Appendix E - Properties

This is the documentation for the properties supported by CMake. Properties can have different scopes. They can be assigned to a source file, a directory, a target, test, etc. By modifying the values of properties the behaviour of the build system can be customized.

Properties of Global Scope

ALLOW_DUPLICATE_CUSTOM_TARGETS: Allow duplicate custom targets to be created.

Normally CMake requires that all targets built in a project have globally unique logical names (see policy CMP0002). This is necessary to generate meaningful project file names in Xcode and VS IDE generators. It also allows the target names to be referenced unambiguously.

Makefile generators are capable of supporting duplicate custom target names. For projects that care only about Makefile generators and do not wish to support Xcode or VS IDE generators, one may set this property to true to allow duplicate custom targets. The property allows multiple add_custom_target command calls in different directories to specify the same target name. However, setting this property will cause non-Makefile generators to produce an error and refuse to generate the project.

DEBUG_CONFIGURATIONS: Specify which configurations are for debugging.

The value must be a semi-colon separated list of configuration names. Currently this property is used only by the target_link_libraries command (see its documentation for details). Additional uses may be defined in the future.

This property must be set at the top level of the project and before the first target_link_libraries command invocation. If any entry in the list does not match a valid configuration for the project the behavior is undefined.

DISABLED_FEATURES: List of features which are disabled during the CMake run.

List of features which are disabled during the CMake run. Be default it contains the names of all packages which were not found. This is determined using the <NAME>_FOUND variables. Packages which are searched QUIET are not listed. A project can add its own features to this list.This property is used by the macros in FeatureSummary.cmake.

ENABLED_FEATURES: List of features which are enabled during the CMake run.

List of features which are enabled during the CMake run. Be default it contains the names of all packages which were found. This is determined using the <NAME>_FOUND variables. Packages which are searched QUIET are not listed. A project can add its own features to this list.This property is used by the macros in FeatureSummary.cmake.

ENABLED_LANGUAGES: Read-only property that contains the list of currently enabled languages

Set to list of currently enabled languages.

FIND_LIBRARY_USE_LIB64_PATHS: Whether FIND_LIBRARY should automatically search lib64 directories.

FIND_LIBRARY_USE_LIB64_PATHS is a boolean specifying whether the FIND_LIBRARY command should automatically search the lib64 variant of directories called lib in the search path when building 64-bit binaries.

FIND_LIBRARY_USE_OPENBSD_VERSIONING: Whether FIND_LIBRARY should find OpenBSD-style shared libraries.

This property is a boolean specifying whether the FIND_LIBRARY command should find shared libraries with OpenBSD-style versioned extension: ".so.<major>.<minor>". The property is set to true on OpenBSD and false on other platforms.

GLOBAL_DEPENDS_DEBUG_MODE: Enable global target dependency graph debug mode.

CMake automatically analyzes the global inter-target dependency graph at the beginning of native build system generation. This property causes it to display details of its analysis to stderr.

GLOBAL_DEPENDS_NO_CYCLES: Disallow global target dependency graph cycles.

CMake automatically analyzes the global inter-target dependency graph at the beginning of native build system generation. It reports an error if the dependency graph contains a cycle that does not consist of all STATIC library targets. This property tells CMake to disallow all cycles completely, even among static libraries.

IN_TRY_COMPILE: Read-only property that is true during a try-compile configuration.

True when building a project inside a TRY_COMPILE or TRY_RUN command.

PACKAGES_FOUND: List of packages which were found during the CMake run.

List of packages which were found during the CMake run. Whether a package has been found is determined using the <NAME>_FOUND variables.

PACKAGES_NOT_FOUND: List of packages which were not found during the CMake run.

List of packages which were not found during the CMake run. Whether a package has been found is determined using the <NAME>_FOUND variables.

REPORT_UNDEFINED_PROPERTIES: If set, report any undefined properties to this file.

If this property is set to a filename then when CMake runs it will report any properties or variables that were accessed but not defined into the filename specified in this property.

RULE_LAUNCH_COMPILE: Specify a launcher for compile rules.

Makefile generators prefix compiler commands with the given launcher command line. This is intended to allow launchers to intercept build problems with high granularity. Non-Makefile generators currently ignore this property.

RULE_LAUNCH_CUSTOM: Specify a launcher for custom rules.

Makefile generators prefix custom commands with the given launcher command line. This is intended to allow launchers to intercept build problems with high granularity. Non-Makefile generators currently ignore this property.

RULE_LAUNCH_LINK: Specify a launcher for link rules.

Makefile generators prefix link and archive commands with the given launcher command line. This is intended to allow launchers to intercept build problems with high granularity. Non-Makefile generators currently ignore this property.

RULE_MESSAGES: Specify whether to report a message for each make rule.

This property specifies whether Makefile generators should add a progress message describing what each build rule does. If the property is not set the default is ON. Set the property to OFF to disable granular messages and report only as each target completes. This is intended to allow scripted builds to avoid the build time cost of detailed reports. If a

CMAKE_RULE_MESSAGES cache entry exists its value initializes the value of this property. Non-Makefile generators currently ignore this property.

TARGET_ARCHIVES_MAY_BE_SHARED_LIBS: Set if shared libraries may be named like archives.

On AIX shared libraries may be named "lib<name>.a". This property is set to true on such platforms.

TARGET_SUPPORTS_SHARED_LIBS: Does the target platform support shared libraries.

TARGET_SUPPORTS_SHARED_LIBS is a boolean specifying whether the target platform supports shared libraries. Basically all current general general purpose OS do so, the exception are usually embedded systems with no or special OSs.

__CMAKE_DELETE_CACHE_CHANGE_VARS_: Internal property

Used to detect compiler changes, Do not set.

Properties on Directories

ADDITIONAL_MAKE_CLEAN_FILES: Additional files to clean during the make clean stage.

A list of files that will be cleaned as a part of the "make clean" stage.

CACHE_VARIABLES: List of cache variables available in the current directory.

This read-only property specifies the list of CMake cache variables currently defined. It is intended for debugging purposes.

CLEAN_NO_CUSTOM: Should the output of custom commands be left.

If this is true then the outputs of custom commands for this directory will not be removed during the "make clean" stage.

COMPILE_DEFINITIONS: Preprocessor definitions for compiling a directory's sources.

The COMPILE_DEFINITIONS property may be set to a semicolon-separated list of preprocessor definitions using the syntax VAR or VAR=value. Function-style definitions are not supported. CMake will automatically escape the value correctly for the native build system (note that CMake language syntax may require escapes to specify some values). This property may be set on a per-configuration basis using the name COMPILE_DEFINITIONS_<CONFIG> where <CONFIG> is an upper-case name (ex. "COMPILE_DEFINITIONS_DEBUG"). This property will be initialized in each directory by its value in the directory's parent.

CMake will automatically drop some definitions that are not supported by the native build tool. The VS6 IDE does not support definition values with spaces (but NMake does).

Dislaimer: Most native build tools have poor support for escaping certain values. CMake has work-arounds for many cases but some values may just not be possible to pass correctly. If a value does not seem to be escaped correctly, do not attempt to work-around the problem by adding escape sequences to the value. Your work-around may break in a future version of CMake that has improved escape support. Instead consider defining the macro in a (configured) header file. Then report the limitation.

COMPILE_DEFINITIONS_<CONFIG>: Per-configuration preprocessor definitions in a directory.

This is the configuration-specific version of COMPILE_DEFINITIONS. This property will be initialized in each directory by its value in the directory's parent.

DEFINITIONS: For CMake 2.4 compatibility only. Use COMPILE_DEFINITIONS instead.

This read-only property specifies the list of flags given so far to the add_definitions command. It is intended for debugging purposes. Use the COMPILE_DEFINITIONS instead.

EXCLUDE_FROM_ALL: Exclude the directory from the all target of its parent.

A property on a directory that indicates if its targets are excluded from the default build target. If it is not, then with a Makefile for example typing make will cause the targets to be built. The same concept applies to the default build of other generators.

IMPLICIT_DEPENDS_INCLUDE_TRANSFORM: Specify #include line transforms for dependencies in a directory.

This property specifies rules to transform macro-like #include lines during implicit dependency scanning of C and C++ source files. The list of rules must be semicolon-separated with each entry of the form "A_MACRO(%)=value-with-%" (the % must be literal). During dependency scanning occurrences of A_MACRO(...) on #include lines will be replaced by the value given with the macro argument substituted for '%'. For example, the entry

```
MYDIR(%)=<mydir/%>
```

will convert lines of the form

```
#include MYDIR(myheader.h)
```

to

```
#include <mydir/myheader.h>
```

allowing the dependency to be followed. This property applies to sources in all targets within a directory. The property value is initialized in each directory by its value in the directory's parent.

INCLUDE_DIRECTORIES: List of preprocessor include file search directories.

This read-only property specifies the list of directories given so far to the include_directories command. It is intended for debugging purposes.

INCLUDE_REGULAR_EXPRESSION: Include file scanning regular expression.

This read-only property specifies the regular expression used during dependency scanning to match include files that should be followed. See the include_regular_expression command.

LINK_DIRECTORIES: List of linker search directories.

This read-only property specifies the list of directories given so far to the link_directories command. It is intended for debugging purposes.

LISTFILE_STACK: The current stack of listfiles being processed.

This property is mainly useful when trying to debug errors in your CMake scripts. It returns a list of what list files are currently being processed, in order. So if one listfile does an INCLUDE command then that is effectively pushing the included listfile onto the stack.

MACROS: List of macro commands available in the current directory.

This read-only property specifies the list of CMake macros currently defined. It is intended for debugging purposes. See the macro command.

PARENT_DIRECTORY: Source directory that added current subdirectory.

This read-only property specifies the source directory that added the current source directory as a subdirectory of the build. In the top-level directory the value is the empty-string.

RULE_LAUNCH_COMPILE: Specify a launcher for compile rules.

See the global property of the same name for details. This overrides the global property for a directory.

RULE_LAUNCH_CUSTOM: Specify a launcher for custom rules.

See the global property of the same name for details. This overrides the global property for a directory.

RULE_LAUNCH_LINK: Specify a launcher for link rules.

See the global property of the same name for details. This overrides the global property for a directory.

TEST_INCLUDE_FILE: A cmake file that will be included when ctest is run.

If you specify TEST_INCLUDE_FILE, that file will be included and processed when ctest is run on the directory.

VARIABLES: List of variables defined in the current directory.

This read-only property specifies the list of CMake variables currently defined. It is intended for debugging purposes.

Properties on Targets

<CONFIG>_OUTPUT_NAME: Old per-configuration target file base name.

This is a configuration-specific version of OUTPUT_NAME. Use OUTPUT_NAME_<CONFIG> instead.

<CONFIG>_POSTFIX: Postfix to append to the target file name for configuration <CONFIG>.

When building with configuration <CONFIG> the value of this property is appended to the target file name built on disk. For non-executable targets, this property is initialized by the value of the variable CMAKE_<CONFIG>_POSTFIX if it is set when a target is created. This property is ignored on the Mac for Frameworks and App Bundles.

ARCHIVE_OUTPUT_DIRECTORY: Output directory in which to build ARCHIVE target files.

This property specifies the directory into which archive target files should be built. There are three kinds of target files that may be built: archive, library, and runtime. Executables are always treated as runtime targets. Static libraries are always treated as archive targets. Module libraries are always treated as library targets. For non-DLL platforms shared libraries are treated as library targets. For DLL platforms the DLL part of a shared library is treated as a runtime target and the corresponding import library is treated as an archive target. All Windows-based systems including Cygwin are DLL platforms. This property is initialized by the value of the variable CMAKE_ARCHIVE_OUTPUT_DIRECTORY if it is set when a target is created.

ARCHIVE_OUTPUT_NAME: Output name for ARCHIVE target files.

This property specifies the base name for archive target files. It overrides OUTPUT_NAME and OUTPUT_NAME_<CONFIG> properties. There are three kinds of target files that may be built: archive, library, and runtime. Executables are always treated as runtime targets. Static libraries are always treated as archive targets. Module libraries are always treated as library targets. For non-DLL platforms shared libraries are treated as library targets. For DLL platforms the DLL part of a shared library is treated as a runtime target and the corresponding

import library is treated as an archive target. All Windows-based systems including Cygwin are DLL platforms.

ARCHIVE_OUTPUT_NAME_<CONFIG>: Per-configuration output name for ARCHIVE target files.

This is the configuration-specific version of ARCHIVE_OUTPUT_NAME.

BUILD_WITH_INSTALL_RPATH: Should build tree targets have install tree rpaths.

BUILD_WITH_INSTALL_RPATH is a boolean specifying whether to link the target in the build tree with the INSTALL_RPATH. This takes precedence over SKIP_BUILD_RPATH and avoids the need for relinking before installation. This property is initialized by the value of the variable CMAKE_BUILD_WITH_INSTALL_RPATH if it is set when a target is created.

COMPILE_DEFINITIONS: Preprocessor definitions for compiling a target's sources.

The COMPILE_DEFINITIONS property may be set to a semicolon-separated list of preprocessor definitions using the syntax VAR or VAR=value. Function-style definitions are not supported. CMake will automatically escape the value correctly for the native build system (note that CMake language syntax may require escapes to specify some values). This property may be set on a per-configuration basis using the name COMPILE_DEFINITIONS_<CONFIG> where <CONFIG> is an upper-case name (ex. "COMPILE_DEFINITIONS_DEBUG").

CMake will automatically drop some definitions that are not supported by the native build tool. The VS6 IDE does not support definition values with spaces (but NMake does).

Dislaimer: Most native build tools have poor support for escaping certain values. CMake has work-arounds for many cases but some values may just not be possible to pass correctly. If a value does not seem to be escaped correctly, do not attempt to work-around the problem by adding escape sequences to the value. Your work-around may break in a future version of CMake that has improved escape support. Instead consider defining the macro in a (configured) header file. Then report the limitation.

COMPILE_DEFINITIONS_<CONFIG>: Per-configuration preprocessor definitions on a target.

This is the configuration-specific version of COMPILE_DEFINITIONS.

COMPILE_FLAGS: Additional flags to use when compiling this target's sources.

The COMPILE_FLAGS property sets additional compiler flags used to build sources within the target. Use COMPILE_DEFINITIONS to pass additional preprocessor definitions.

DEBUG_POSTFIX: See target property <CONFIG>_POSTFIX.

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

DEFINE_SYMBOL: Define a symbol when compiling this target's sources.

DEFINE_SYMBOL sets the name of the preprocessor symbol defined when compiling sources in a shared library. If not set here then it is set to target_EXPORTS by default (with some substitutions if the target is not a valid C identifier). This is useful for headers to know whether they are being included from inside their library our outside to properly setup dllexport/dllimport decorations.

ENABLE_EXPORTS: Specify whether an executable exports symbols for loadable modules.

Normally an executable does not export any symbols because it is the final program. It is possible for an executable to export symbols to be used by loadable modules. When this property is set to true CMake will allow other targets to "link" to the executable with the TARGET_LINK_LIBRARIES command. On all platforms a target-level dependency on the executable is created for targets that link to it. For non-DLL platforms the link rule is simply ignored since the dynamic loader will automatically bind symbols when the module is loaded. For DLL platforms an import library will be created for the exported symbols and then used for linking. All Windows-based systems including Cygwin are DLL platforms.

EXCLUDE_FROM_ALL: Exclude the target from the all target.

A property on a target that indicates if the target is excluded from the default build target. If it is not, then with a Makefile for example typing make will cause this target to be built. The same concept applies to the default build of other generators. Installing a target with EXCLUDE_FROM_ALL set to true has undefined behavior.

EchoString: A message to be displayed when the target is built.

A message to display on some generators (such as Makefiles) when the target is built.

FRAMEWORK: This target is a framework on the Mac.

If a shared library target has this property set to true it will be built as a framework when built on the mac. It will have the directory structure required for a framework and will be suitable to be used with the -framework option

Fortran_MODULE_DIRECTORY: Specify output directory for Fortran modules provided by the target.

If the target contains Fortran source files that provide modules and the compiler supports a module output directory this specifies the directory in which the modules will be placed. When this property is not set the modules will be placed in the build directory corresponding to the target's source directory. If the variable CMAKE_Fortran_MODULE_DIRECTORY is set when a target is created its value is used to initialize this property.

GENERATOR_FILE_NAME: Generator's file for this target.

An internal property used by some generators to record the name of project or dsp file associated with this target.

HAS_CXX: Link the target using the C++ linker tool (obselete).

This is equivalent to setting the LINKER_LANGUAGE property to CXX. See that property's documentation for details.

IMPLICIT_DEPENDS_INCLUDE_TRANSFORM: Specify #include line transforms for dependencies in a target.

This property specifies rules to transform macro-like #include lines during implicit dependency scanning of C and C++ source files. The list of rules must be semicolon-separated with each entry of the form "A_MACRO(%)=value-with-%" (the % must be literal). During dependency scanning occurrences of A_MACRO(...) on #include lines will be replaced by the value given with the macro argument substituted for '%'. For example, the entry

```
MYDIR(%)=<mydir/%>
```

will convert lines of the form

```
#include MYDIR(myheader.h)
```

to

```
#include <mydir/myheader.h>
```

allowing the dependency to be followed. This property applies to sources in the target on which it is set.

IMPORTED: Read-only indication of whether a target is IMPORTED.

The boolean value of this property is true for targets created with the IMPORTED option to add_executable or add_library. It is false for targets built within the project.

IMPORTED_CONFIGURATIONS: Configurations provided for an IMPORTED target.

Lists configuration names available for an IMPORTED target. The names correspond to configurations defined in the project from which the target is imported. If the importing project uses a different set of configurations the names may be mapped using the MAP_IMPORTED_CONFIG_<CONFIG> property. Ignored for non-imported targets.

IMPORTED_IMPLIB: Full path to the import library for an IMPORTED target.

Specifies the location of the ".lib" part of a windows DLL. Ignored for non-imported targets.

IMPORTED_IMPLIB_<CONFIG>: Per-configuration version of IMPORTED_IMPLIB property.

This property is used when loading settings for the <CONFIG> configuration of an imported target. Configuration names correspond to those provided by the project from which the target is imported.

IMPORTED_LINK_DEPENDENT_LIBRARIES: Dependent shared libraries of an imported shared library.

Shared libraries may be linked to other shared libraries as part of their implementation. On some platforms the linker searches for the dependent libraries of shared libraries they are including in the link. This property lists the dependent shared libraries of an imported library. The list should be disjoint from the list of interface libraries in the IMPORTED_LINK_INTERFACE_LIBRARIES property. On platforms requiring dependent shared libraries to be found at link time CMake uses this list to add appropriate files or paths to the link command line. Ignored for non-imported targets.

IMPORTED_LINK_DEPENDENT_LIBRARIES_<CONFIG>: Per-configuration version of IMPORTED_LINK_DEPENDENT_LIBRARIES.

This property is used when loading settings for the <CONFIG> configuration of an imported target. Configuration names correspond to those provided by the project from which the target is imported. If set, this property completely overrides the generic property for the named configuration.

IMPORTED_LINK_INTERFACE_LANGUAGES: Languages compiled into an IMPORTED static library.

Lists languages of soure files compiled to produce a STATIC IMPORTED library (such as "C" or "CXX"). CMake accounts for these languages when computing how to link a target to the imported library. For example, when a C executable links to an imported C++ static library CMake chooses the C++ linker to satisfy language runtime dependencies of the static library.

This property is ignored for targets that are not STATIC libraries. This property is ignored for non-imported targets.

IMPORTED_LINK_INTERFACE_LANGUAGES_<CONFIG>: Per-configuration version of IMPORTED_LINK_INTERFACE_LANGUAGES.

This property is used when loading settings for the <CONFIG> configuration of an imported target. Configuration names correspond to those provided by the project from which the target

is imported. If set, this property completely overrides the generic property for the named configuration.

IMPORTED_LINK_INTERFACE_LIBRARIES: Transitive link interface of an IMPORTED target.

Lists libraries whose interface is included when an IMPORTED library target is linked to another target. The libraries will be included on the link line for the target. Unlike the LINK_INTERFACE_LIBRARIES property, this property applies to all imported target types, including STATIC libraries. This property is ignored for non-imported targets.

IMPORTED_LINK_INTERFACE_LIBRARIES_<CONFIG>: Per-configuration version of IMPORTED_LINK_INTERFACE_LIBRARIES.

This property is used when loading settings for the <CONFIG> configuration of an imported target. Configuration names correspond to those provided by the project from which the target is imported. If set, this property completely overrides the generic property for the named configuration.

IMPORTED_LINK_INTERFACE_MULTIPLICITY: Repetition count for cycles of IMPORTED static libraries.

This is LINK_INTERFACE_MULTIPLICITY for IMPORTED targets.

IMPORTED_LINK_INTERFACE_MULTIPLICITY_<CONFIG>: Per-configuration repetition count for cycles of IMPORTED archives.

This is the configuration-specific version of IMPORTED_LINK_INTERFACE_MULTIPLICITY. If set, this property completely overrides the generic property for the named configuration.

IMPORTED_LOCATION: Full path to the main file on disk for an IMPORTED target.

Specifies the location of an IMPORTED target file on disk. For executables this is the location of the executable file. For bundles on OS X this is the location of the executable file inside Contents/MacOS under the application bundle folder. For static libraries and modules this is the location of the library or module. For shared libraries on non-DLL platforms this is the location of the shared library. For frameworks on OS X this is the location of the library file symlink just inside the framework folder. For DLLs this is the location of the ".dll" part of the library. For UNKNOWN libraries this is the location of the file to be linked. Ignored for non-imported targets.

IMPORTED_LOCATION_<CONFIG>: Per-configuration version of IMPORTED_LOCATION property.

This property is used when loading settings for the <CONFIG> configuration of an imported target. Configuration names correspond to those provided by the project from which the target is imported.

IMPORTED_SONAME: The "soname" of an IMPORTED target of shared library type.

Specifies the "soname" embedded in an imported shared library. This is meaningful only on platforms supporting the feature. Ignored for non-imported targets.

IMPORTED_SONAME_<CONFIG>: Per-configuration version of IMPORTED_SONAME property.

This property is used when loading settings for the <CONFIG> configuration of an imported target. Configuration names correspond to those provided by the project from which the target is imported.

IMPORT_PREFIX: What comes before the import library name.

Similar to the target property PREFIX, but used for import libraries (typically corresponding to a DLL) instead of regular libraries. A target property that can be set to override the prefix (such as "lib") on an import library name.

IMPORT_SUFFIX: What comes after the import library name.

Similar to the target property SUFFIX, but used for import libraries (typically corresponding to a DLL) instead of regular libraries. A target property that can be set to override the suffix (such as ".lib") on an import library name.

INSTALL_NAME_DIR: Mac OSX directory name for installed targets.

INSTALL_NAME_DIR is a string specifying the directory portion of the "install_name" field of shared libraries on Mac OSX to use in the installed targets.

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 property is initialized by the value of the variable CMAKE_INSTALL_RPATH if it is set when a target is created.

INSTALL_RPATH_USE_LINK_PATH: Add paths to linker search and installed rpath.

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 property is initialized by the value of the variable CMAKE_INSTALL_RPATH_USE_LINK_PATH if it is set when a target is created.

LABELS: Specify a list of text labels associated with a target.

Target label semantics are currently unspecified.

LIBRARY_OUTPUT_DIRECTORY: Output directory in which to build LIBRARY target files.

This property specifies the directory into which library target files should be built. There are three kinds of target files that may be built: archive, library, and runtime. Executables are

always treated as runtime targets. Static libraries are always treated as archive targets. Module libraries are always treated as library targets. For non-DLL platforms shared libraries are treated as library targets. For DLL platforms the DLL part of a shared library is treated as a runtime target and the corresponding import library is treated as an archive target. All Windows-based systems including Cygwin are DLL platforms. This property is initialized by the value of the variable CMAKE_LIBRARY_OUTPUT_DIRECTORY if it is set when a target is created.

LIBRARY_OUTPUT_NAME: Output name for LIBRARY target files.

This property specifies the base name for library target files. It overrides OUTPUT_NAME and OUTPUT_NAME_<CONFIG> properties. There are three kinds of target files that may be built: archive, library, and runtime. Executables are always treated as runtime targets. Static libraries are always treated as archive targets. Module libraries are always treated as library targets. For non-DLL platforms shared libraries are treated as library targets. For DLL platforms the DLL part of a shared library is treated as a runtime target and the corresponding import library is treated as an archive target. All Windows-based systems including Cygwin are DLL platforms.

LIBRARY_OUTPUT_NAME_<CONFIG>: Per-configuration output name for LIBRARY target files.

This is the configuration-specific version of LIBRARY_OUTPUT_NAME.

LINKER_LANGUAGE: Specifies language whose compiler will invoke the linker.

For executables, shared libraries, and modules, this sets the language whose compiler is used to link the target (such as "C" or "CXX"). A typical value for an executable is the language of the source file providing the program entry point (main). If not set, the language with the highest linker preference value is the default. See documentation of CMAKE_<LANG>_LINKER_PREFERENCE variables.

LINK_FLAGS: Additional flags to use when linking this target.

The LINK_FLAGS property can be used to add extra flags to the link step of a target. LINK_FLAGS_<CONFIG> will add to the configuration <CONFIG>, for example, DEBUG, RELEASE, MINSIZEREL, RELWITHDEBINFO.

LINK_FLAGS_<CONFIG>: Per-configuration linker flags for a target.

This is the configuration-specific version of LINK_FLAGS.

LINK_INTERFACE_LIBRARIES: List public interface libraries for a shared library or executable.

By default linking to a shared library target transitively links to targets with which the library itself was linked. For an executable with exports (see the ENABLE_EXPORTS property) no default transitive link dependencies are used. This property replaces the default transitive link

dependencies with an explict list. When the target is linked into another target the libraries listed (and recursively their link interface libraries) will be provided to the other target also. If the list is empty then no transitive link dependencies will be incorporated when this target is linked into another target even if the default set is non-empty. This property is ignored for STATIC libraries.

LINK_INTERFACE_LIBRARIES_<CONFIG>: Per-configuration list of public interface libraries for a target.

This is the configuration-specific version of LINK_INTERFACE_LIBRARIES. If set, this property completely overrides the generic property for the named configuration.

LINK_INTERFACE_MULTIPLICITY: Repetition count for STATIC libraries with cyclic dependencies.

When linking to a STATIC library target with cyclic dependencies the linker may need to scan more than once through the archives in the strongly connected component of the dependency graph. CMake by default constructs the link line so that the linker will scan through the component at least twice. This property specifies the minimum number of scans if it is larger than the default. CMake uses the largest value specified by any target in a component.

LINK_INTERFACE_MULTIPLICITY_<CONFIG>: Per-configuration repetition count for cycles of STATIC libraries.

This is the configuration-specific version of LINK_INTERFACE_MULTIPLICITY. If set, this property completely overrides the generic property for the named configuration.

LINK_SEARCH_END_STATIC: End a link line such that static system libraries are used.

Some linkers support switches such as -Bstatic and -Bdynamic to determine whether to use static or shared libraries for -lXXX options. CMake uses these options to set the link type for libraries whose full paths are not known or (in some cases) are in implicit link directories for the platform. By default the linker search type is left at -Bdynamic by the end of the library list. This property switches the final linker search type to -Bstatic.

LOCATION: Read-only location of a target on disk.

For an imported target, this read-only property returns the value of the LOCATION_<CONFIG> property for an unspecified configuration <CONFIG> provided by the target.

For a non-imported target, this property is provided for compatibility with CMake 2.4 and below. It was meant to get the location of an executable target's output file for use in add_custom_command. The path may contain a build-system-specific portion that is replaced at build time with the configuration getting built (such as "$(ConfigurationName)" in VS). In CMake 2.6 and above add_custom_command automatically recognizes a target name in its

COMMAND and DEPENDS options and computes the target location. Therefore this property is not needed for creating custom commands.

LOCATION_<CONFIG>: Read-only property providing a target location on disk.

A read-only property that indicates where a target's main file is located on disk for the configuration <CONFIG>. The property is defined only for library and executable targets. An imported target may provide a set of configurations different from that of the importing project. By default CMake looks for an exact-match but otherwise uses an arbitrary available configuration. Use the MAP_IMPORTED_CONFIG_<CONFIG> property to map imported configurations explicitly.

MACOSX_BUNDLE: Build an executable as an application bundle on Mac OS X.

When this property is set to true the executable when built on Mac OS X will be created as an application bundle. This makes it a GUI executable that can be launched from the Finder. See the MACOSX_BUNDLE_INFO_PLIST target property for information about creation of the Info.plist file for the application bundle.

MACOSX_BUNDLE_INFO_PLIST: Specify a custom Info.plist template for a Mac OS X App Bundle.

An executable target with MACOSX_BUNDLE enabled will be built as an application bundle on Mac OS X. By default its Info.plist file is created by configuring a template called MacOSXBundleInfo.plist.in located in the CMAKE_MODULE_PATH. This property specifies an alternative template file name which may be a full path.

The following target properties may be set to specify content to be configured into the file:

```
MACOSX_BUNDLE_INFO_STRING
MACOSX_BUNDLE_ICON_FILE
MACOSX_BUNDLE_GUI_IDENTIFIER
MACOSX_BUNDLE_LONG_VERSION_STRING
MACOSX_BUNDLE_BUNDLE_NAME
MACOSX_BUNDLE_SHORT_VERSION_STRING
MACOSX_BUNDLE_BUNDLE_VERSION
MACOSX_BUNDLE_COPYRIGHT
```

CMake variables of the same name may be set to affect all targets in a directory that do not have each specific property set. If a custom Info.plist is specified by this property it may of course hard-code all the settings instead of using the target properties.

MACOSX_FRAMEWORK_INFO_PLIST: Specify a custom Info.plist template for a Mac OS X Framework.

An library target with FRAMEWORK enabled will be built as a framework on Mac OS X. By default its Info.plist file is created by configuring a template called MacOSXFrameworkInfo.plist.in located in the CMAKE_MODULE_PATH. This property specifies an alternative template file name which may be a full path.

The following target properties may be set to specify content to be configured into the file:

```
MACOSX_FRAMEWORK_ICON_FILE
MACOSX_FRAMEWORK_IDENTIFIER
MACOSX_FRAMEWORK_SHORT_VERSION_STRING
MACOSX_FRAMEWORK_BUNDLE_VERSION
```

CMake variables of the same name may be set to affect all targets in a directory that do not have each specific property set. If a custom Info.plist is specified by this property it may of course hard-code all the settings instead of using the target properties.

MAP_IMPORTED_CONFIG_<CONFIG>: Map from project configuration to IMPORTED target's configuration.

List configurations of an imported target that may be used for the current project's <CONFIG> configuration. Targets imported from another project may not provide the same set of configuration names available in the current project. Setting this property tells CMake what imported configurations are suitable for use when building the <CONFIG> configuration. The first configuration in the list found to be provided by the imported target is selected. If no matching configurations are available the imported target is considered to be not found. This property is ignored for non-imported targets.

OUTPUT_NAME: Output name for target files.

This sets the base name for output files created for an executable or library target. If not set, the logical target name is used by default.

OUTPUT_NAME_<CONFIG>: Per-configuration target file base name.

This is the configuration-specific version of OUTPUT_NAME.

POST_INSTALL_SCRIPT: Deprecated install support.

The PRE_INSTALL_SCRIPT and POST_INSTALL_SCRIPT properties are the old way to specify CMake scripts to run before and after installing a target. They are used only when the old INSTALL_TARGETS command is used to install the target. Use the INSTALL command instead.

PREFIX: What comes before the library name.

A target property that can be set to override the prefix (such as "lib") on a library name.

PRE_INSTALL_SCRIPT: Deprecated install support.

The PRE_INSTALL_SCRIPT and POST_INSTALL_SCRIPT properties are the old way to specify CMake scripts to run before and after installing a target. They are used only when the old INSTALL_TARGETS command is used to install the target. Use the INSTALL command instead.

PRIVATE_HEADER: Specify private header files in a FRAMEWORK shared library target.

Shared library targets marked with the FRAMEWORK property generate frameworks on OS X and normal shared libraries on other platforms. This property may be set to a list of header files to be placed in the PrivateHeaders directory inside the framework folder. On non-Apple platforms these headers may be installed using the PRIVATE_HEADER option to the install(TARGETS) command.

PROJECT_LABEL: Change the name of a target in an IDE.

Can be used to change the name of the target in an IDE like visual stuido.

PUBLIC_HEADER: Specify public header files in a FRAMEWORK shared library target.

Shared library targets marked with the FRAMEWORK property generate frameworks on OS X and normal shared libraries on other platforms. This property may be set to a list of header files to be placed in the Headers directory inside the framework folder. On non-Apple platforms these headers may be installed using the PUBLIC_HEADER option to the install(TARGETS) command.

RESOURCE: Specify resource files in a FRAMEWORK shared library target.

Shared library targets marked with the FRAMEWORK property generate frameworks on OS X and normal shared libraries on other platforms. This property may be set to a list of files to be placed in the Resources directory inside the framework folder. On non-Apple platforms these files may be installed using the RESOURCE option to the install(TARGETS) command.

RULE_LAUNCH_COMPILE: Specify a launcher for compile rules.

See the global property of the same name for details. This overrides the global and directory property for a target.

RULE_LAUNCH_CUSTOM: Specify a launcher for custom rules.

See the global property of the same name for details. This overrides the global and directory property for a target.

RULE_LAUNCH_LINK: Specify a launcher for link rules.

See the global property of the same name for details. This overrides the global and directory property for a target.

RUNTIME_OUTPUT_DIRECTORY: Output directory in which to build RUNTIME target files.

This property specifies the directory into which runtime target files should be built. There are three kinds of target files that may be built: archive, library, and runtime. Executables are always treated as runtime targets. Static libraries are always treated as archive targets. Module libraries are always treated as library targets. For non-DLL platforms shared libraries are treated as library targets. For DLL platforms the DLL part of a shared library is treated as a runtime target and the corresponding import library is treated as an archive target. All Windows-based systems including Cygwin are DLL platforms. This property is initialized by the value of the variable CMAKE_RUNTIME_OUTPUT_DIRECTORY if it is set when a target is created.

RUNTIME_OUTPUT_NAME: Output name for RUNTIME target files.

This property specifies the base name for runtime target files. It overrides OUTPUT_NAME and OUTPUT_NAME_<CONFIG> properties. There are three kinds of target files that may be built: archive, library, and runtime. Executables are always treated as runtime targets. Static libraries are always treated as archive targets. Module libraries are always treated as library targets. For non-DLL platforms shared libraries are treated as library targets. For DLL platforms the DLL part of a shared library is treated as a runtime target and the corresponding import library is treated as an archive target. All Windows-based systems including Cygwin are DLL platforms.

RUNTIME_OUTPUT_NAME_<CONFIG>: Per-configuration output name for RUNTIME target files.

This is the configuration-specific version of RUNTIME_OUTPUT_NAME.

SKIP_BUILD_RPATH: Should rpaths be used for the build tree.

SKIP_BUILD_RPATH is a boolean specifying whether to skip automatic generation of an rpath allowing the target to run from the build tree. This property is initialized by the value of the variable CMAKE_SKIP_BUILD_RPATH if it is set when a target is created.

SOURCES: Source names specified for a target.

Read-only list of sources specified for a target. The names returned are suitable for passing to the set_source_files_properties command.

SOVERSION: What version number is this target.

For shared libraries VERSION and SOVERSION can be used to specify the build version and api version respectively. When building or installing appropriate symlinks are created if the platform supports symlinks and the linker supports so-names. If only one of both is specified the missing is assumed to have the same version number. For shared libraries and executables on Windows the VERSION attribute is parsed to extract a "major.minor" version number. These numbers are used as the image version of the binary.

STATIC_LIBRARY_FLAGS: Extra flags to use when linking static libraries.

Extra flags to use when linking a static library.

SUFFIX: What comes after the library name.

A target property that can be set to override the suffix (such as ".so") on a library name.

TYPE: The type of the target.

This read-only property can be used to test the type of the given target. It will be one of STATIC_LIBRARY, MODULE_LIBRARY, SHARED_LIBRARY, EXECUTABLE or one of the internal target types.

VERSION: What version number is this target.

For shared libraries VERSION and SOVERSION can be used to specify the build version and api version respectively. When building or installing appropriate symlinks are created if the platform supports symlinks and the linker supports so-names. If only one of both is specified the missing is assumed to have the same version number. For executables VERSION can be used to specify the build version. When building or installing appropriate symlinks are created if the platform supports symlinks. For shared libraries and executables on Windows the VERSION attribute is parsed to extract a "major.minor" version number. These numbers are used as the image version of the binary.

VS_KEYWORD: Visual Studio project keyword.

Can be set to change the visual studio keyword, for example QT integration works better if this is set to Qt4VSv1.0.

VS_SCC_LOCALPATH: Visual Studio Source Code Control Provider.

Can be set to change the visual studio source code control local path property.

VS_SCC_PROJECTNAME: Visual Studio Source Code Control Project.

Can be set to change the visual studio source code control project name property.

VS_SCC_PROVIDER: Visual Studio Source Code Control Provider.

Can be set to change the visual studio source code control provider property.

WIN32_EXECUTABLE: Build an executable with a WinMain entry point on windows.

When this property is set to true the executable when linked on Windows will be created with a WinMain() entry point instead of of just main().This makes it a GUI executable instead of a console application. See the CMAKE_MFC_FLAG variable documentation to configure use of MFC for WinMain executables.

XCODE_ATTRIBUTE_<an-attribute>: Set Xcode target attributes directly.

Tell the Xcode generator to set '<an-attribute>' to a given value in the generated Xcode project. Ignored on other generators.

Properties on Tests

ENVIRONMENT: Specify environment variables that should be defined for running a test.

If set to a list of environment variables and values of the form MYVAR=value those environment variables will be defined while running the test. The environment is restored to its previous state after the test is done.

FAIL_REGULAR_EXPRESSION: If the output matches this regular expression the test will fail.

If set, if the output matches one of specified regular expressions, the test will fail.For example: PASS_REGULAR_EXPRESSION "[^a-z]Error;ERROR;Failed"

LABELS: Specify a list of text labels associated with a test.

The list is reported in dashboard submissions.

MEASUREMENT: Specify a CDASH measurement and value to be reported for a test.

If set to a name then that name will be reported to CDASH as a named measurement with a value of 1. You may also specify a value by setting MEASUREMENT to "measurement=value".

PASS_REGULAR_EXPRESSION: The output must match this regular expression for the test to pass.

If set, the test output will be checked against the specified regular expressions and at least one of the regular expressions has to match, otherwise the test will fail.

TIMEOUT: How many seconds to allow for this test.

This property if set will limit a test to not take more than the specified number of seconds to run. If it exceeds that the test process will be killed and ctest will move to the next test. This setting takes precedence over CTEST_TESTING_TIMEOUT.

WILL_FAIL: If set to true, this will invert the pass/fail flag of the test.

This property can be used for tests that are expected to fail and return a non zero return code.

Properties on Source Files

ABSTRACT: Is this source file an abstract class.

A property on a source file that indicates if the source file represents a class that is abstract. This only makes sense for languages that have a notion of an abstract class and it is only used by some tools that wrap classes into other languages.

COMPILE_DEFINITIONS: Preprocessor definitions for compiling a source file.

The COMPILE_DEFINITIONS property may be set to a semicolon-separated list of preprocessor definitions using the syntax VAR or VAR=value. Function-style definitions are not supported. CMake will automatically escape the value correctly for the native build system (note that CMake language syntax may require escapes to specify some values). This property may be set on a per-configuration basis using the name COMPILE_DEFINITIONS_<CONFIG> where <CONFIG> is an upper-case name (ex. "COMPILE_DEFINITIONS_DEBUG").

CMake will automatically drop some definitions that are not supported by the native build tool. The VS6 IDE does not support definition values with spaces (but NMake does). Xcode does not support per-configuration definitions on source files.

Dislaimer: Most native build tools have poor support for escaping certain values. CMake has work-arounds for many cases but some values may just not be possible to pass correctly. If a value does not seem to be escaped correctly, do not attempt to work-around the problem by adding escape sequences to the value. Your work-around may break in a future version of CMake that has improved escape support. Instead consider defining the macro in a (configured) header file. Then report the limitation.

COMPILE_DEFINITIONS_<CONFIG>: Per-configuration preprocessor definitions on a source file.

This is the configuration-specific version of COMPILE_DEFINITIONS. Note that Xcode does not support per-configuration source file flags so this property will be ignored by the Xcode generator.

COMPILE_FLAGS: Additional flags to be added when compiling this source file.

These flags will be added to the list of compile flags when this source file builds. Use COMPILE_DEFINITIONS to pass additional preprocessor definitions.

EXTERNAL_OBJECT: If set to true then this is an object file.

If this property is set to true then the source file is really an object file and should not be compiled. It will still be linked into the target though.

GENERATED: Is this source file generated as part of the build process.

If a source file is generated by the build process CMake will handle it differently in temrs of dependency checking etc. Otherwise having a non-existent source file could create problems.

HEADER_FILE_ONLY: Is this source file only a header file.

A property on a source file that indicates if the source file is a header file with no associated implementation. This is set automatically based on the file extension and is used by CMake to determine is certain dependency information should be computed.

KEEP_EXTENSION: Make the output file have the same extension as the source file.

If this property is set then the file extension of the output file will be the same as that of the source file. Normally the output file extension is computed based on the language of the source file, for example .cxx will go to a .o extension.

LABELS: Specify a list of text labels associated with a source file.

This property has meaning only when the source file is listed in a target whose LABELS property is also set. No other semantics are currently specified.

LANGUAGE: What programming language is the file.

A property that can be set to indicate what programming language the source file is. If it is not set the language is determined based on the file extension. Typical values are CXX C etc.

LOCATION: The full path to a source file.

A read only property on a SOURCE FILE that contains the full path to the source file.

MACOSX_PACKAGE_LOCATION: Place a source file inside a Mac OS X bundle or framework.

Executable targets with the MACOSX_BUNDLE property set are built as Mac OS X application bundles on Apple platforms. Shared library targets with the FRAMEWORK property set are built as Mac OS X frameworks on Apple platforms. Source files listed in the target with this property set will be copied to a directory inside the bundle or framework content folder specified by the property value. For bundles the content folder is "<name>.app/Contents". For frameworks the content folder is "<name>.framework/Versions/<version>". See the PUBLIC_HEADER, PRIVATE_HEADER, and RESOURCE target properties for specifying files meant for Headers, PrivateHeadres, or Resources directories.

OBJECT_DEPENDS: Additional files on which a compiled object file depends.

Specifies a semicolon-separated list of full-paths to files on which any object files compiled from this source file depend. An object file will be recompiled if any of the named files is newer than it.

This property need not be used to specify the dependency of a source file on a generated header file that it includes. Although the property was originally introduced for this purpose, it is no longer necessary. If the generated header file is created by a custom command in the same target as the source file, the automatic dependency scanning process will recognize the dependency. If the generated header file is created by another target, an inter-target

dependency should be created with the add_dependencies command (if one does not already exist due to linking relationships).

OBJECT_OUTPUTS: Additional outputs for a Makefile rule.

Additional outputs created by compilation of this source file. If any of these outputs is missing the object will be recompiled. This is supported only on Makefile generators and will be ignored on other generators.

SYMBOLIC: Is this just a name for a rule.

If SYMBOLIC (boolean) is set to true the build system will be informed that the source file is not actually created on disk but instead used as a symbolic name for a build rule.

WRAP_EXCLUDE: Exclude this source file from any code wrapping techniques.

Some packages can wrap source files into alternate languages to provide additional functionality. For example, C++ code can be wrapped into Java or Python etc using SWIG etc. If WRAP_EXCLUDE is set to true (1 etc) that indicates then this source file should not be wrapped.

Properties on Cache Entries

ADVANCED: True if entry should be hidden by default in GUIs.

This is a boolean value indicating whether the entry is considered interesting only for advanced configuration. The mark_as_advanced() command modifies this property.

HELPSTRING: Help associated with entry in GUIs.

This string summarizes the purpose of an entry to help users set it through a CMake GUI.

MODIFIED: Internal management property. Do not set or get.

This is an internal cache entry property managed by CMake to track interactive user modification of entries. Ignore it.

STRINGS: Enumerate possible STRING entry values for GUI selection.

For cache entries with type STRING, this enumerates a set of values. CMake GUIs may use this to provide a selection widget instead of a generic string entry field. This is for convenience only. CMake does not enforce that the value matches one of those listed.

TYPE: Widget type for entry in GUIs.

Cache entry values are always strings, but CMake GUIs present widgets to help users set values. The GUIs use this property as a hint to determine the widget type. Valid TYPE values are:

```
BOOL          = Boolean ON/OFF value.
PATH          = Path to a directory.
FILEPATH      = Path to a file.
STRING        = Generic string value.
INTERNAL      = Do not present in GUI at all.
STATIC        = Value managed by CMake, do not change.
UNINITIALIZED = Type not yet specified.
```

Generally the TYPE of a cache entry should be set by the command which creates it (set, option, find_library, etc.).

VALUE: Value of a cache entry.

This property maps to the actual value of a cache entry. Setting this property always sets the value without checking, so use with care.