **More CMake scripting**
 - Custom commands
 - Generated files
- 4 **Advanced CMake usage**
 - Cross-compiling with CMake
 - Export your project



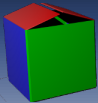
Exporting/Import your project

Export/Import to/from others

CMake can help project using CMake as a build system to export/import targets to/from other project using CMake as a build system.

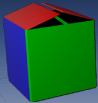
No more time for that today sorry, see:

http://www.cmake.org/Wiki/CMake/Tutorials/Exporting_and_Importing_Targets



Outline

- 5 **CPack: Packaging made easy**
- 6 CPack with CMake
- 7 Various package generators



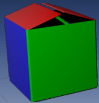
Introduction

A Package generator

In the same way as CMake generates build files, CPack generates packager files.

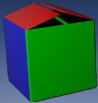
- Archive generators [ZIP,TGZ,...] (All platforms)
- DEB, RPM (Linux)
- Cygwin Source or Binary (Windows/Cygwin)
- NSIS (Windows, Linux)
- DragNDrop, Bundle, OSXX11 (MacOS)



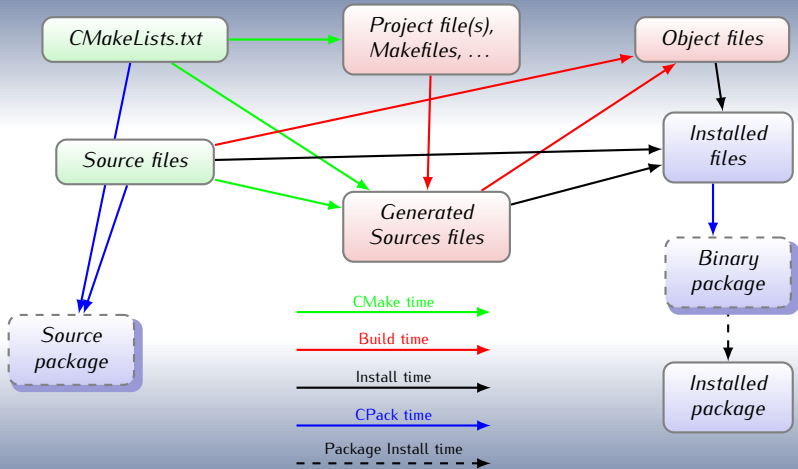


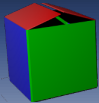
Outline

- 5 CPack: Packaging made easy
- 6 CPack with CMake**
- 7 Various package generators



The CMake workflow (pictured)





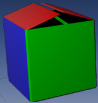
The CPack application

CPack standalone

CPack is a standalone application which behavior is driven by a configuration file e.g. `CPackConfig.cmake`. This file is a CMake language script which defines `CPACK_XXXX` variables: the config parameters of the CPack run.

CPack with CMake

When CPack is used to package a project built with CPack then the CPack configuration is usually generated by CMake by including `CPack.cmake` in the main `CMakeLists.txt`:
`include(CPack)`

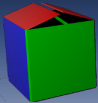


CPack variables in CMakeLists.txt

When used with CMake, one writes something like this in CMakeLists.txt:

```
1 set(CPACK_GENERATOR "TGZ")
2 if (WIN32)
3     list(APPEND CPACK_GENERATOR "NSIS")
4 elseif (APPLE)
5     list(APPEND CPACK_GENERATOR "Bundle")
6 endif(WIN32)
7 set(CPACK_SOURCE_GENERATOR "ZIP;TGZ")
8 set(CPACK_PACKAGE_VERSION_MAJOR 0)
9 set(CPACK_PACKAGE_VERSION_MINOR 1)
10 set(CPACK_PACKAGE_VERSION_PATCH 0)
11 include(CPack)
```

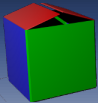
This will create CPackSourceConfig.cmake and CPackConfig.cmake in the build tree and will bring you the package and package_source built-in targets.



A CPack config file I

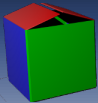
A CPack config file looks like this one:

```
1 # This file will be configured to contain variables for CPack.  
2 # These variables should be set in the CMake list file of the  
3 # project before CPack module is included.  
4 ...  
5 SET(CPACK_BINARY_BUNDLE "")  
6 SET(CPACK_BINARY_CYGWIN "")  
7 SET(CPACK_BINARY_DEB "")  
8 ...  
9 SET(CPACK_BINARY_ZIP "")  
10 SET(CPACK_CMAKE_GENERATOR "Unix_█Makefiles")  
11 SET(CPACK_GENERATOR "TGZ")  
12 SET(CPACK_INSTALL_CMAKE_PROJECTS "/home/erk/erkit/CMakeTutorial/  
    examples/build;TotallyFree;ALL;/")  
13 SET(CPACK_INSTALL_PREFIX "/usr/local")  
14 SET(CPACK_MODULE_PATH "")  
15 SET(CPACK_NSIS_DISPLAY_NAME "TotallyFree_█0.1.0")
```

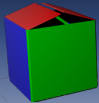
A CPack config file II

```
16 SET(CPACK_NSIS_INSTALLER_ICON_CODE "")
17 SET(CPACK_NSIS_INSTALL_ROOT "$PROGRAMFILES")
18 SET(CPACK_NSIS_PACKAGE_NAME "TotallyFree_0.1.0")
19 SET(CPACK_OUTPUT_CONFIG_FILE "/home/erk/erkit/CMakeTutorial/
    examples/build/CPackConfig.cmake")
20 SET(CPACK_PACKAGE_DEFAULT_LOCATION "/")
21 SET(CPACK_PACKAGE_DESCRIPTION_FILE "/home/erk/CMake/cmake-Verk-
    HEAD/share/cmake-2.8/Templates/CPack.GenericDescription.txt
    ")
22 SET(CPACK_PACKAGE_DESCRIPTION_SUMMARY "TotallyFree_ built_ using_
    CMake")
23 SET(CPACK_PACKAGE_FILE_NAME "TotallyFree-0.1.0-Linux")
24 SET(CPACK_PACKAGE_INSTALL_DIRECTORY "TotallyFree_0.1.0")
25 SET(CPACK_PACKAGE_INSTALL_REGISTRY_KEY "TotallyFree_0.1.0")
26 SET(CPACK_PACKAGE_NAME "TotallyFree")
27 SET(CPACK_PACKAGE_RELOCATABLE "true")
28 SET(CPACK_PACKAGE_VENDOR "Humanity")
29 SET(CPACK_PACKAGE_VERSION "0.1.0")
```



A CPack config file III

```
30 SET(CPACK_RESOURCE_FILE_LICENSE "/home/erk/CMake/cmake-Verk-HEAD
    /share/cmake-2.8/Templates/CPack.GenericLicense.txt")
31 SET(CPACK_RESOURCE_FILE_README "/home/erk/CMake/cmake-Verk-HEAD/
    share/cmake-2.8/Templates/CPack.GenericDescription.txt")
32 SET(CPACK_RESOURCE_FILE_WELCOME "/home/erk/CMake/cmake-Verk-HEAD
    /share/cmake-2.8/Templates/CPack.GenericWelcome.txt")
33 SET(CPACK_SET_DESTDIR "OFF")
34 SET(CPACK_SOURCE_CYGWIN "")
35 SET(CPACK_SOURCE_GENERATOR "TGZ;TBZ2;TZ")
36 SET(CPACK_SOURCE_OUTPUT_CONFIG_FILE "/home/erk/erkit/
    CMakeTutorial/examples/build/CPackSourceConfig.cmake")
37 SET(CPACK_SOURCE_TBZ2 "ON")
38 SET(CPACK_SOURCE_TGZ "ON")
39 SET(CPACK_SOURCE_TZ "ON")
40 SET(CPACK_SOURCE_ZIP "OFF")
41 SET(CPACK_SYSTEM_NAME "Linux")
42 SET(CPACK_TOPLEVEL_TAG "Linux")
```



CPack running steps I

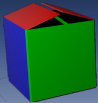
For a CMake enabled project one can run CPack in two ways:

- 1 use the build tool to run targets: `package` or `package_source`
- 2 invoke CPack manually from within the build tree e.g.:

```
$ cpack -G RPM
```

Currently `cpack` has [almost] no builtin documentation support besides `cpack --help` (work is underway though), thus the best CPack documentation is currently found on the Wiki:

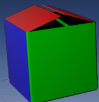
- <http://www.cmake.org/Wiki/CMake:CPackConfiguration>
- <http://www.cmake.org/Wiki/CMake:CPackPackageGenerators>
- http://www.cmake.org/Wiki/CMake:Component_Install_With_CPack



CPack running steps II

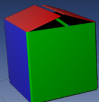
Whichever way you call it, the CPack steps are:

- 1 cpack command starts and parses arguments etc...
- 2 it reads `CPackConfig.cmake` (usually found in the build tree) or the file given as an argument to `--config` command line option.
- 3 it iterates over the generators list found in `CPACK_GENERATOR` (or from `-G` command line option). For each generator:
 - 3 (re)sets `CPACK_GENERATOR` to the one currently being iterated over
 - 3 includes the `CPACK_PROJECT_CONFIG_FILE`
 - 3 installs the project into CPack private location (using `DESTDIR`)
 - 3 calls the generator and produces the package(s) for that generator



CPack running steps III

```
_____ cpack command line example _____  
1  $ cpack -G "TGZ;RPM"  
2  CPack: Create package using TGZ  
3  CPack: Install projects  
4  CPack: - Run preinstall target for: TotallyFree  
5  CPack: - Install project: TotallyFree  
6  CPack: Create package  
7  CPack: - package: <...>/build/TotallyFree-0.1.0-Linux.tar.gz generated.  
8  CPack: Create package using RPM  
9  CPack: Install projects  
10 CPack: - Run preinstall target for: TotallyFree  
11 CPack: - Install project: TotallyFree  
12 CPack: Create package  
13   CPackRPM: Will use GENERATED spec file: <...>/build/_CPack_Packages/Linux/RPM/SPECS/totallyfree.spec  
14 CPack: - package: <...>/build/TotallyFree-0.1.0-Linux.rpm generated.  
15  $
```



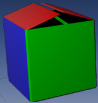
CPack running steps IV

make package example

```
1 $ make package
2 [ 33%] Built target acrodicT
3 [ 66%] Built target AcrodicTlibre
4 [100%] Built target Acrolibre
5 Run CPack packaging tool...
6 CPack: Create package using TGZ
7 CPack: Install projects
8 CPack: - Run preinstall target for: TotallyFree
9 CPack: - Install project: TotallyFree
10 CPack: Create package
11 CPack: - package: <...>/build/TotallyFree-0.1.0-Linux.tar.gz generated.
```

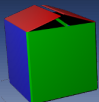
Rebuild project

In the `make package` case CMake is checking that the project does not need a rebuild.



CPack running steps V

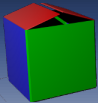
```
_____ make package_source example _____  
1  $ make package_source  
2  make package_source  
3  Run CPack packaging tool for source...  
4  CPack: Create package using TGZ  
5  CPack: Install projects  
6  CPack: - Install directory: <...>/totally-free  
7  CPack: Create package  
8  CPack: - package: <...>/build/TotallyFree-0.1.0-Source.tar.gz generated.  
9  CPack: Create package using TBZ2  
10 CPack: Install projects  
11 CPack: - Install directory: <...>/totally-free  
12 CPack: Create package  
13 CPack: - package: <...>/build/TotallyFree-0.1.0-Source.tar.bz2 generated.  
14 CPack: Create package using TZ  
15 CPack: Install projects  
16 CPack: - Install directory: <...>/totally-free  
17 CPack: Create package  
18 CPack: - package: <...>/build/TotallyFree-0.1.0-Source.tar.Z generated.
```



The CPack workflow (pictured)

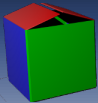
CMakeLists.txt

Source files

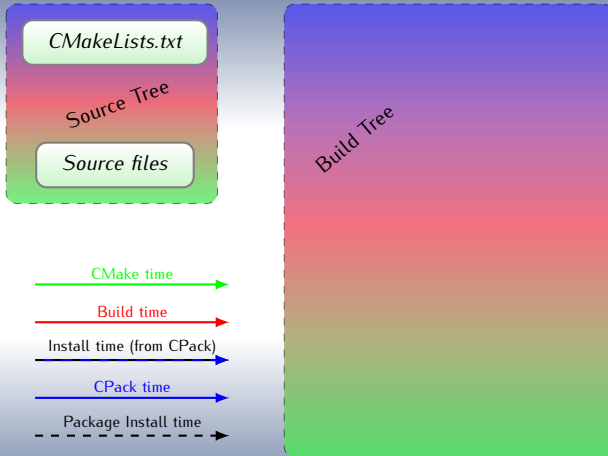


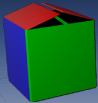
The CPack workflow (pictured)



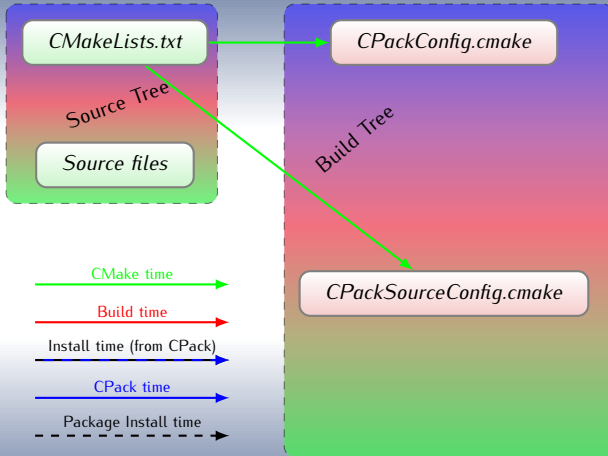


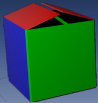
The CPack workflow (pictured)



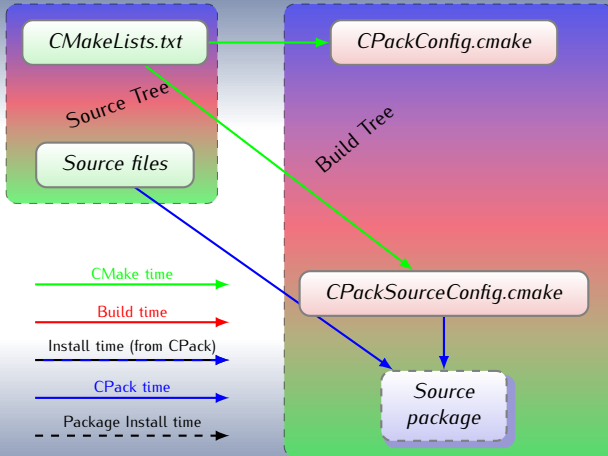


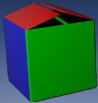
The CPack workflow (pictured)



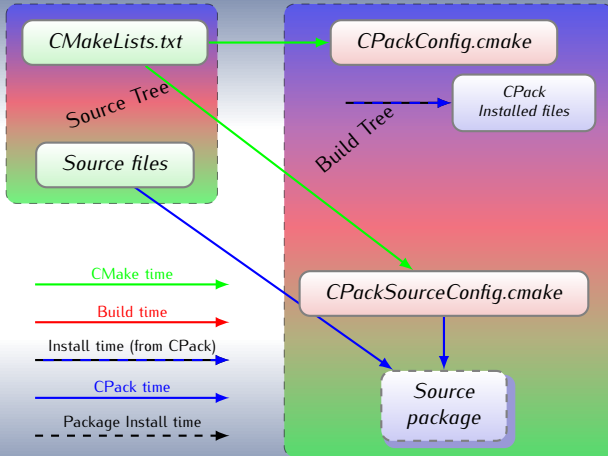


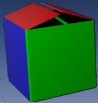
The CPack workflow (pictured)



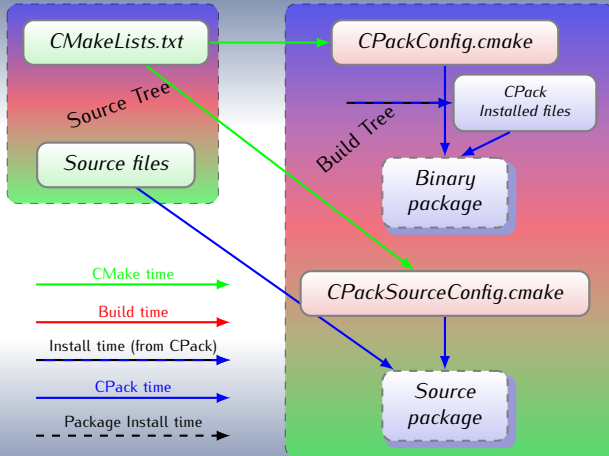


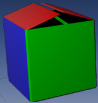
The CPack workflow (pictured)



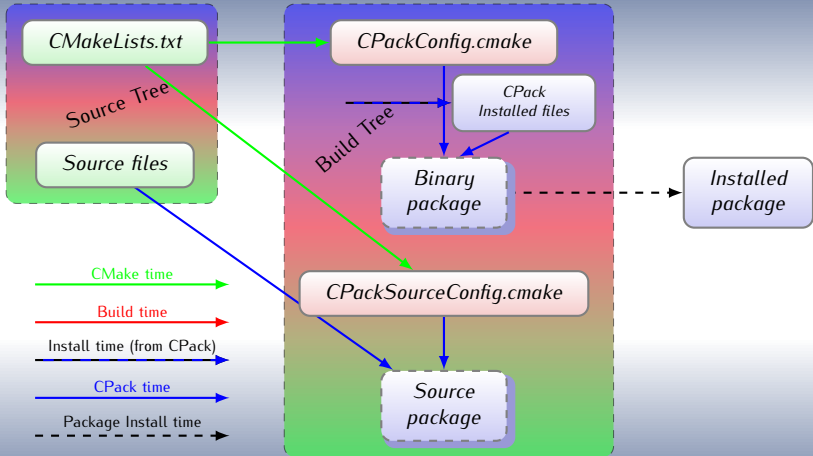


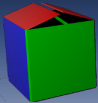
The CPack workflow (pictured)





The CPack workflow (pictured)





Source vs Binary Generators

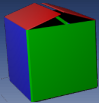
CPack does not really distinguish “source” from “binaries”!!

CPack source package

The CPack configuration file is: `CPackSourceConfig.cmake`. The CPack source generator is essentially packaging directories with install, exclude and include rules.

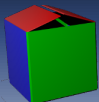
CPack binary package

The CPack configuration file is: `CPackConfig.cmake`. Moreover CPack knows that a project is built with CMake and inherits many properties from the install rules found in the project.



Outline

- 5 CPack: Packaging made easy
- 6 CPack with CMake
- 7 Various package generators**



Archive Generators

A family of generators

The archive generators is a family of generators which is supported on all CMake supported platforms through libarchive:
<http://code.google.com/p/libarchive/>.

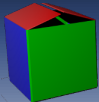
STGZ Self extracting Tar GZip compression

TBZ2 Tar BZip2 compression

TGZ Tar GZip compression

TZ Tar Compress compression

ZIP Zip archive



Linux-friendly generators

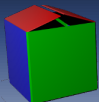
- Tar-kind archive generators
- Binary RPM: only needs `rpmbuild` to work.
- Binary DEB: works on any Linux distros.

CPack vs native tools

One could argue “why using CPack for building `.deb` or `.rpm`”. The primary target of CPack RPM and DEB generators are people who are NOT professional packager. Those people can get a clean package without too much effort and get better package than a bare TAR archive.

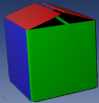
No official packaging replacement

Those generators are **no replacement** for official packaging tools.



Windows-friendly generators

- Zip archive generator
- NullSoft System Installer generator
(<http://nsis.sourceforge.net/>
Support component installation, produce nice GUI installer.
- MSI installer requested:
<http://public.kitware.com/Bug/view.php?id=11575>.
- Cygwin: Binary and Source generators.



MacOS-friendly generators

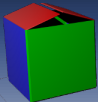
- Tar-kind archive generators
- DragNDrop
- PackageMaker
- Bundle
- OSXX11

Don't ask me

I'm not a MacOS user and I don't know them. Go and read the Wiki or ask on the ML.

[http://www.cmake.org/Wiki/CMake:
CPackPackageGenerators](http://www.cmake.org/Wiki/CMake:CPackPackageGenerators)

<http://www.cmake.org/cmake/help/mailing.html>

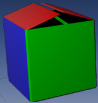


Packaging Components I

CMake+CPack installation components?

Sometimes you want to split the installer into components.

- 1 Use COMPONENT argument in your install rules (in the CMakeLists.txt),
- 2 Add some more [CPack] information about how to group components,
- 3 Choose a component-aware CPack generator
- 4 Choose the behavior (1 package file per component, 1 package file per group, etc...)
- 5 Possibly specify generator specific behavior in CPACK_PROJECT_CONFIG_FILE
- 6 Run CPack.



Packaging Components II

demo with ComponentExample

More detailed documentation here:

http://www.cmake.org/Wiki/CMake:Component_Install_With_CPack

Component aware generator

- Not all generators do support component (i.e. they are MONOLITHIC)
- Some produce a single package file containing all components. (e.g. NSIS)
- Others produce several package files containing one or several components. (e.g. ArchiveGenerator, RPM, DEB)



Outline

- 8 **Systematic Testing**
- 9 CTest submission to CDash
- 10 References



Outline

8 Systematic Testing

9 **CTest submission to CDash**

10 References



More to come on CTest/CDash

Sorry...out of time!!

CMake and its friends are so much fun and powerful that I ran out of time to reach a detailed presentation of CTest/CDash, stay tuned for next time...

In the meantime:

- Go there: <http://www.cdash.org>
- Open your own (free) Dashboard: <http://my.cdash.org/>
- CDash 2.0 should be released in the next few weeks (mid-february)
- A course on CMake/CTest/CDash in Lyon on April, 2 2012 (<http://formations.kitware.fr>)



Outline

8 Systematic Testing

9 CTest submission to CDash

10 **References**



References I



CDash home page, Feb. 2011.

<http://www.cdash.org>.



CMake home page, Feb. 2012.

<http://www.cmake.org>.



CMake Wiki, Feb. 2012.

<http://www.cmake.org/Wiki/CMake>.



Development/CMake on KDE TechBase, Feb. 2012.

<http://techbase.kde.org/Development/CMake>.



Ken Martin and Bill Hoffman.

Mastering CMake: A Cross-Platform Build System.

Kitware, Inc., 5th edition edition, 2010.

ISBN-13 978-1930934221.