NAME

cmake-variables - CMake Variables Reference

VARIABLES THAT PROVIDE INFORMATION

CMAKE_ARGC

Number of command line arguments passed to CMake in script mode.

When run in -P script mode, CMake sets this variable to the number of command line arguments. See also CMAKE_ARGV0, 1, 2 ...

CMAKE_ARGV0

Command line argument passed to CMake in script mode.

When run in -P script mode, CMake sets this variable to the first command line argument. It then also sets CMAKE_ARGV1, CMAKE_ARGV2, also CMAKE_ARGC.

CMAKE_AR

Name of archiving tool for static libraries.

This specifies the 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

This variable exists only for backwards compatibility. It contains the same value as **CMAKE_MAKE_PROGRAM**. Use that variable instead.

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 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 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 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_DIR

Full directory of the listfile currently being processed.

As CMake processes the listfiles in your project this variable will always be set to the directory where the listfile which is currently being processed (CMAKE_CURRENT_LIST_FILE) is located. The value has dynamic scope. When CMake starts processing commands in a source file it sets this variable to the directory where this file is located. When CMake finishes processing commands from the file it restores the previous value. Therefore the value of the variable inside a macro or function is the directory of the file invoking the bottom-most entry on the call stack, not the directory of the file containing the macro or function definition.

See also CMAKE_CURRENT_LIST_FILE.

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. The value has dynamic scope. When CMake starts processing commands in a source file it sets this variable to the location of the file. When CMake finishes processing commands from the file it restores the previous value. Therefore the value of the variable inside a macro or function is the file invoking the bottom-most entry on the call stack, not the file containing the macro or function definition.

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. Defined only for Makefile generators when not using an extra generator for an IDE.

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

CMAKE_EXECUTABLE_SUFFIX
The suffix for executables on this platform.

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

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

CMAKE_EXTRA_GENERATOR
The extra generator used to build the project.

When using the Eclipse, CodeBlocks or KDevelop generators, CMake generates Makefiles (CMAKE_GENERATOR) and additionally project files for the respective IDE. This IDE project file generator is stored in CMAKE_EXTRA_GENERATOR (e.g. Eclipse CDT4).

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_GENERATOR_TOOLSET
Native build system toolset name specified by user.

Some CMake generators support a toolset name to be given to the native build system to choose a compiler. If the user specifies a toolset name (e.g. via the cmake -T option) the value will be available in this variable.

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 filename if used on this platform.

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

CMAKE_JOB_POOL_COMPILE
This variable is used to initialize the **JOB_POOL_COMPILE** property on all the targets. See **JOB_POOL_COMPILE** for additional information.

CMAKE_JOB_POOL_LINK
This variable is used to initialize the **JOB_POOL_LINK** property on all the targets. See **JOB_POOL_LINK** for additional information.

CMAKE_LINK_LIBRARY_SUFFIX
The suffix for libraries that you link to.

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

CMAKE_MAJOR_VERSION
First version number component of the **CMAKE_VERSION** variable.

CMAKE_MAKE_PROGRAM
Tool that can launch the native build system. The value may be the full path to an executable or just the tool name if it is expected to be in the **PATH**.

The tool selected depends on the **CMAKE_GENERATOR** used to configure the project:

- The Makefile generators set this to **make**, **gmake**, or a generator-specific tool (e.g. **nmake** for NMake Makefiles).

  These generators store **CMAKE_MAKE_PROGRAM** in the CMake cache so that it may be edited by the user.

- The Ninja generator sets this to **ninja**.

  This generator stores **CMAKE_MAKE_PROGRAM** in the CMake cache so that it may be edited by the user.

- The Xcode generator sets this to **xcodebuild** (or possibly an otherwise undocumented **cmakexbuild** wrapper implementing some workarounds).

  This generator stores **CMAKE_MAKE_PROGRAM** in the CMake cache so that it may be edited by the user.

- The Visual Studio generators set this to the full path to **MSBuild.exe** (VS >= 10), **devenv.com** (VS 7,8,9), **VCExpress.exe** (VS Express 8,9), or **msdev.exe** (VS 6). (See also variables **CMAKE_VS_MSBUILD_COMMAND**, **CMAKE_VS_DEVENV_COMMAND**, and **CMAKE_VS_MSDEV_COMMAND**.)

These generators prefer to lookup the build tool at build time rather than to store **CMAKE_MAKE_PROGRAM** in the CMake cache ahead of time. This is because the tools are version-specific and can be located using the Windows Registry. It is also necessary because the proper build tool may depend on the project content (e.g. the Intel Fortran plugin to VS 10 and 11 requires **devenv.com** to build its **.vfproj** project files even though **MSBuild.exe** is normally preferred to support the **CMAKE_GENERATOR_TOOLSET**).

For compatibility with versions of CMake prior to 3.0, if a user or project explicitly adds **CMAKE_MAKE_PROGRAM** to the CMake cache then CMake will use the specified value if possible.

The **CMAKE_MAKE_PROGRAM** variable is set for use by project code. The value is also used by the cmake(1) **--build** and ctest(1) **--build-and-test** tools to launch the native build process.

CMAKE_MINIMUM_REQUIRED_VERSION
Version specified to cmake_minimum_required command

Variable containing the VERSION component specified in the cmake_minimum_required command.

CMAKE_MINOR_VERSION
Second version number component of the **CMAKE_VERSION** variable.

CMAKE_PARENT_LIST_FILE
Full path to the CMake file that included the current one.

While processing a CMake file loaded by include() or find_package() this variable contains the full path to the file including it. The top of the include stack is always the CMakeLists.txt for the current directory. See also CMAKE_CURRENT_LIST_FILE.

CMAKE_PATCH_VERSION
Third version number component of the **CMAKE_VERSION** variable.

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_SCRIPT_MODE_FILE
Full path to the -P script file currently being processed.

When run in -P script mode, CMake sets this variable to the full path of the script file. When run to configure a CMakeLists.txt file, this variable is not set.

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 filename, .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 filename 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_INSTALL_RULES
Whether to disable generation of installation rules.

If TRUE, cmake will neither generate installaton rules nor will it generate cmake_install.cmake files. This variable is FALSE by default.

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. To omit RPATH in the install step, but not the build step, use CMAKE_SKIP_INSTALL_RPATH instead.

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 filename, .lib on Windows.

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

CMAKE_TOOLCHAIN_FILE

Path to toolchain file supplied to **cmake(1)**.

This variable is specified on the command line when cross-compiling with CMake. It is the path to a file which is read early in the CMake run and which specifies locations for compilers and toolchain utilities, and other target platform and compiler related information.

CMAKE_TWEAK_VERSION

Defined to **0** for compatibility with code written for older CMake versions that may have defined higher values.

NOTE:

In CMake versions 2.8.2 through 2.8.12, this variable holds the fourth version number component of the **CMAKE_VERSION** variable.

CMAKE_VERBOSE_MAKEFILE

Enable verbose output from Makefile builds.

This variable is a cache entry initialized (to FALSE) by the **project()** command. Users may enable the option in their local build tree to get more verbose output from Makefile builds and show each command line as it is launched.

CMAKE_VERSION

The CMake version string as three non-negative integer components separated by **.** and possibly followed by **-** and other information. The first two components represent the feature level and the third component represents either a bug-fix level or development date.

Release versions and release candidate versions of CMake use the format:

```
<major>.<minor>.<patch>[-rc<n>]
```

where the **<patch>** component is less than **20000000**. Development versions of CMake use the format:

```
<major>.<minor>.<date>[-<id>]
```

where the **<date>** component is of format **CCYYMMDD** and **<id>** may contain arbitrary text. This represents development as of a particular date following the **<major>.<minor>** feature release.

Individual component values are also available in variables:

- **CMAKE_MAJOR_VERSION**
- **CMAKE_MINOR_VERSION**
- **CMAKE_PATCH_VERSION**
- **CMAKE_TWEAK_VERSION**

Use the **if()** command **VERSION_LESS**, **VERSION_EQUAL**, or **VERSION_GREATER** operators to compare version string values against **CMAKE_VERSION** using a component-wise test. Version component values may be 10 or larger so do not attempt to compare version strings as floating-point numbers.

NOTE:

CMake versions 2.8.2 through 2.8.12 used three components for the feature level. Release versions represented the bug-fix level in a fourth component, i.e. **<major>.<minor>.<patch>**[**.<tweak>**][**-rc<n>**]. Development versions represented the development date in the fourth component, i.e. **<major>.<minor>.<patch>.<date>**[**-<id>**].

CMake versions prior to 2.8.2 used three components for the feature level and had no bug-fix component. Release versions used an even-valued second component, i.e. **<major>.<even-minor>.<patch>**[**-rc<n>**]. Development versions used an odd-valued second component with the development date as the third component, i.e. **<major>.<odd-minor>.<date>**.

The **CMAKE_VERSION** variable is defined by CMake 2.6.3 and higher. Earlier versions defined only the individual component variables.

CMAKE_VS_DEVENV_COMMAND

The generators for **Visual Studio 7** and above set this variable to the **devenv.com** command installed with the corresponding Visual Studio version. Note that this variable may be empty on Visual Studio Express editions because they do not provide this tool.

This variable is not defined by other generators even if **devenv.com** is installed on the computer.

The **CMAKE_VS_MSBUILD_COMMAND** is also provided for **Visual Studio 10 2010** and above. See also the**CMAKE_MAKE_PR OGRAM** variable.

CMAKE_VS_INTEL_Fortran_PROJECT_VERSION

When generating for Visual Studio 7 or greater with the Intel Fortran plugin installed, this specifies the .vfproj project file format version. This is intended for internal use by CMake and should not be used by project code.

CMAKE_VS_MSBUILD_COMMAND

The generators for **Visual Studio 10 2010** and above set this variable to the **MSBuild.exe** command installed with the corresponding Visual Studio version.

This variable is not defined by other generators even if **MSBuild.exe** is installed on the computer.

The **CMAKE_VS_DEVENV_COMMAND** is also provided for the non-Express editions of Visual Studio. See also the **CMAKE_MAKE_PROGRAM** variable.

CMAKE_VS_MSDEV_COMMAND

The **Visual Studio 6** generator sets this variable to the **msdev.exe** command installed with Visual Studio 6.

This variable is not defined by other generators even if **msdev.exe** is installed on the computer.

See also the **CMAKE_MAKE_PROGRAM** variable.

CMAKE_VS_PLATFORM_TOOLSET

Visual Studio Platform Toolset name.

VS 10 and above use MSBuild under the hood and support multiple compiler toolchains. CMake may specify a toolset explicitly, such as v110 for VS 11 or Windows7.1SDK for 64-bit support in VS 10 Express. CMake provides the name of the chosen toolset in this variable.

CMAKE_XCODE_PLATFORM_TOOLSET

Xcode compiler selection.

Xcode supports selection of a compiler from one of the installed toolsets. CMake provides the name of the chosen toolset in this variable, if any is explicitly selected (e.g. via the cmake -T option).

PROJECT_BINARY_DIR

Full path to build directory for project.

This is the binary 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 **add_subdirectory()** is used to connect several projects.

PROJECT_NAME

Name of the project given to the project command.

This is the name given to the most recent **project()** command.

<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.

<PROJECT–NAME>_VERSION

Value given to the **VERSION** option of the most recent call to the **project()** command with project name **<PROJECT-NAME>**, if any.

See also the component-wise version variables **<PROJECT-NAME>_VERSION_MAJOR**, **<PROJECT-NAME>_VERSION_MINOR**, **<PROJECT-NAME>_VERSION_PATCH**, and **<PROJECT-NAME>_VERSION_TWEAK**.

<PROJECT–NAME>_VERSION_MAJOR

First version number component of the **<PROJECT-NAME>_VERSION** variable as set by the **project()** command.

<PROJECT–NAME>_VERSION_MINOR

Second version number component of the **<PROJECT-NAME>_VERSION** variable as set by the **project()** command.

<PROJECT–NAME>_VERSION_PATCH

Third version number component of the **<PROJECT-NAME>_VERSION** variable as set by the **project()** command.

<PROJECT–NAME>_VERSION_TWEAK

Fourth version number component of the **<PROJECT-NAME>_VERSION** variable as set by the **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_VERSION

Value given to the **VERSION** option of the most recent call to the **project()** command, if any.

See also the component-wise version variables **PROJECT_VERSION_MAJOR**, **PROJECT_VERSION_MINOR**, **PROJECT_VERSION_PATCH**, and **PROJECT_VERSION_TWEAK**.

PROJECT_VERSION_MAJOR

First version number component of the **PROJECT_VERSION** variable as set by the **project()** command.

PROJECT_VERSION_MINOR

Second version number component of the **PROJECT_VERSION** variable as set by the **project()** command.

PROJECT_VERSION_PATCH

Third version number component of the **PROJECT_VERSION** variable as set by the **project()** command.

PROJECT_VERSION_TWEAK

Fourth version number component of the **PROJECT_VERSION** variable as set by the **project()** command.

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_ABSOLUTE_DESTINATION_FILES
List of files which have been installed using an ABSOLUTE DESTINATION path.

This variable is defined by CMake-generated cmake_install.cmake scripts. It can be used (read-only) by programs or scripts that source those install scripts. This is used by some CPack generators (e.g. RPM).

CMAKE_APPBUNDLE_PATH
Search path for OS X application bundles used by the **find_program()**, and **find_package()** commands.

CMAKE_AUTOMOC_RELAXED_MODE
Switch between strict and relaxed automoc mode.

By default, **AUTOMOC** behaves exactly as described in the documentation of the **AUTOMOC** target property. When set to **TRUE**, it accepts more input and tries to find the correct input file for **moc** even if it differs from the documented behaviour. In this mode it e.g. also checks whether a header file is intended to be processed by moc when a **foo.moc** file has been included.

Relaxed mode has to be enabled for KDE4 compatibility.

CMAKE_BACKWARDS_COMPATIBILITY
Deprecated. See CMake Policy **CMP0001** documentation.

CMAKE_BUILD_TYPE
Specifies the build type on single-configuration generators.

This statically specifies what build type (configuration) will be built in this build tree. Possible values are empty, Debug, Release, RelWithDebInfo and MinSizeRel. This variable is only meaningful to single-configuration generators (such as make and Ninja) i.e. those which choose a single configuration when CMake runs to generate a build tree as opposed to multi-configuration generators which offer selection of the build configuration within the generated build environment. There are many per-config properties and variables (usually following clean SOME_VAR_<CONFIG> order conventions), such as CMAKE_C_FLAGS_<CONFIG>, specified as uppercase: CMAKE_C_FLAGS_[DEBUG|RELEASE|RELWITHDEBINFO|MINSIZEREL]. For example, in a build tree configured to build type Debug, CMake will see to having CMAKE_C_FLAGS_DEBUG settings get added to the CMAKE_C_FLAGS settings. See also CMAKE_CONFIGURATION_TYPES.

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 on multi-config generators.

This specifies what build types (configurations) 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 for details of managing configuration data, and CMAKE_CFG_INTDIR.

CMAKE_DEBUG_TARGET_PROPERTIES
Enables tracing output for target properties.

This variable can be populated with a list of properties to generate debug output for when evaluating target properties. Currently it can only be used when evaluating the **INCLUDE_DIRECTORIES**, **COMPILE_DEFINITIONS**, **COMPILE_OPTIONS**, **AUTOUIC_OPTIONS**, **POSITION_INDEPENDENT_CODE** target properties and any other property listed in **COMPATIBLE_INTERFACE_STRING** and other **COMPATIBLE_INTERFACE_** properties. It outputs an origin for each entry in the target property. Default is unset.

CMAKE_DISABLE_FIND_PACKAGE_<PackageName>
Variable for disabling find_package() calls.

Every non-REQUIRED find_package() call in a project can be disabled by setting the variable CMAKE_DISABLE_FIND_PACKAGE_<PackageName> to TRUE. This can be used to build a project without an optional package, although that package is installed.

This switch should be used during the initial CMake run. Otherwise if the package has already been found in a previous CMake run, the variables which have been stored in the cache will still be there. In that case it is recommended to remove the cache variables for this package from the cache using the cache editor or cmake -U

CMAKE_ERROR_DEPRECATED
Whether to issue deprecation errors for macros and functions.

If TRUE, this can be used by macros and functions to issue fatal errors when deprecated macros or functions are used. This variable is FALSE by default.

CMAKE_ERROR_ON_ABSOLUTE_INSTALL_DESTINATION
Ask cmake_install.cmake script to error out as soon as a file with absolute INSTALL DESTINATION is encountered.

The fatal error is emitted before the installation of the offending file takes place. This variable is used by CMake-generated cmake_install.cmake scripts. If one sets this variable to ON while running the script, it may get fatal error messages from the script.

CMAKE_SYSROOT
Path to pass to the compiler in the **--sysroot** flag.

The **CMAKE_SYSROOT** content is passed to the compiler in the **--sysroot** flag, if supported. The path is also stripped from the RPATH/RUNPATH if necessary on installation. The **CMAKE_SYSROOT** is also used to prefix paths searched by the **find_*** commands.

This variable may only be set in a toolchain file specified by the **CMAKE_TOOLCHAIN_FILE** variable.

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_FIND_NO_INSTALL_PREFIX
Ignore the **CMAKE_INSTALL_PREFIX** when searching for assets.

CMake adds the **CMAKE_INSTALL_PREFIX** and the **CMAKE_STAGING_PREFIX** variable to the **CMAKE_SYSTEM_PREFIX_PATH** by default. This variable may be set on the command line to control that behavior.

Set **CMAKE_FIND_NO_INSTALL_PREFIX** to TRUE to tell find_package not to search in the **CMAKE_INSTALL_PREFIX** or **CMAKE_STAGING_PREFIX** by default. Note that the prefix may still be searched for other reasons, such as being the same prefix as the CMake installation, or for being a built-in system prefix.

CMAKE_FIND_PACKAGE_WARN_NO_MODULE

Tell find_package to warn if called without an explicit mode.

If find_package is called without an explicit mode option (MODULE, CONFIG or NO_MODULE) and no Find<pkg>.cmake module is in CMAKE_MODULE_PATH then CMake implicitly assumes that the caller intends to search for a package configuration file. If no package configuration file is found then the wording of the failure message must account for both the case that the package is really missing and the case that the project has a bug and failed to provide the intended Find module. If instead the caller specifies an explicit mode option then the failure message can be more specific.

Set CMAKE_FIND_PACKAGE_WARN_NO_MODULE to TRUE to tell find_package to warn when it implicitly assumes Config mode. This helps developers enforce use of an explicit mode in all calls to find_package within a project.

CMAKE_FIND_ROOT_PATH

List of root paths to search on the filesystem.

This variable is most useful when cross-compiling. CMake uses the paths in this list as alternative roots to find filesystem items with **find_package()**, **find_library()** etc.

CMAKE_FIND_ROOT_PATH_MODE_INCLUDE

This variable controls whether the **CMAKE_FIND_ROOT_PATH** and **CMAKE_SYSROOT** are used by **find_file()** and **find_path()**.

If set to **ONLY**, then only the roots in **CMAKE_FIND_ROOT_PATH** will be searched. If set to **NEVER**, then the roots in **CMAKE_FIND_ROOT_PATH** will be ignored and only the host system root will be used. If set to **BOTH**, then the host system paths and the paths in **CMAKE_FIND_ROOT_PATH** will be searched.

CMAKE_FIND_ROOT_PATH_MODE_LIBRARY

This variable controls whether the **CMAKE_FIND_ROOT_PATH** and **CMAKE_SYSROOT** are used by **find_library()**.

If set to **ONLY**, then only the roots in **CMAKE_FIND_ROOT_PATH** will be searched. If set to **NEVER**, then the roots in **CMAKE_FIND_ROOT_PATH** will be ignored and only the host system root will be used. If set to **BOTH**, then the host system paths and the paths in **CMAKE_FIND_ROOT_PATH** will be searched.

CMAKE_FIND_ROOT_PATH_MODE_PACKAGE

This variable controls whether the **CMAKE_FIND_ROOT_PATH** and **CMAKE_SYSROOT** are used by **find_package()**.

If set to **ONLY**, then only the roots in **CMAKE_FIND_ROOT_PATH** will be searched. If set to **NEVER**, then the roots in **CMAKE_FIND_ROOT_PATH** will be ignored and only the host system root will be used. If set to **BOTH**, then the host system paths and the paths in **CMAKE_FIND_ROOT_PATH** will be searched.

CMAKE_FIND_ROOT_PATH_MODE_PROGRAM

This variable controls whether the **CMAKE_FIND_ROOT_PATH** and **CMAKE_SYSROOT** are used by **find_program()**.

If set to **ONLY**, then only the roots in **CMAKE_FIND_ROOT_PATH** will be searched. If set to **NEVER**, then the roots in **CMAKE_FIND_ROOT_PATH** will be ignored and only the host system root will be used. If set to **BOTH**, then the host system paths and the paths in **CMAKE_FIND_ROOT_PATH** will be searched.

CMAKE_FRAMEWORK_PATH

Search path for OS X frameworks used by the **find_library()**, **find_package()**, **find_path()**, and **find_file()** commands.

CMAKE_IGNORE_PATH

Path to be ignored by FIND_XXX() commands.

Specifies directories to be ignored by searches in FIND_XXX() commands. This is useful in cross-compiled environments where some system directories contain incompatible but possibly linkable libraries. For example, on cross-compiled cluster environments, this allows a user to ignore directories containing libraries meant for the front-end machine that modules like FindX11 (and others) would normally search. By default this is empty; it is intended to be set by the project. Note that CMAKE_IGNORE_PATH takes a list of directory names, NOT a list of prefixes. If you want to ignore paths under prefixes (bin, include, lib, etc.), youll need to specify them explicitly. See also CMAKE_PREFIX_PATH, CMAKE_LIBRARY_PATH, CMAKE_INCLUDE_PATH, CMAKE_PROGRAM_PATH.

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_INCLUDE_DIRECTORIES_BEFORE

Whether to append or prepend directories by default in **include_directories()**.

This variable affects the default behavior of the **include_directories()** command. Setting this variable to ON is equivalent to using the BEFORE option in all uses of that command.

CMAKE_INCLUDE_DIRECTORIES_PROJECT_BEFORE

Whether to force prepending of project include directories.

This variable affects the order of include directories generated in compiler command lines. If set to ON, it causes the **CMAKE_SOURCE_DIR** and the **CMAKE_BINARY_DIR** to appear first.

CMAKE_INSTALL_DEFAULT_COMPONENT_NAME

Default component used in install() commands.

If an install() command is used without the COMPONENT argument, these files will be grouped into a default component. The name of this default install component will be taken from this variable. It defaults to Unspecified.

CMAKE_INSTALL_PREFIX

Install directory used by install.

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

On UNIX one can use the DESTDIR mechanism in order to relocate the whole installation. DESTDIR means DESTination DIRectory. It is commonly used by makefile users in order to install software at non-default location. It is usually invoked like this:

```
make DESTDIR=/home/john install
```

which will install the concerned software using the installation prefix, e.g. /usr/local prepended with the DESTDIR value which finally gives /home/john/usr/local.

WARNING: DESTDIR may not be used on Windows because installation prefix usually contains a drive letter like in C:/Program Files which cannot be prepended with some other prefix.

The installation prefix is also added to CMAKE_SYSTEM_PREFIX_PATH so that find_package, find_program, find_library, find_path, and find_file will search the prefix for other software.

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 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

List of directories to search for CMake modules.

Commands like include() and find_package() search for files in directories listed by this variable before checking the default modules that come with CMake.

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_POLICY_DEFAULT_CMP<NNNN>

Default for CMake Policy CMP<NNNN> when it is otherwise left unset.

Commands cmake_minimum_required(VERSION) and cmake_policy(VERSION) by default leave policies introduced after the given version unset. Set CMAKE_POLICY_DEFAULT_CMP<NNNN> to OLD or NEW to specify the default for policy CMP<NNNN>, where <NNNN> is the policy number.

This variable should not be set by a project in CMake code; use cmake_policy(SET) instead. Users running CMake may set this variable in the cache (e.g. -DCMAKE_POLICY_DEFAULT_CMP<NNNN>=<OLD|NEW>) to set a policy not otherwise set by the project. Set to OLD to quiet a policy warning while using old behavior or to NEW to try building the project with new behavior.

CMAKE_POLICY_WARNING_CMP<NNNN>

Explicitly enable or disable the warning when CMake Policy **CMP<NNNN>** is not set. This is meaningful only for the few policies that do not warn by default:

- **CMAKE_POLICY_WARNING_CMP0025** controls the warning for policy **CMP0025**.
- **CMAKE_POLICY_WARNING_CMP0047** controls the warning for policy **CMP0047**.

This variable should not be set by a project in CMake code. Project developers running CMake may set this variable in their cache to enable the warning (e.g. **-DCMAKE_POLICY_WARNING_CMP<NNNN>=ON**). Alternatively, running **cmake(1)** with the **--debug-output** or **--trace** option will also enable the warning.

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_PROJECT_<PROJECT–NAME>_INCLUDE
A CMake language file or module to be included by the **project()** command. This is is intended for injecting custom code into project builds without modifying their source.

CMAKE_SKIP_INSTALL_ALL_DEPENDENCY
Dont 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_STAGING_PREFIX
This variable may be set to a path to install to when cross-compiling. This can be useful if the path in **CMAKE_SYSROOT** is read-only, or otherwise should remain pristine.

The CMAKE_STAGING_PREFIX location is also used as a search prefix by the **find_*** commands. This can be controlled by setting the **CMAKE_FIND_NO_INSTALL_PREFIX** variable.

If any RPATH/RUNPATH entries passed to the linker contain the CMAKE_STAGING_PREFIX, the matching path fragments are replaced with the **CMAKE_INSTALL_PREFIX**.

CMAKE_SYSTEM_IGNORE_PATH
Path to be ignored by FIND_XXX() commands.

Specifies directories to be ignored by searches in FIND_XXX() commands. This is useful in cross-compiled environments where some system directories contain incompatible but possibly linkable libraries. For example, on cross-compiled cluster environments, this allows a user to ignore directories containing libraries meant for the front-end machine that modules like FindX11 (and others) would normally search. By default this contains a list of directories containing incompatible binaries for the host system. See also CMAKE_SYSTEM_PREFIX_PATH, CMAKE_SYSTEM_LIBRARY_PATH, CMAKE_SYSTEM_INCLUDE_PATH, and CMAKE_SYSTEM_PROGRAM_PATH.

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_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, the CMAKE_INSTALL_PREFIX and the **CMAKE_STAGING_PREFIX**. 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, and CMAKE_SYSTEM_IGNORE_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 CMake file that overrides platform information.

CMake loads the specified file while enabling support for each language from either the project() or enable_language() commands. It is loaded after CMakes builtin compiler and platform information modules have been loaded but before the information is used. The file may set platform information variables to override CMakes defaults.

This feature is intended for use only in overriding information variables that must be set before CMake builds its first test project to check that the compiler for a language works. It should not be used to load a file in cases that a normal include() will work. Use it only as a last resort for behavior that cannot be achieved any other way. For example, one may set CMAKE_C_FLAGS_INIT to change the default value used to initialize CMAKE_C_FLAGS before it is cached. The override file should NOT be used to set anything that could be set after languages are enabled, such as variables like CMAKE_RUNTIME_OUTPUT_DIRECTORY that affect the placement of binaries. Information set in the file will be used for try_compile and try_run builds too.

CMAKE_WARN_DEPRECATED

Whether to issue deprecation warnings for macros and functions.

If TRUE, this can be used by macros and functions to issue deprecation warnings. This variable is FALSE by default.

CMAKE_WARN_ON_ABSOLUTE_INSTALL_DESTINATION

Ask cmake_install.cmake script to warn each time a file with absolute INSTALL DESTINATION is encountered.

This variable is used by CMake-generated cmake_install.cmake scripts. If one sets this variable to ON while running the script, it may get warning messages from the script.

VARIABLES THAT DESCRIBE THE SYSTEM

APPLE

True if running on Mac OS X.

Set to true on Mac OS X.

BORLAND
True if the Borland compiler is being used.

This is set to 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 X operating systems.

Set to true when the host system is Apple OS X.

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
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_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_LIBRARY_ARCHITECTURE_REGEX
Regex matching possible target architecture library directory names.

This is used to detect CMAKE_<lang>_LIBRARY_ARCHITECTURE from the implicit linker search path by matching the <arch> name.

CMAKE_LIBRARY_ARCHITECTURE
Target architecture library directory name, if detected.

This is the value of CMAKE_<lang>_LIBRARY_ARCHITECTURE as detected for one of the enabled languages.

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
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
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_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.

Set to true when using Cygwin.

ENV
Access environment variables.

Use the syntax $ENV{VAR} to read environment variable VAR. See also the set() command to set ENV{VAR}.

MSVC10
True when using Microsoft Visual C 10.0

Set to true when the compiler is version 10.0 of Microsoft Visual C.

MSVC11
True when using Microsoft Visual C 11.0

Set to true when the compiler is version 11.0 of Microsoft Visual C.

MSVC12
True when using Microsoft Visual C 12.0

Set to true when the compiler is version 12.0 of Microsoft Visual C.

MSVC60

True when using Microsoft Visual C 6.0

Set to true when the compiler is version 6.0 of Microsoft Visual C.

MSVC70

True when using Microsoft Visual C 7.0

Set to true when the compiler is version 7.0 of Microsoft Visual C.

MSVC71

True when using Microsoft Visual C 7.1

Set to true when the compiler is version 7.1 of Microsoft Visual C.

MSVC80

True when using Microsoft Visual C 8.0

Set to true when the compiler is version 8.0 of Microsoft Visual C.

MSVC90

True when using Microsoft Visual C 9.0

Set to true when the compiler is version 9.0 of Microsoft Visual C.

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

True when using Microsoft Visual C

Set to true when the compiler is some version of Microsoft Visual C.

MSVC_VERSION

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

Known version numbers are:

```
1200 = VS 6.0
1300 = VS 7.0
1310 = VS 7.1
1400 = VS 8.0
1500 = VS 9.0
1600 = VS 10.0
1700 = VS 11.0
1800 = VS 12.0
```

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.

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 THAT CONTROL THE BUILD

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_AUTOMOC_MOC_OPTIONS

Additional options for **moc** when using **CMAKE_AUTOMOC**.

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

CMAKE_AUTOMOC

Whether to handle **moc** automatically for Qt targets.

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

CMAKE_AUTORCC

Whether to handle **rcc** automatically for Qt targets.

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

CMAKE_AUTORCC_OPTIONS

Whether to handle **rcc** automatically for Qt targets.

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

CMAKE_AUTOUIC

Whether to handle **uic** automatically for Qt targets.

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

CMAKE_AUTOUIC_OPTIONS

Whether to handle **uic** automatically for Qt targets.

This variable is used to initialize the **AUTOUIC_OPTIONS** 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_<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_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_<CONFIG>
Flags to be used when linking an executable.

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

CMAKE_EXE_LINKER_FLAGS
Linker flags to be used to create executables.

These flags will be used by the linker when creating an executable.

CMAKE_Fortran_FORMAT
Set to FIXED or FREE to indicate the Fortran source layout.

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

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_GNUtoMS
Convert GNU import libraries (.dll.a) to MS format (.lib).

This variable is used to initialize the GNUtoMS property on targets when they are created. See that target property for additional information.

CMAKE_INCLUDE_CURRENT_DIR_IN_INTERFACE
Automatically add the current source- and build directories to the INTERFACE_INCLUDE_DIRECTORIES.

If this variable is enabled, CMake automatically adds for each shared library target, static library target, module target and executable target, ${CMAKE_CURRENT_SOURCE_DIR} and ${CMAKE_CURRENT_BINARY_DIR} to the INTERFACE_INCLUDE_DIRECTORIES.By default CMAKE_INCLUDE_CURRENT_DIR_IN_INTERFACE is OFF.

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_<LANG>_VISIBILITY_PRESET
Default value for <LANG>_VISIBILITY_PRESET of targets.

This variable is used to initialize the <LANG>_VISIBILITY_PRESET property on all the targets. See that target property for additional information.

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 to be used to add a library search path to a compiler.

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

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

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

CMAKE_LINK_DEPENDS_NO_SHARED
Whether to skip link dependencies on shared library files.

This variable initializes the LINK_DEPENDS_NO_SHARED property on targets when they are created. See that target property for additional information.

CMAKE_LINK_INTERFACE_LIBRARIES
Default value for LINK_INTERFACE_LIBRARIES of targets.

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

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

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

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

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

CMAKE_MACOSX_BUNDLE
Default value for MACOSX_BUNDLE of targets.

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

CMAKE_MACOSX_RPATH
Whether to use rpaths on Mac OS X.

This variable is used to initialize the **MACOSX_RPATH** property on all targets.

CMAKE_MAP_IMPORTED_CONFIG_<CONFIG>
Default value for MAP_IMPORTED_CONFIG_<CONFIG> of targets.

This variable is used to initialize the MAP_IMPORTED_CONFIG_<CONFIG> property on all the targets. See that target property for additional information.

CMAKE_MODULE_LINKER_FLAGS_<CONFIG>
Flags to be used when linking a module.

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

CMAKE_MODULE_LINKER_FLAGS
Linker flags to be used to create modules.

These flags will be used by the linker when creating a module.

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_NO_SYSTEM_FROM_IMPORTED
Default value for NO_SYSTEM_FROM_IMPORTED of targets.

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

CMAKE_OSX_ARCHITECTURES
Target specific architectures for OS X.

This variable is used to initialize the **OSX_ARCHITECTURES** property on each target as it is creaed. See that target property for additional information.

The value of this variable should be set prior to the first **project()** or **enable_language()** command invocation because it may influence configuration of the toolchain and flags. It is intended to be set locally by the user creating a build tree.

This variable is ignored on platforms other than OS X.

CMAKE_OSX_DEPLOYMENT_TARGET
Specify the minimum version of OS X on which the target binaries are to be deployed. CMake uses this value for the **-mmacosx-version-min** flag and to help choose the default SDK (see **CMAKE_OSX_SYSROOT**).

If not set explicitly the value is initialized by the **MACOSX_DEPLOYMENT_TARGET** environment variable, if set, and otherwise computed based on the host platform.

The value of this variable should be set prior to the first **project()** or **enable_language()** command invocation because it may influence configuration of the toolchain and flags. It is intended to be set locally by the user creating a build tree.

This variable is ignored on platforms other than OS X.

CMAKE_OSX_SYSROOT
Specify the location or name of the OS X platform SDK to be used. CMake uses this value to compute the value of the **-isysroot** flag or equivalent and to help the **find_*** commands locate files in the SDK.

If not set explicitly the value is initialized by the **SDKROOT** environment variable, if set, and otherwise computed based on the **CMAKE_OSX_DEPLOYMENT_TARGET** or the host platform.

The value of this variable should be set prior to the first **project()** or **enable_language()** command invocation because it may influence configuration of the toolchain and flags. It is intended to be set locally by the user creating a build tree.

This variable is ignored on platforms other than OS X.

CMAKE_PDB_OUTPUT_DIRECTORY

Output directory for MS debug symbol **.pdb** files generated by the linker for executable and shared library targets.

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

CMAKE_PDB_OUTPUT_DIRECTORY_<CONFIG>

Per-configuration output directory for MS debug symbol **.pdb** files generated by the linker for executable and shared library targets.

This is a per-configuration version of **CMAKE_PDB_OUTPUT_DIRECTORY**. This variable is used to initialize the **PDB_OUTPUT_DIRECTORY_<CONFIG>** property on all the targets. See that target property for additional information.

CMAKE_POSITION_INDEPENDENT_CODE

Default value for POSITION_INDEPENDENT_CODE of targets.

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

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_SHARED_LINKER_FLAGS_<CONFIG>

Flags to be used when linking a shared library.

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

CMAKE_SHARED_LINKER_FLAGS

Linker flags to be used to create shared libraries.

These flags will be used by the linker when creating a shared library.

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_SKIP_INSTALL_RPATH

Do not include RPATHs in the install 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 installed without RPATH, even if RPATH is enabled when building. This can be useful for example to allow running tests from the build directory with RPATH enabled before the installation step. To omit RPATH in both the build and install steps, use CMAKE_SKIP_RPATH instead.

CMAKE_STATIC_LINKER_FLAGS_<CONFIG>

Flags to be used when linking a static library.

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

CMAKE_STATIC_LINKER_FLAGS

Linker flags to be used to create static libraries.

These flags will be used by the linker when creating a static library.

CMAKE_TRY_COMPILE_CONFIGURATION

Build configuration used for try_compile and try_run projects.

Projects built by try_compile and try_run are built synchronously during the CMake configuration step. Therefore a specific build configuration must be chosen even if the generated build system supports multiple configurations.

CMAKE_USE_RELATIVE_PATHS

Use relative paths (May not work!).

If this is set to TRUE, then 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.

CMAKE_VISIBILITY_INLINES_HIDDEN

Default value for VISIBILITY_INLINES_HIDDEN of targets.

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

CMAKE_WIN32_EXECUTABLE

Default value for WIN32_EXECUTABLE of targets.

This variable is used to initialize the WIN32_EXECUTABLE property on all the targets. See that target property for additional information.

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 FOR LANGUAGES

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. Unlike the other per-language variables, this uses the GNU syntax for identifying languages instead of the CMake syntax. Recognized values of the <LANG> suffix are:

```
CC = C compiler
CXX = C++ compiler
G77 = Fortran compiler
```

CMAKE_Fortran_MODDIR_DEFAULT

Fortran default module output directory.

Most Fortran compilers write .mod files to the current working directory. For those that do not, this is set to . and used when the Fortran_MODULE_DIRECTORY target property is not set.

CMAKE_Fortran_MODDIR_FLAG

Fortran flag for module output directory.

This stores the flag needed to pass the value of the Fortran_MODULE_DIRECTORY target property to the compiler.

CMAKE_Fortran_MODOUT_FLAG
Fortran flag to enable module output.

Most Fortran compilers write .mod files out by default. For others, this stores the flag needed to enable module output.

CMAKE_INTERNAL_PLATFORM_ABI
An internal variable subject to change.

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

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>_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 the language <LANG>.

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
Compiler identification string.

A short string unique to the compiler vendor. Possible values include:

```
Absoft = Absoft Fortran (absoft.com)
ADSP = Analog VisualDSP++ (analog.com)
AppleClang = Apple Clang (apple.com)
Clang = LLVM Clang (clang.llvm.org)
Cray = Cray Compiler (cray.com)
Embarcadero, Borland = Embarcadero (embarcadero.com)
G95 = G95 Fortran (g95.org)
GNU = GNU Compiler Collection (gcc.gnu.org)
HP = Hewlett-Packard Compiler (hp.com)
Intel = Intel Compiler (intel.com)
MIPSpro = SGI MIPSpro (sgi.com)
MSVC = Microsoft Visual Studio (microsoft.com)
```

```
PGI = The Portland Group (pgroup.com)
PathScale = PathScale (pathscale.com)
SDCC = Small Device C Compiler (sdcc.sourceforge.net)
SunPro = Oracle Solaris Studio (oracle.com)
TI = Texas Instruments (ti.com)
TinyCC = Tiny C Compiler (tinycc.org)
Watcom = Open Watcom (openwatcom.org)
XL, VisualAge, zOS = IBM XL (ibm.com)
```

This variable is not guaranteed to be defined for all compilers or languages.

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>_COMPILER
The full path to the compiler for LANG.

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

CMAKE_<LANG>_COMPILER_EXTERNAL_TOOLCHAIN
The external toolchain for cross-compiling, if supported.

Some compiler toolchains do not ship their own auxilliary utilities such as archivers and linkers. The compiler driver may support a command-line argument to specify the location of such tools. CMAKE_<LANG>_COMPILER_EXTERNAL_TOOLCHAIN may be set to a path to a path to the external toolchain and will be passed to the compiler driver if supported.

This variable may only be set in a toolchain file specified by the **CMAKE_TOOLCHAIN_FILE** variable.

CMAKE_<LANG>_COMPILER_TARGET
The target for cross-compiling, if supported.

Some compiler drivers are inherently cross-compilers, such as clang and QNX qcc. These compiler drivers support a command-line argument to specify the target to cross-compile for.

This variable may only be set in a toolchain file specified by the **CMAKE_TOOLCHAIN_FILE** variable.

CMAKE_<LANG>_COMPILER_VERSION
Compiler version string.

Compiler version in major[.minor[.patch[.tweak]]] format. This variable is not guaranteed to be defined for all compilers or languages.

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>_FLAGS
Flags for all build types.

<LANG> flags used regardless of the value of CMAKE_BUILD_TYPE.

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 compilers language. CMake automatically detects these directories for each language and reports the results in this variable.

When a library in one of these directories is given by full path to target_link_libraries() CMake will generate the -l<name> form on link lines to ensure the linker searches its implicit directories for the library. Note that some toolchains read implicit directories from an environment variable such as LIBRARY_PATH so keep its value consistent when operating in a given build tree.

CMAKE_<LANG>_IMPLICIT_LINK_FRAMEWORK_DIRECTORIES
Implicit linker framework search path detected for language <LANG>.

These paths are implicit linker framework search directories for the compilers 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 compilers language. CMake automatically detects these libraries and flags for each language and reports the results in this variable.

CMAKE_<LANG>_LIBRARY_ARCHITECTURE
Target architecture library directory name detected for <lang>.

If the <lang> compiler passes to the linker an architecture-specific system library search directory such as <prefix>/lib/<arch> this variable contains the <arch> name if/as detected by CMake.

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>_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>_LINK_EXECUTABLE
Rule variable to link an executable.

Rule variable to link an 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>_SIMULATE_ID
Identification string of simulated compiler.

Some compilers simulate other compilers to serve as drop-in replacements. When CMake detects such a compiler it sets this variable to what would have been the CMAKE_<LANG>_COMPILER_ID for the simulated compiler.

CMAKE_<LANG>_SIMULATE_VERSION
Version string of simulated compiler.

Some compilers simulate other compilers to serve as drop-in replacements. When CMake detects such a compiler it sets this variable to what would have been the CMAKE_<LANG>_COMPILER_VERSION for the simulated compiler.

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_USER_MAKE_RULES_OVERRIDE_<LANG>

Specify a CMake file that overrides platform information for <LANG>.

This is a language-specific version of CMAKE_USER_MAKE_RULES_OVERRIDE loaded only when enabling language <LANG>.

VARIABLES FOR CPACK

CPACK_ABSOLUTE_DESTINATION_FILES

List of files which have been installed using an ABSOLUTE DESTINATION path.

This variable is a Read-Only variable which is set internally by CPack during installation and before packaging using CMAKE_ABSOLUTE_DESTINATION_FILES defined in cmake_install.cmake scripts. The value can be used within CPack project configuration file and/or CPack<GEN>.cmake file of <GEN> generator.

CPACK_COMPONENT_INCLUDE_TOPLEVEL_DIRECTORY

Boolean toggle to include/exclude top level directory (component case).

Similar usage as CPACK_INCLUDE_TOPLEVEL_DIRECTORY but for the component case. See CPACK_INCLUDE_TOPLEVEL_DIRECTORY documentation for the detail.

CPACK_ERROR_ON_ABSOLUTE_INSTALL_DESTINATION

Ask CPack to error out as soon as a file with absolute INSTALL DESTINATION is encountered.

The fatal error is emitted before the installation of the offending file takes place. Some CPack generators, like NSIS,enforce this internally. This variable triggers the definition ofC-MAKE_ERROR_ON_ABSOLUTE_INSTALL_DESTINATION when CPack runsVariables common to all CPack generators

CPACK_INCLUDE_TOPLEVEL_DIRECTORY

Boolean toggle to include/exclude top level directory.

When preparing a package CPack installs the item under the so-called top level directory. The purpose of is to include (set to 1 or ON or TRUE) the top level directory in the package or not (set to 0 or OFF or FALSE).

Each CPack generator has a built-in default value for this variable. E.g. Archive generators (ZIP, TGZ, ...) includes the top level whereas RPM or DEB dont. The user may override the default value by setting this variable.

There is a similar variable CPACK_COMPONENT_INCLUDE_TOPLEVEL_DIRECTORY which may be used to override the behavior for the component packaging case which may have different default value for historical (now backward compatibility) reason.

CPACK_INSTALL_SCRIPT

Extra CMake script provided by the user.

If set this CMake script will be executed by CPack during its local [CPack-private] installation which is done right before packaging the files. The script is not called by e.g.: make install.

CPACK_PACKAGING_INSTALL_PREFIX

The prefix used in the built package.

Each CPack generator has a default value (like /usr). This default value may be overwritten from the CMakeLists.txt or the cpack command line by setting an alternative value.

e.g. set(CPACK_PACKAGING_INSTALL_PREFIX /opt)

This is not the same purpose as CMAKE_INSTALL_PREFIX which is used when installing from the build tree without building a package.

CPACK_SET_DESTDIR

Boolean toggle to make CPack use DESTDIR mechanism when packaging.

DESTDIR means DESTination DIRectory. It is commonly used by makefile users in order to install software at non-default location. It is a basic relocation mechanism that should not be used on Windows (see CMAKE_INSTALL_PREFIX documentation). It is usually invoked like this:

```
make DESTDIR=/home/john install
```

which will install the concerned software using the installation prefix, e.g. /usr/local prepended with the DESTDIR value which finally gives /home/john/usr/local. When preparing a package, CPack first installs the items to be packaged in a local (to the build tree) directory by using the same DESTDIR mechanism. Nevertheless, if CPACK_SET_DESTDIR is set then CPack will set DESTDIR before doing the local install. The most noticeable difference is that without CPACK_SET_DESTDIR, CPack uses CPACK_PACKAGING_INSTALL_PREFIX as a prefix whereas with CPACK_SET_DESTDIR set, CPack will use CMAKE_INSTALL_PREFIX as a prefix.

Manually setting CPACK_SET_DESTDIR may help (or simply be necessary) if some install rules uses absolute DESTINATION (see CMake INSTALL command). However, starting with CPack/CMake 2.8.3 RPM and DEB installers tries to handle DESTDIR automatically so that it is seldom necessary for the user to set it.

CPACK_WARN_ON_ABSOLUTE_INSTALL_DESTINATION

Ask CPack to warn each time a file with absolute INSTALL DESTINATION is encountered.

This variable triggers the definition of CMAKE_WARN_ON_ABSOLUTE_INSTALL_DESTINATION when CPack runs cmake_install.cmake scripts.

COPYRIGHT