Appendix A - Variables

Variables That Change Behavior

BUILD_SHARED_LIBS: Global flag to cause add_library to create shared libraries if on.

If present and true, this will cause all libraries to be built shared unless the library was explicitly added as a static library. This variable is often added to projects as an OPTION so that each user of a project can decide if they want to build the project using shared or static libraries.

CMAKE_BACKWARDS_COMPATIBILITY: Version of cmake required to build project

From the point of view of backwards compatibility, this specifies what version of CMake should be supported. By default this value is the version number of CMake that you are running. You can set this to an older version of CMake to support deprecated commands of CMake in projects that were written to use older versions of CMake. This can be set by the user or set at the beginning of a CMakeLists file.

CMAKE_BUILD_TYPE: Specifies the build type for make based generators.

This specifies what build type will be built in this tree. Possible values are empty, Debug, Release, RelWithDebInfo and MinSizeRel. This variable is only supported for make based generators. If this variable is supported, then CMake will also provide initial values for the variables with the name CMAKE_C_FLAGS_[Debug|Release|RelWithDebInfo|MinSizeRel]. For example, if CMAKE_BUILD_TYPE is Debug, then CMAKE_C_FLAGS_DEBUG will be added to the CMAKE_C_FLAGS.

CMAKE_COLOR_MAKEFILE: Enables color output when using the Makefile generator.

When enabled, the generated Makefiles will produce colored output. Default is ON.

CMAKE_CONFIGURATION_TYPES: Specifies the available build types.

This specifies what build types will be available such as Debug, Release, RelWithDebInfo etc. This has reasonable defaults on most platforms. But can be extended to provide other build types. See also CMAKE_BUILD_TYPE.

CMAKE_FIND_LIBRARY_PREFIXES: Prefixes to prepend when looking for libraries.

This specifies what prefixes to add to library names when the find_library command looks for libraries. On UNIX systems this is typically lib, meaning that when trying to find the foo library it will look for libfoo.

CMAKE_FIND_LIBRARY_SUFFIXES: Suffixes to append when looking for libraries.

This specifies what suffixes to add to library names when the find_library command looks for libraries. On Windows systems this is typically .lib and .dll, meaning that when trying to find the foo library it will look for foo.dll etc.

CMAKE_INCLUDE_PATH: Path used for searching by FIND_FILE() and FIND_PATH().

Specifies a path which will be used both by FIND_FILE() and FIND_PATH(). Both commands will check each of the contained directories for the existence of the file which is currently searched. By default it is empty, it is intended to be set by the project. See also CMAKE_SYSTEM_INCLUDE_PATH, CMAKE_PREFIX_PATH.

CMAKE_INSTALL_PREFIX: Install directory used by install.

If "make install" is invoked or INSTALL is built, this directory is pre-pended onto all install directories. This variable defaults to /usr/local on UNIX and c:/Program Files on Windows.

CMAKE_LIBRARY_PATH: Path used for searching by FIND_LIBRARY().

Specifies a path which will be used by FIND_LIBRARY(). FIND_LIBRARY() will check each of the contained directories for the existence of the library which is currently searched. By default it is empty, it is intended to be set by the project. See also CMAKE_SYSTEM_LIBRARY_PATH, CMAKE_PREFIX_PATH.

CMAKE_MFC_FLAG: Tell CMake to use MFC for an executable or dll.

This can be set in a CMakeLists.txt file and will enable MFC in the application. It should be set to 1 for static the static MFC library, and 2 for the shared MFC library. This is used in visual studio 6 and 7 project files. The CMakeSetup dialog used MFC and the CMakeLists.txt looks like this:

```
add_definitions(-D_AFXDLL)
set(CMAKE_MFC_FLAG 2)
add_executable(CMakeSetup WIN32 ${SRCS})
```

CMAKE_MODULE_PATH: Path to look for cmake modules to load.

Specifies a path to override the default search path for CMake modules. For example include commands will look in this path first for modules to include.

CMAKE_NOT_USING_CONFIG_FLAGS: Skip _BUILD_TYPE flags if true.

This is an internal flag used by the generators in CMake to tell CMake to skip the _BUILD_TYPE flags.

CMAKE_PREFIX_PATH: Path used for searching by FIND_XXX(), with appropriate suffixes added.

Specifies a path which will be used by the FIND_XXX() commands. It contains the "base" directories, the FIND_XXX() commands append appropriate subdirectories to the base directories. So FIND_PROGRAM() adds /bin to each of the directories in the path, FIND_LIBRARY() appends /lib to each of the directories, and FIND_PATH() and FIND_FILE() append /include . By default it is empty, it is intended to be set by the project. See also CMAKE_SYSTEM_PREFIX_PATH, CMAKE_INCLUDE_PATH, CMAKE_LIBRARY_PATH, CMAKE_PROGRAM_PATH.

CMAKE_PROGRAM_PATH: Path used for searching by FIND_PROGRAM().

Specifies a path which will be used by FIND_PROGRAM(). FIND_PROGRAM() will check each of the contained directories for the existence of the program which is currently searched. By default it is empty, it is intended to be set by the project. See also CMAKE_SYSTEM_PROGRAM_PATH, CMAKE_PREFIX_PATH.

CMAKE_SKIP_INSTALL_ALL_DEPENDENCY: Don't make the install target depend on the all target.

By default, the "install" target depends on the "all" target. This has the effect, that when "make install" is invoked or INSTALL is built, first the "all" target is built, then the installation starts. If CMAKE_SKIP_INSTALL_ALL_DEPENDENCY is set to TRUE, this dependency is not created, so the installation process will start immediately, independent from whether the project has been completely built or not.

CMAKE_SYSTEM_INCLUDE_PATH: Path used for searching by FIND_FILE() and FIND_PATH().

Specifies a path which will be used both by FIND_FILE() and FIND_PATH(). Both commands will check each of the contained directories for the existence of the file which is currently searched. By default it contains the standard directories for the current system. It is

NOT intended to be modified by the project, use CMAKE_INCLUDE_PATH for this. See also CMAKE_SYSTEM_PREFIX_PATH.

CMAKE_SYSTEM_LIBRARY_PATH: Path used for searching by FIND_LIBRARY().

Specifies a path which will be used by FIND_LIBRARY(). FIND_LIBRARY() will check each of the contained directories for the existence of the library which is currently searched. By default it contains the standard directories for the current system. It is NOT intended to be modified by the project, use CMAKE_SYSTEM_LIBRARY_PATH for this. See also CMAKE_SYSTEM_PREFIX_PATH.

CMAKE_SYSTEM_PREFIX_PATH: Path used for searching by FIND_XXX(), with appropriate suffixes added.

Specifies a path which will be used by the FIND_XXX() commands. It contains the "base" directories, the FIND_XXX() commands append appropriate subdirectories to the base directories. So FIND_PROGRAM() adds /bin to each of the directories in the path, FIND_LIBRARY() appends /lib to each of the directories, and FIND_PATH() and FIND_FILE() append /include . By default this contains the standard directories for the current system. It is NOT intended to be modified by the project, use CMAKE_PREFIX_PATH for this. See also CMAKE_SYSTEM_INCLUDE_PATH, CMAKE_SYSTEM_LIBRARY_PATH, CMAKE_SYSTEM_PROGRAM_PATH.

CMAKE_SYSTEM_PROGRAM_PATH: Path used for searching by FIND_PROGRAM().

Specifies a path which will be used by FIND_PROGRAM(). FIND_PROGRAM() will check each of the contained directories for the existence of the program which is currently searched. By default it contains the standard directories for the current system. It is NOT intended to be modified by the project, use CMAKE_PROGRAM_PATH for this. See also CMAKE_SYSTEM_PREFIX_PATH.

CMAKE_USER_MAKE_RULES_OVERRIDE: Specify a file that can change the build rule variables.

If this variable is set, it should to point to a CMakeLists.txt file that will be read in by CMake after all the system settings have been set, but before they have been used. This would allow you to override any variables that need to be changed for some special project.

Variables That Describe the System

APPLE: True if running on Mac OS X.

BORLAND: True if the Borland compiler is being used.

CMAKE_CL_64: Using the 64 bit compiler from Microsoft

Set to true when using the 64 bit cl compiler from Microsoft.

CMAKE_COMPILER_2005: Using the Visual Studio 2005 compiler from Microsoft

Set to true when using the Visual Studio 2005 compiler from Microsoft.

CMAKE_HOST_APPLE: True for Apple OS Xoperating systems.

CMAKE_HOST_SYSTEM: Name of system CMake is being run on.

The same as CMAKE_SYSTEM but for the host system instead of the target system when cross compiling.

CMAKE_HOST_SYSTEM_NAME: Name of the OS CMake is running on.

The same as CMAKE_SYSTEM_NAME but for the host system instead of the target system when cross compiling.

CMAKE_HOST_SYSTEM_PROCESSOR: The name of the CPU CMake is running on.

The same as CMAKE_SYSTEM_PROCESSOR but for the host system instead of the target system when cross compiling.

CMAKE_HOST_SYSTEM_VERSION: OS version CMake is running on.

The same as CMAKE_SYSTEM_VERSION but for the host system instead of the target system when cross compiling.

CMAKE_HOST_UNIX: True for UNIX and UNIX like operating systems.

Set to true when the host system is UNIX or UNIX like (i.e. APPLE and CYGWIN).

CMAKE_HOST_WIN32: True on windows systems, including win64.

Set to true when the host system is Windows and on cygwin.

CMAKE_OBJECT_PATH_MAX: Maximum object file full-path length allowed by native build tools.

CMake computes for every source file an object file name that is unique to the source file and deterministic with respect to the full path to the source file. This allows multiple source files in a target to share the same name if they lie in different directories without rebuilding when one is added or removed. However, it can produce long full paths in a few cases, so CMake shortens the path using a hashing scheme when the full path to an object file exceeds a limit. CMake has a built-in limit for each platform that is sufficient for common tools, but some native tools may have a lower limit. This variable may be set to specify the limit explicitly. The value must be an integer no less than 128.

CMAKE_SYSTEM: Name of system CMake is compiling for.

This variable is the composite of CMAKE_SYSTEM_NAME and CMAKE_SYSTEM_VERSION, like this ${CMAKE_SYSTEM_NAME}-

${CMAKE_SYSTEM_VERSION}. If CMAKE_SYSTEM_VERSION is not set, then CMAKE_SYSTEM is the same as CMAKE_SYSTEM_NAME.

CMAKE_SYSTEM_NAME: Name of the OS CMake is building for.

This is the name of the operating system on which CMake is targeting. On systems that have the uname command, this variable is set to the output of uname -s. Linux, Windows, and Darwin for Mac OS X are the values found on the big three operating systems.

CMAKE_SYSTEM_PROCESSOR: The name of the CPU CMake is building for.

On systems that support uname, this variable is set to the output of uname -p, on Windows it is set to the value of the environment variable PROCESSOR_ARCHITECTURE

CMAKE_SYSTEM_VERSION: OS version CMake is building for.

A numeric version string for the system, on systems that support uname, this variable is set to the output of uname -r. On other systems this is set to major-minor version numbers.

CYGWIN: True for cygwin.

MSVC: True when using Microsoft Visual C

MSVC80: True when using Microsoft Visual C 8.0

MSVC_IDE: True when using the Microsoft Visual C IDE

Set to true when the target platform is the Microsoft Visual C IDE, as opposed to the command line compiler.

MSVC_VERSION: The version of Microsoft Visual C/C++ being used if any.

The version of Microsoft Visual C/C++ being used if any. For example 1300 is MSVC 6.0.

UNIX: True for UNIX and UNIX like operating systems.

Set to true when the target system is UNIX or UNIX like (i.e. APPLE and CYGWIN).

WIN32: True on windows systems, including win64.

Set to true when the target system is Windows and on cygwin.

XCODE_VERSION: Version of Xcode (Xcode generator only).

Under the Xcode generator, this is the version of Xcode as specified in "Xcode.app/Contents/version.plist" (such as "3.1.2").

Variables for Languages

CMAKE_<LANG>_ARCHIVE_APPEND: Rule variable to append to a static archive.

This is a rule variable that tells CMake how to append to a static archive. It is used in place of CMAKE_<LANG>_CREATE_STATIC_LIBRARY on some platforms in order to support large object counts. See also CMAKE_<LANG>_ARCHIVE_CREATE and CMAKE_<LANG>_ARCHIVE_FINISH.

CMAKE_<LANG>_ARCHIVE_CREATE: Rule variable to create a new static archive.

This is a rule variable that tells CMake how to create a static archive. It is used in place of CMAKE_<LANG>_CREATE_STATIC_LIBRARY on some platforms in order to support large object counts. See also CMAKE_<LANG>_ARCHIVE_APPEND and CMAKE_<LANG>_ARCHIVE_FINISH.

CMAKE_<LANG>_ARCHIVE_FINISH: Rule variable to finish an existing static archive.

This is a rule variable that tells CMake how to finish a static archive. It is used in place of CMAKE_<LANG>_CREATE_STATIC_LIBRARY on some platforms in order to support large object counts. See also CMAKE_<LANG>_ARCHIVE_CREATE and CMAKE_<LANG>_ARCHIVE_APPEND.

CMAKE_<LANG>_COMPILER: The full path to the compiler for LANG.

This is the command that will be used as the <LANG> compiler. Once set, you cannot change this variable.

CMAKE_<LANG>_COMPILER_ABI: An internal variable subject to change.

This is used in determining the compiler ABI and is subject to change.

CMAKE_<LANG>_COMPILER_ID: An internal variable subject to change.

This is used in determining the compiler and is subject to change.

CMAKE_<LANG>_COMPILER_LOADED: Defined to true if the language is enabled.

When language <LANG> is enabled by project() or enable_language() this variable is defined to 1.

CMAKE_<LANG>_COMPILE_OBJECT: Rule variable to compile a single object file.

This is a rule variable that tells CMake how to compile a single object file for for the language <LANG>.

CMAKE_<LANG>_CREATE_SHARED_LIBRARY: Rule variable to create a shared library.

This is a rule variable that tells CMake how to create a shared library for the language <LANG>.

CMAKE_<LANG>_CREATE_SHARED_MODULE: Rule variable to create a shared module.

This is a rule variable that tells CMake how to create a shared library for the language <LANG>.

CMAKE_<LANG>_CREATE_STATIC_LIBRARY: Rule variable to create a static library.

This is a rule variable that tells CMake how to create a static library for the language <LANG>.

CMAKE_<LANG>_FLAGS_DEBUG: Flags for Debug build type or configuration.

<LANG> flags used when CMAKE_BUILD_TYPE is Debug.

CMAKE_<LANG>_FLAGS_MINSIZEREL: Flags for MinSizeRel build type or configuration.

<LANG> flags used when CMAKE_BUILD_TYPE is MinSizeRel.Short for minimum size release.

CMAKE_<LANG>_FLAGS_RELEASE: Flags for Release build type or configuration.

<LANG> flags used when CMAKE_BUILD_TYPE is Release

CMAKE_<LANG>_FLAGS_RELWITHDEBINFO: Flags for RelWithDebInfo type or configuration.

<LANG> flags used when CMAKE_BUILD_TYPE is RelWithDebInfo. Short for Release With Debug Information.

CMAKE_<LANG>_IGNORE_EXTENSIONS: File extensions that should be ignored by the build.

This is a list of file extensions that may be part of a project for a given language but are not compiled.

CMAKE_<LANG>_IMPLICIT_INCLUDE_DIRECTORIES: Directories implicitly searched by the compiler for header files.

CMake does not explicitly specify these directories on compiler command lines for language <LANG>. This prevents system include directories from being treated as user include directories on some compilers.

CMAKE_<LANG>_IMPLICIT_LINK_DIRECTORIES: Implicit linker search path detected for language <LANG>.

Compilers typically pass directories containing language runtime libraries and default library search paths when they invoke a linker. These paths are implicit linker search directories for the compiler's language. CMake automatically detects these directories for each language and reports the results in this variable.

CMAKE_<LANG>_IMPLICIT_LINK_LIBRARIES: Implicit link libraries and flags detected for language <LANG>.

Compilers typically pass language runtime library names and other flags when they invoke a linker. These flags are implicit link options for the compiler's language. CMake automatically detects these libraries and flags for each language and reports the results in this variable.

CMAKE_<LANG>_LINKER_PREFERENCE: Preference value for linker language selection.

The "linker language" for executable, shared library, and module targets is the language whose compiler will invoke the linker. The LINKER_LANGUAGE target property sets the language explicitly. Otherwise, the linker language is that whose linker preference value is highest among languages compiled and linked into the target. See also the CMAKE_<LANG>_LINKER_PREFERENCE_PROPAGATES variable.

CMAKE_<LANG>_LINKER_PREFERENCE_PROPAGATES: True if CMAKE_<LANG>_LINKER_PREFERENCE propagates across targets.

This is used when CMake selects a linker language for a target. Languages compiled directly into the target are always considered. A language compiled into static libraries linked by the target is considered if this variable is true.

CMAKE_<LANG>_LINK_EXECUTABLE : Rule variable to link and executable.

Rule variable to link and executable for the given language.

CMAKE_<LANG>_OUTPUT_EXTENSION: Extension for the output of a compile for a single file.

This is the extension for an object file for the given <LANG>. For example .obj for C on Windows.

CMAKE_<LANG>_PLATFORM_ID: An internal variable subject to change.

This is used in determining the platform and is subject to change.

CMAKE_<LANG>_SIZEOF_DATA_PTR: Size of pointer-to-data types for language <LANG>.

This holds the size (in bytes) of pointer-to-data types in the target platform ABI. It is defined for languages C and CXX (C++).

CMAKE_<LANG>_SOURCE_FILE_EXTENSIONS: Extensions of source files for the given language.

This is the list of extensions for a given languages source files.

CMAKE_COMPILER_IS_GNU<LANG>: True if the compiler is GNU.

If the selected <LANG> compiler is the GNU compiler then this is TRUE, if not it is FALSE.

CMAKE_INTERNAL_PLATFORM_ABI: An internal variable subject to change.

This is used in determining the compiler ABI and is subject to change.

CMAKE_USER_MAKE_RULES_OVERRIDE_<LANG>: Specify a file that can change the build rule variables.

If this variable is set, it should to point to a CMakeLists.txt file that will be read in by CMake after all the system settings have been set, but before they have been used. This would allow you to override any variables that need to be changed for some language.

Variables That Control the Build

CMAKE_<CONFIG>_POSTFIX: Default filename postfix for libraries under configuration <CONFIG>.

When a non-executable target is created its <CONFIG>_POSTFIX target property is initialized with the value of this variable if it is set.

CMAKE_ARCHIVE_OUTPUT_DIRECTORY: Where to put all the ARCHIVE targets when built.

This variable is used to initialize the ARCHIVE_OUTPUT_DIRECTORY property on all the targets. See that target property for additional information.

CMAKE_BUILD_WITH_INSTALL_RPATH: Use the install path for the RPATH

Normally CMake uses the build tree for the RPATH when building executables etc on systems that use RPATH. When the software is installed the executables etc are relinked by CMake to have the install RPATH. If this variable is set to true then the software is always built with the install path for the RPATH and does not need to be relinked when installed.

CMAKE_DEBUG_POSTFIX: See variable CMAKE_<CONFIG>_POSTFIX.

This variable is a special case of the more-general CMAKE_<CONFIG>_POSTFIX variable for the DEBUG configuration.

CMAKE_EXE_LINKER_FLAGS: Linker flags used to create executables.

Flags used by the linker when creating an executable.

CMAKE_EXE_LINKER_FLAGS_[CMAKE_BUILD_TYPE]: Flag used when linking an executable.

Same as CMAKE_C_FLAGS_* but used by the linker when creating executables.

CMAKE_Fortran_MODULE_DIRECTORY: Fortran module output directory.

This variable is used to initialize the Fortran_MODULE_DIRECTORY property on all the targets. See that target property for additional information.

CMAKE_INCLUDE_CURRENT_DIR: Automatically add the current source- and build directories to the include path.

If this variable is enabled, CMake automatically adds in each directory ${CMAKE_CURRENT_SOURCE_DIR} and ${CMAKE_CURRENT_BINARY_DIR} to the include path for this directory. These additional include directories do not propagate down to subdirectories. This is useful mainly for out-of-source builds, where files generated into the build tree are included by files located in the source tree.

By default CMAKE_INCLUDE_CURRENT_DIR is OFF.

CMAKE_INSTALL_NAME_DIR: Mac OS X directory name for installed targets.

CMAKE_INSTALL_NAME_DIR is used to initialize the INSTALL_NAME_DIR property on all targets. See that target property for more information.

CMAKE_INSTALL_RPATH: The rpath to use for installed targets.

A semicolon-separated list specifying the rpath to use in installed targets (for platforms that support it). This is used to initialize the target property INSTALL_RPATH for all targets.

CMAKE_INSTALL_RPATH_USE_LINK_PATH: Add paths to linker search and installed rpath.

CMAKE_INSTALL_RPATH_USE_LINK_PATH is a boolean that if set to true will append directories in the linker search path and outside the project to the INSTALL_RPATH. This is used to initialize the target property INSTALL_RPATH_USE_LINK_PATH for all targets.

CMAKE_LIBRARY_OUTPUT_DIRECTORY: Where to put all the LIBRARY targets when built.

This variable is used to initialize the LIBRARY_OUTPUT_DIRECTORY property on all the targets. See that target property for additional information.

CMAKE_LIBRARY_PATH_FLAG: The flag used to add a library search path to a compiler.

The flag used to specify a library directory to the compiler. On most compilers this is "-L".

CMAKE_LINK_DEF_FILE_FLAG : Linker flag used to specify a .def file for dll creation.

The flag used to add a .def file when creating a dll on Windows, this is only defined on Windows.

CMAKE_LINK_LIBRARY_FILE_FLAG: Flag used to link a library specified by a path to its file.

The flag used before a library file path is given to the linker. This is needed only on very few platforms.

CMAKE_LINK_LIBRARY_FLAG: Flag used to link a library into an executable.

The flag used to specify a library to link to an executable. On most compilers this is "-l".

CMAKE_NO_BUILTIN_CHRPATH: Do not use the builtin ELF editor to fix RPATHs on installation.

When an ELF binary needs to have a different RPATH after installation than it does in the build tree, CMake uses a builtin editor to change the RPATH in the installed copy. If this variable is set to true then CMake will relink the binary before installation instead of using its builtin editor.

CMAKE_RUNTIME_OUTPUT_DIRECTORY: Where to put all the RUNTIME targets when built.

This variable is used to initialize the RUNTIME_OUTPUT_DIRECTORY property on all the targets. See that target property for additional information.

CMAKE_SKIP_BUILD_RPATH: Do not include RPATHs in the build tree.

Normally CMake uses the build tree for the RPATH when building executables etc on systems that use RPATH. When the software is installed the executables etc are relinked by CMake to have the install RPATH. If this variable is set to true then the software is always built with no RPATH.

CMAKE_USE_RELATIVE_PATHS: Use relative paths (May not work!).

If this is set to TRUE, then the CMake will use relative paths between the source and binary tree. This option does not work for more complicated projects, and relative paths are used when possible. In general, it is not possible to move CMake generated Makefiles to a different location regardless of the value of this variable.

EXECUTABLE_OUTPUT_PATH: Old executable location variable.

The target property RUNTIME_OUTPUT_DIRECTORY supercedes this variable for a target if it is set. Executable targets are otherwise placed in this directory.

LIBRARY_OUTPUT_PATH: Old library location variable.

The target properties ARCHIVE_OUTPUT_DIRECTORY, LIBRARY_OUTPUT_DIRECTORY, and RUNTIME_OUTPUT_DIRECTORY supercede this variable for a target if they are set. Library targets are otherwise placed in this directory.

Variables That Provide Information

variables defined by CMake, that give information about the project, and CMake

CMAKE_AR: Name of archiving tool for static libraries.

This specifies name of the program that creates archive or static libraries.

CMAKE_BINARY_DIR: The path to the top level of the build tree.

This is the full path to the top level of the current CMake build tree. For an in-source build, this would be the same as CMAKE_SOURCE_DIR.

CMAKE_BUILD_TOOL: Tool used for the actual build process.

This variable is set to the program that will be needed to build the output of CMake. If the generator selected was Visual Studio 6, the CMAKE_MAKE_PROGRAM will be set to msdev, for Unix Makefiles it will be set to make or gmake, and for Visual Studio 7 it set to devenv. For NMake Makefiles the value is nmake. This can be useful for adding special flags and commands based on the final build environment.

CMAKE_CACHEFILE_DIR: The directory with the CMakeCache.txt file.

This is the full path to the directory that has the CMakeCache.txt file in it. This is the same as CMAKE_BINARY_DIR.

CMAKE_CACHE_MAJOR_VERSION: Major version of CMake used to create the CMakeCache.txt file

This is stores the major version of CMake used to write a CMake cache file. It is only different when a different version of CMake is run on a previously created cache file.

CMAKE_CACHE_MINOR_VERSION: Minor version of CMake used to create the CMakeCache.txt file

This is stores the minor version of CMake used to write a CMake cache file. It is only different when a different version of CMake is run on a previously created cache file.

CMAKE_CACHE_PATCH_VERSION: Patch version of CMake used to create the CMakeCache.txt file

This is stores the patch version of CMake used to write a CMake cache file. It is only different when a different version of CMake is run on a previously created cache file.

CMAKE_CFG_INTDIR: Build-time reference to per-configuration output subdirectory.

For native build systems supporting multiple configurations in the build tree (such as Visual Studio and Xcode), the value is a reference to a build-time variable specifying the name of the per-configuration output subdirectory. On Makefile generators this evaluates to "." because there is only one configuration in a build tree. Example values:

```
$(IntDir)        = Visual Studio 6
$(OutDir)        = Visual Studio 7, 8, 9
$(Configuration) = Visual Studio 10
$(CONFIGURATION) = Xcode
.                = Make-based tools
```

Since these values are evaluated by the native build system, this variable is suitable only for use in command lines that will be evaluated at build time. Example of intended usage:

```
add_executable(mytool mytool.c)
add_custom_command(
    OUTPUT out.txt
    COMMAND
      ${CMAKE_CURRENT_BINARY_DIR}/${CMAKE_CFG_INTDIR}/mytool
      ${CMAKE_CURRENT_SOURCE_DIR}/in.txt out.txt
    DEPENDS mytool in.txt
    )
add_custom_target(drive ALL DEPENDS out.txt)
```

Note that CMAKE_CFG_INTDIR is no longer necessary for this purpose but has been left for compatibility with existing projects. Instead add_custom_command() recognizes executable target names in its COMMAND option, so "${CMAKE_CURRENT_BINARY_DIR}/${CMAKE_CFG_INTDIR}/mytool" can be replaced by just "mytool".

This variable is read-only. Setting it is undefined behavior. In multi-configuration build systems the value of this variable is passed as the value of preprocessor symbol "CMAKE_INTDIR" to the compilation of all source files.

CMAKE_COMMAND: The full path to the cmake executable.

This is the full path to the CMake executable cmake which is useful from custom commands that want to use the cmake -E option for portable system commands, e.g. /usr/local/bin/cmake

CMAKE_CROSSCOMPILING: Is CMake currently cross compiling.

This variable will be set to true by CMake if CMake is cross compiling. Specifically if the build platform is different from the target platform.

CMAKE_CTEST_COMMAND: Full path to ctest command installed with cmake.

This is the full path to the CTest executable ctest which is useful from custom commands that want to use the cmake -E option for portable system commands.

CMAKE_CURRENT_BINARY_DIR: The path to the binary directory currently being processed.

This the full path to the build directory that is currently being processed by cmake. Each directory added by add_subdirectory will create a binary directory in the build tree, and as it is being processed this variable will be set. For in-source builds this is the current source directory being processed.

CMAKE_CURRENT_LIST_FILE: Full path to the listfile currently being processed.

As CMake processes the listfiles in your project this variable will always be set to the one currently being processed. See also CMAKE_PARENT_LIST_FILE.

CMAKE_CURRENT_LIST_LINE: The line number of the current file being processed.

This is the line number of the file currently being processed by cmake.

CMAKE_CURRENT_SOURCE_DIR: The path to the source directory currently being processed.

This the full path to the source directory that is currently being processed by cmake.

CMAKE_DL_LIBS: Name of library containing dlopen and dlcose.

The name of the library that has dlopen and dlclose in it, usually -ldl on most UNIX machines.

CMAKE_EDIT_COMMAND: Full path to cmake-gui or ccmake.

This is the full path to the CMake executable that can graphically edit the cache. For example, cmake-gui, ccmake, or cmake -i.

CMAKE_EXECUTABLE_SUFFIX: The suffix for executables on this platform.

The suffix to use for the end of an executable if any, .exe on Windows.

CMAKE_EXECUTABLE_SUFFIX_<LANG> overrides this for language <LANG>.

CMAKE_EXTRA_SHARED_LIBRARY_SUFFIXES: Additional suffixes for shared libraries.

Extensions for shared libraries other than that specified by CMAKE_SHARED_LIBRARY_SUFFIX, if any. CMake uses this to recognize external shared library files during analysis of libraries linked by a target.

CMAKE_GENERATOR: The generator used to build the project.

The name of the generator that is being used to generate the build files. (e.g. "Unix Makefiles", "Visual Studio 6", etc.)

CMAKE_HOME_DIRECTORY: Path to top of source tree.

This is the path to the top level of the source tree.

CMAKE_IMPORT_LIBRARY_PREFIX: The prefix for import libraries that you link to.

The prefix to use for the name of an import library if used on this platform.

CMAKE_IMPORT_LIBRARY_PREFIX_<LANG> overrides this for language <LANG>.

CMAKE_IMPORT_LIBRARY_SUFFIX: The suffix for import libraries that you link to.

The suffix to use for the end of an import library if used on this platform.

CMAKE_IMPORT_LIBRARY_SUFFIX_<LANG> overrides this for language <LANG>.

CMAKE_LINK_LIBRARY_SUFFIX: The suffix for libraries that you link to.

The suffix to use for the end of a library, .lib on Windows.

CMAKE_MAJOR_VERSION: The Major version of cmake (i.e. the 2 in 2.X.X)

This specifies the major version of the CMake executable being run.

CMAKE_MAKE_PROGRAM: See CMAKE_BUILD_TOOL.

This variable is around for backwards compatibility, see CMAKE_BUILD_TOOL.

CMAKE_MINOR_VERSION: The Minor version of cmake (i.e. the 4 in X.4.X).

This specifies the minor version of the CMake executable being run.

CMAKE_PARENT_LIST_FILE: Full path to the parent listfile of the one currently being processed.

As CMake processes the listfiles in your project this variable will always be set to the listfile that included or somehow invoked the one currently being processed. See also CMAKE_CURRENT_LIST_FILE.

CMAKE_PATCH_VERSION: The patch version of cmake (i.e. the 3 in X.X.3).

This specifies the patch version of the CMake executable being run.

CMAKE_PROJECT_NAME: The name of the current project.

This specifies name of the current project from the closest inherited PROJECT command.

CMAKE_RANLIB: Name of randomizing tool for static libraries.

This specifies name of the program that randomizes libraries on UNIX, not used on Windows, but may be present.

CMAKE_ROOT: Install directory for running cmake.

This is the install root for the running CMake and the Modules directory can be found here. This is commonly used in this format: ${CMAKE_ROOT}/Modules

CMAKE_SHARED_LIBRARY_PREFIX: The prefix for shared libraries that you link to.

The prefix to use for the name of a shared library, lib on UNIX.

CMAKE_SHARED_LIBRARY_PREFIX_<LANG> overrides this for language <LANG>.

CMAKE_SHARED_LIBRARY_SUFFIX: The suffix for shared libraries that you link to.

The suffix to use for the end of a shared library, .dll on Windows.

CMAKE_SHARED_LIBRARY_SUFFIX_<LANG> overrides this for language <LANG>.

CMAKE_SHARED_MODULE_PREFIX: The prefix for loadable modules that you link to.

The prefix to use for the name of a loadable module on this platform.

CMAKE_SHARED_MODULE_PREFIX_<LANG> overrides this for language <LANG>.

CMAKE_SHARED_MODULE_SUFFIX: The suffix for shared libraries that you link to.

The suffix to use for the end of a loadable module on this platform

CMAKE_SHARED_MODULE_SUFFIX_<LANG> overrides this for language <LANG>.

CMAKE_SIZEOF_VOID_P: Size of a void pointer.

This is set to the size of a pointer on the machine, and is determined by a try compile. If a 64 bit size is found, then the library search path is modified to look for 64 bit libraries first.

CMAKE_SKIP_RPATH: If true, do not add run time path information.

If this is set to TRUE, then the rpath information is not added to compiled executables. The default is to add rpath information if the platform supports it.This allows for easy running from the build tree.

CMAKE_SOURCE_DIR: The path to the top level of the source tree.

This is the full path to the top level of the current CMake source tree. For an in-source build, this would be the same as CMAKE_BINARY_DIR.

CMAKE_STANDARD_LIBRARIES: Libraries linked into every executable and shared library.

This is the list of libraries that are linked into all executables and libraries.

CMAKE_STATIC_LIBRARY_PREFIX: The prefix for static libraries that you link to.

The prefix to use for the name of a static library, lib on UNIX.

CMAKE_STATIC_LIBRARY_PREFIX_<LANG> overrides this for language <LANG>.

CMAKE_STATIC_LIBRARY_SUFFIX: The suffix for static libraries that you link to.

The suffix to use for the end of a static library, .lib on Windows.

CMAKE_STATIC_LIBRARY_SUFFIX_<LANG> overrides this for language <LANG>.

CMAKE_USING_VC_FREE_TOOLS: True if free visual studio tools being used.

This is set to true if the compiler is Visual Studio free tools.

CMAKE_VERBOSE_MAKEFILE: Create verbose Makefiles if on.

This variable defaults to false. You can set this variable to true to make CMake produce verbose Makefiles that show each command line as it is used.

CMAKE_VERSION: The full version of cmake in major.minor.patch format.

This specifies the full version of the CMake executable being run. This variable is defined by versions 2.6.3 and higher. See variables CMAKE_MAJOR_VERSION, CMAKE_MINOR_VERSION, and CMAKE_PATCH_VERSION for individual version components.

PROJECT_BINARY_DIR: Full path to build directory for project.

This is the binary directory of the most recent PROJECT command.

PROJECT_NAME: Name of the project given to the project command.

This is the name given to the most recent PROJECT command.

PROJECT_SOURCE_DIR: Top level source directory for the current project.

This is the source directory of the most recent PROJECT command.

[Project name]_BINARY_DIR: Top level binary directory for the named project.

A variable is created with the name used in the PROJECT command, and is the binary directory for the project. This can be useful when SUBDIR is used to connect several projects.

[Project name]_SOURCE_DIR: Top level source directory for the named project.

A variable is created with the name used in the PROJECT command, and is the source directory for the project. This can be useful when add_subdirectory is used to connect several projects.