Appendix B – Command Line Reference

CMake Command Line Options

```
cmake [options] <path-to-source>
cmake [options] <path-to-existing-build>
```

The "cmake" executable is the CMake command-line interface. It may be used to configure projects in scripts. Project configuration settings may be specified on the command line with the -D option. The -i option will cause cmake to interactively prompt for such settings.

-C <initial-cache>: Pre-load a script to populate the cache.

When cmake is first run in an empty build tree, it creates a CMakeCache.txt file and populates it with customizable settings for the project. This option may be used to specify a file from which to load cache entries before the first pass through the project's cmake listfiles. The loaded entries take priority over the project's default values. The given file should be a CMake script containing SET commands that use the CACHE option, not a cache-format file.

-D <var>:<type>=<value>: Create a cmake cache entry.

When cmake is first run in an empty build tree, it creates a CMakeCache.txt file and populates it with customizable settings for the project. This option may be used to specify a setting that takes priority over the project's default value. The option may be repeated for as many cache entries as desired.

-U <globbing_expr>: Remove matching entries from CMake cache.

This option may be used to remove one or more variables from the CMakeCache.txt file, globbing expressions using * and ? are supported. The option may be repeated for as many cache entries as desired.

Use with care, you can make your CMakeCache.txt non-working.

-G <generator-name>: Specify a Makefile generator.

CMake may support multiple native build systems on certain platforms. A Makefile generator is responsible for generating a particular build system. Possible generator names are specified in the Generators section.

-Wno-dev: Suppress developer warnings.

Suppress warnings that are meant for the author of the CMakeLists.txt files.

-Wdev: Enable developer warnings.

Enable warnings that are meant for the author of the CMakeLists.txt files.

-E: CMake command mode.

For true platform independence, CMake provides a list of commands that can be used on all systems. Run with -E help for the usage information. Commands available are: chdir, copy, copy_if_different copy_directory, compare_files, echo, echo_append, environment, make_directory, md5sum, remove_directory, remove, tar, time, touch, touch_nocreate, write_regv, delete_regv, comspec, create_symlink.

-i: Run in wizard mode.

Wizard mode runs cmake interactively without a GUI. The user is prompted to answer questions about the project configuration. The answers are used to set cmake cache values.

-L[A][H]: List non-advanced cached variables.

List cache variables will run CMake and list all the variables from the CMake cache that are not marked as INTERNAL or ADVANCED. This will effectively display current CMake settings, which can be then changed with -D option. Changing some of the variable may result in more variables being created. If A is specified, then it will display also advanced variables. If H is specified, it will also display help for each variable.

--build <dir>: Build a CMake-generated project binary tree.

This abstracts a native build tool's command-line interface with the following options:

```
<dir>          = Project binary directory to be built.
--target <tgt> = Build <tgt> instead of default targets.
--config <cfg> = For multi-configuration tools, choose <cfg>.
--clean-first  = Build target 'clean' first, then build.
                 (To clean only, use --target 'clean'.)
--             = Pass remaining options to the native tool.
```

Run cmake --build with no options for quick help.

-N: View mode only.

Only load the cache. Do not actually run configure and generate steps.

-P <file>: Process script mode.

Process the given cmake file as a script written in the CMake language. No configure or generate step is performed and the cache is not modified. If variables are defined using -D, this must be done before the -P argument.

--graphviz=[file]: Generate graphviz of dependencies.

Generate a graphviz input file that will contain all the library and executable dependencies in the project.

--system-information [file]: Dump information about this system.

Dump a wide range of information about the current system. If run from the top of a binary tree for a CMake project it will dump additional information such as the cache, log files etc.

--debug-trycompile: Do not delete the try compile directories..

Do not delete the files and directories created for try_compile calls. This is useful in debugging failed try_compiles. It may however change the results of the try-compiles as old junk from a previous try-compile may cause a different test to either pass or fail incorrectly. This option is best used for one try-compile at a time, and only when debugging.

--debug-output: Put cmake in a debug mode.

Print extra stuff during the cmake run like stack traces with message(send_error) calls.

--trace: Put cmake in trace mode.

Print a trace of all calls made and from where with message(send_error) calls.

--help-command cmd [file]: Print help for a single command and exit.

Full documentation specific to the given command is displayed. If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-command-list [file]: List available listfile commands and exit.

The list contains all commands for which help may be obtained by using the --help-command argument followed by a command name. If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-commands [file]: Print help for all commands and exit.

Full documentation specific for all current command is displayed.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-compatcommands [file]: Print help for compatibility commands.

Full documentation specific for all compatibility commands is displayed.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-module module [file]: Print help for a single module and exit.

Full documentation specific to the given module is displayed.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-module-list [file]: List available modules and exit.

The list contains all modules for which help may be obtained by using the --help-module argument followed by a module name. If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-modules [file]: Print help for all modules and exit.

Full documentation for all modules is displayed. If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-custom-modules [file]: Print help for all custom modules and exit.

Full documentation for all custom modules is displayed. If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-policy cmp [file]: Print help for a single policy and exit.

Full documentation specific to the given policy is displayed.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-policies [file]: Print help for all policies and exit.

Full documentation for all policies is displayed.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-property prop [file]: Print help for a single property and exit.

Full documentation specific to the given property is displayed.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-property-list [file]: List available properties and exit.

The list contains all properties for which help may be obtained by using the --help-property argument followed by a property name. If a file is specified, the help is written into it.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-properties [file]: Print help for all properties and exit.

Full documentation for all properties is displayed.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-variable var [file]: Print help for a single variable and exit.

Full documentation specific to the given variable is displayed.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-variable-list [file]: List documented variables and exit.

The list contains all variables for which help may be obtained by using the --help-variable argument followed by a variable name. If a file is specified, the help is written into it.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--help-variables [file]: Print help for all variables and exit.

Full documentation for all variables is displayed.If a file is specified, the documentation is written into and the output format is determined depending on the filename suffix. Supported are man page, HTML, DocBook and plain text.

--copyright [file]: Print the CMake copyright and exit.

If a file is specified, the copyright is written into it.

--help: Print usage information and exit.

Usage describes the basic command line interface and its options.

--help-full [file]: Print full help and exit.

Full help displays most of the documentation provided by the UNIX man page. It is provided for use on non-UNIX platforms, but is also convenient if the man page is not installed. If a file is specified, the help is written into it.

--help-html [file]: Print full help in HTML format.

This option is used by CMake authors to help produce web pages. If a file is specified, the help is written into it.

--help-man [file]: Print full help as a UNIX man page and exit.

This option is used by the cmake build to generate the UNIX man page. If a file is specified, the help is written into it.

--version [file]: Show program name/version banner and exit.

If a file is specified, the version is written into it.

CMake Generators

The following generators are available within CMake:

Borland Makefiles: Generates Borland Makefiles.

MSYS Makefiles: Generates MSYS Makefiles.

The Makefiles use /bin/sh as the shell. They require msys to be installed on the machine.

MinGW Makefiles: Generates a Makefile for use with mingw32-make.

The Makefiles generated use cmd.exe as the shell. They do not require msys or a UNIX shell.

NMake Makefiles: Generates NMake Makefiles.

Unix Makefiles: Generates standard UNIX Makefiles.

A hierarchy of UNIX Makefiles is generated into the build tree. Any standard UNIX-style make program can build the project through the default make target. A "make install" target is also provided.

Visual Studio 10: Generates Visual Studio 10 project files.

Visual Studio 6: Generates Visual Studio 6 project files.

Visual Studio 7: Generates Visual Studio .NET 2002 project files.

Visual Studio 7 .NET 2003: Generates Visual Studio .NET 2003 project files.

Visual Studio 8 2005: Generates Visual Studio .NET 2005 project files.

Visual Studio 8 2005 Win64: Generates Visual Studio .NET 2005 Win64 project files.

Visual Studio 9 2008: Generates Visual Studio 9 2008 project files.

Visual Studio 9 2008 Win64: Generates Visual Studio 9 2008 Win64 project files.

Watcom WMake: Generates Watcom WMake Makefiles.

CodeBlocks - MinGW Makefiles: Generates CodeBlocks project files.

Project files for CodeBlocks will be created in the top directory and in every subdirectory which features a CMakeLists.txt file containing a PROJECT() call. Additionally a hierarchy of Makefiles is generated into the build tree. The appropriate make program can build the project through the default make target. A "make install" target is also provided.

CodeBlocks - NMake Makefiles: Generates CodeBlocks project files.

Project files for CodeBlocks will be created in the top directory and in every subdirectory which features a CMakeLists.txt file containing a PROJECT() call. Additionally a hierarchy of Makefiles is generated into the build tree. The appropriate make program can build the project through the default make target. A "make install" target is also provided.

CodeBlocks - Unix Makefiles: Generates CodeBlocks project files.

Project files for CodeBlocks will be created in the top directory and in every subdirectory which features a CMakeLists.txt file containing a PROJECT() call. Additionally a hierarchy of Makefiles is generated into the build tree. The appropriate make program can build the project through the default make target. A "make install" target is also provided.

Eclipse CDT4 - MinGW Makefiles: Generates Eclipse CDT 4.0 project files.

Project files for Eclipse will be created in the top directory and will have a linked resource to every subdirectory which features a CMakeLists.txt file containing a PROJECT() call.Additionally a hierarchy of Makefiles is generated into the build tree. The appropriate make program can build the project through the default make target. A "make install" target is also provided.

Eclipse CDT4 - NMake Makefiles: Generates Eclipse CDT 4.0 project files.

Project files for Eclipse will be created in the top directory and will have a linked resource to every subdirectory which features a CMakeLists.txt file containing a PROJECT() call.Additionally a hierarchy of Makefiles is generated into the build tree. The appropriate

make program can build the project through the default make target. A "make install" target is also provided.

Eclipse CDT4 - Unix Makefiles: Generates Eclipse CDT 4.0 project files.

Project files for Eclipse will be created in the top directory and will have a linked resource to every subdirectory which features a CMakeLists.txt file containing a PROJECT() call.Additionally a hierarchy of Makefiles is generated into the build tree. The appropriate make program can build the project through the default make target. A "make install" target is also provided.

CTest Command Line Options

```
ctest [options]
```

The "ctest" executable is the CMake test driver program. CMake-generated build trees created for projects that use the ENABLE_TESTING and ADD_TEST commands have testing support. This program will run the tests and report results.

-C <cfg>, --build-config <cfg>: Choose configuration to test.

Some CMake-generated build trees can have multiple build configurations in the same tree. This option can be used to specify which one should be tested. Example configurations are "Debug" and "Release".

-V,--verbose: Enable verbose output from tests.

Test output is normally suppressed and only summary information is displayed. This option will show all test output.

-VV,--extra-verbose: Enable more verbose output from tests.

Test output is normally suppressed and only summary information is displayed. This option will show even more test output.

--debug: Displaying more verbose internals of CTest.

This feature will result in large number of output that is mostly useful for debugging dashboard problems.

--output-on-failure: Output anything outputted by the test program if the test should fail. This option can also be enabled by setting the environment variable CTEST_OUTPUT_ON_FAILURE

-F: Enable failover.

This option allows ctest to resume a test set execution that was previously interrupted. If no interruption occurred, the -F option will have no effect.

-Q,--quiet: Make ctest quiet.

This option will suppress all the output. The output log file will still be generated if the --output-log is specified. Options such as --verbose, --extra-verbose, and --debug are ignored if --quiet is specified.

-O <file>, --output-log <file>: Output to log file

This option tells ctest to write all its output to a log file.

-N,--show-only: Disable actual execution of tests.

This option tells ctest to list the tests that would be run but not actually run them. Useful in conjunction with the -R and -E options.

-L <regex>, --label-regex <regex>: Run tests with labels matching regular expression.

This option tells ctest to run only the tests whose labels match the given regular expression.

-R <regex>, --tests-regex <regex>: Run tests matching regular expression.

This option tells ctest to run only the tests whose names match the given regular expression.

-E <regex>, --exclude-regex <regex>: Exclude tests matching regular expression.

This option tells ctest to NOT run the tests whose names match the given regular expression.

-LE <regex>, --label-exclude <regex>: Exclude tests with labels matching regular expression.

This option tells ctest to NOT run the tests whose labels match the given regular expression.

-D <dashboard>, --dashboard <dashboard>: Execute dashboard test

This option tells ctest to perform act as a Dart client and perform a dashboard test. All tests are <Mode><Test>, where Mode can be Experimental, Nightly, and Continuous, and Test can be Start, Update, Configure, Build, Test, Coverage, and Submit.

-M <model>, --test-model <model>: Sets the model for a dashboard

This option tells ctest to act as a Dart client where the TestModel can be Experimental, Nightly, and Continuous. Combining -M and -T is similar to -D

-T <action>, --test-action <action>: Sets the dashboard action to perform

This option tells ctest to act as a Dart client and perform some action such as start, build, test etc. Combining -M and -T is similar to -D

--track <track>: Specify the track to submit dashboard to

Submit dashboard to specified track instead of default one. By default, the dashboard is submitted to Nightly, Experimental, or Continuous track, but by specifying this option, the track can be arbitrary.

-S <script>, --script <script>: Execute a dashboard for a configuration

This option tells ctest to load in a configuration script which sets a number of parameters such as the binary and source directories. Then ctest will do what is required to create and run a dashboard. This option basically sets up a dashboard and then runs ctest -D with the appropriate options.

-SP <script>, --script-new-process <script>: Execute a dashboard for a configuration

This option does the same operations as -S but it will do them in a seperate process. This is primarily useful in cases where the script may modify the environment and you do not want the modified enviroment to impact other -S scripts.

-A <file>, --add-notes <file>: Add a notes file with submission

This option tells ctest to include a notes file when submitting dashboard.

-I [Start,End,Stride,test#,test#|Test file], --tests-information: Run a specific number of tests by number.

This option causes ctest to run tests starting at number Start, ending at number End, and incrementing by Stride. Any additional numbers after Stride are considered individual test numbers. Start, End,or stride can be empty. Optionally a file can be given that contains the same syntax as the command line.

-U, --union: Take the Union of -I and -R

When both -R and -I are specified by default the intersection of tests are run. By specifying -U the union of tests is run instead.

--max-width <width>: Set the max width for a test name to output

Set the maximum width for each test name to show in the output. This allows the user to widen the output to avoid cliping the test name which can be very annoying.

--interactive-debug-mode [0|1]: Set the interactive mode to 0 or 1.

This option causes ctest to run tests in either an interactive mode or a non-interactive mode. On Windows this means that in non-interactive mode, all system debug pop up windows are blocked. In dashboard mode (Experimental, Nightly, Continuous), the default is non-interactive. When just running tests not for a dashboard the default is to allow popups and interactive debugging.

--no-label-summary: Disable timing summary information for labels.

This option tells ctest to not print summary information for each label associated with the tests run. If there are no labels on the tests, nothing extra is printed.

--build-and-test: Configure, build and run a test.

This option tells ctest to configure (i.e. run cmake on), build, and or execute a test. The configure and test steps are optional. The arguments to this command line are the source and binary directories. By default this will run CMake on the Source/Bin directories specified unless --build-nocmake is specified. Both --build-makeprogram and --build-generator MUST be provided to use --built-and-test. If --test-command is specified then that will be run after the build is complete. Other options that affect this mode are --build-target --build-nocmake, --build-run-dir, --build-two-config, --build-exe-dir, --build-project,--build-noclean, --build-options

--build-target: Specify a specific target to build.

This option goes with the --build-and-test option, if left out the all target is built.

--build-nocmake: Run the build without running cmake first.

Skip the cmake step.

--build-run-dir: Specify directory to run programs from.

Directory where programs will be after it has been compiled.

--build-two-config: Run CMake twice

--build-exe-dir: Specify the directory for the executable.

--build-generator: Specify the generator to use.

--build-project: Specify the name of the project to build.

--build-makeprogram: Specify the make program to use.

--build-noclean: Skip the make clean step.

--build-config-sample: A sample executable to use to determine the configuraiton

A sample executable to use to determine the configuraiton that should be used. e.g. Debug/Release/etc

--build-options: Add extra options to the build step.

This option must be the last option with the exception of --test-command

--test-command: The test to run with the --build-and-test option.

--test-timeout: The time limit in seconds, internal use only.

--tomorrow-tag: Nightly or experimental starts with next day tag.

This is useful if the build will not finish in one day.

--ctest-config: The configuration file used to initialize CTest state when submitting dashboards.

This option tells CTest to use different initialization file instead of CTestConfiguration.tcl. This way multiple initialization files can be used for example to submit to multiple dashboards.

--overwrite: Overwrite CTest configuration option.

By default ctest uses configuration options from configuration file. This option will overwrite the configuration option.

--extra-submit <file>[;<file>]: Submit extra files to the dashboard.

This option will submit extra files to the dashboard.

--force-new-ctest-process: Run child CTest instances as new processes

By default CTest will run child CTest instances within the same process. If this behavior is not desired, this argument will enforce new processes for child CTest processes.

--submit-index: Submit individual dashboard tests with specific index

This option allows performing the same CTest action (such as test) multiple times and submit all stages to the same dashboard (Dart2 required). Each execution requires different index.

CPack Command Line Options

```
cpack -G <generator> [options]
```

The "cpack" executable is the CMake packaging program. CMake-generated build trees created for projects that use the INSTALL_* commands have packaging support. This program will generate the package.

-G <generator>: Use the specified generator to generate package.

CPack may support multiple native packaging systems on certain platforms. A generator is responsible for generating input files for particular system and invoking that systems. Possible generator names are specified in the Generators section.

-C <Configuration>: Specify the project configuration

This option specifies the configuration that the project was build with, for example 'Debug', 'Release'.

-D <var>=<value>: Set a CPack variable.

Set a variable that can be used by the generator.

--config <config file>: Specify the config file.

Specify the config file to use to create the package. By default CPackConfig.cmake in the current directory will be used.

CPack Generators

NSIS: Null Soft Installer

STGZ: Self extracting Tar GZip compression

TBZ2: Tar BZip2 compression

TGZ: Tar GZip compression

TZ: Tar Compress compression

ZIP: ZIP file format