NAME

cmake-properties - CMake Properties Reference

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.

AUTOGEN_TARGETS_FOLDER

Name of **FOLDER** for ***_automoc** targets that are added automatically by CMake for targets for which **AUTOMOC** is enabled.

If not set, CMake uses the **FOLDER** property of the parent target as a default value for this property. See also the documentation for the **FOLDER** target property and the **AUTOMOC** target property.

AUTOMOC_TARGETS_FOLDER

Name of **FOLDER** for ***_automoc** targets that are added automatically by CMake for targets for which **AUTOMOC** is enabled.

This property is obsolete. Use **AUTOGEN_TARGETS_FOLDER** instead.

If not set, CMake uses the **FOLDER** property of the parent target as a default value for this property. See also the documentation for the **FOLDER** target property and the **AUTOMOC** target property.

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. By 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. By 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.

JOB_POOLS
Ninja only: List of available pools.

A pool is a named integer property and defines the maximum number of concurrent jobs which can be started by a rule assigned to the pool. The **JOB_POOLS** property is a semicolon-separated list of pairs using the syntax NAME=integer (without a space after the equality sign).

For instance:

```
set_property(GLOBAL PROPERTY JOB_POOLS two_jobs=2 ten_jobs=10)
```

Defined pools could be used globally by setting **CMAKE_JOB_POOL_COMPILE** and **CMAKE_JOB_POOL_LINK** or per target by setting the target properties **JOB_POOL_COMPILE** and **JOB_POOL_LINK**.

PREDEFINED_TARGETS_FOLDER
Name of FOLDER for targets that are added automatically by CMake.

If not set, CMake uses CMakePredefinedTargets as a default value for this property. Targets such as INSTALL, PACKAGE and RUN_TESTS will be organized into this FOLDER. See also the documentation for the FOLDER target property.

ECLIPSE_EXTRA_NATURES
List of natures to add to the generated Eclipse project file.

Eclipse projects specify language plugins by using natures. This property should be set to the unique identifier for a nature (which looks like a Java package name).

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.

USE_FOLDERS
Use the FOLDER target property to organize targets into folders.

If not set, CMake treats this property as OFF by default. CMake generators that are capable of organizing into a hierarchy of folders use the values of the FOLDER target property to name those folders. See also the documentation for the FOLDER target property.

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.

CMAKE_CONFIGURE_DEPENDS
Tell CMake about additional input files to the configuration process. If any named file is modified the build system will re-run CMake to re-configure the file and generate the build system again.

Specify files as a semicolon-separated list of paths. Relative paths are interpreted as relative to the current source directory.

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

This is the configuration-specific version of **COMPILE_DEFINITIONS** where **<CONFIG>** is an upper-case name (ex. **COMPILE_DEFINITIONS_DEBUG**).

This property will be initialized in each directory by its value in the directorys parent.

Contents of **COMPILE_DEFINITIONS_<CONFIG>** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

Generator expressions should be preferred instead of setting this property.

COMPILE_DEFINITIONS
Preprocessor definitions for compiling a directorys sources.

This property specifies the list of options given so far to the **add_definitions()** command.

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 will be initialized in each directory by its value in the directorys 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).

Disclaimer: 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. Known limitations include:

```
# - broken almost everywhere
; - broken in VS IDE 7.0 and Borland Makefiles
, - broken in VS IDE
% - broken in some cases in NMake
& | - broken in some cases on MinGW
 < > \" - broken in most Make tools on Windows
```

CMake does not reject these values outright because they do work in some cases. Use with caution.

Contents of **COMPILE_DEFINITIONS** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

The corresponding **COMPILE_DEFINITIONS_<CONFIG>** property may be set to specify per-configuration definitions. Generator expressions should be preferred instead of setting the alternative property.

COMPILE_OPTIONS

List of options to pass to the compiler.

This property specifies the list of options given so far to the **add_compile_options()** command.

This property is used to populate the **COMPILE_OPTIONS** target property, which is used by the generators to set the options for the compiler.

Contents of **COMPILE_OPTIONS** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

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 directorys parent.

INCLUDE_DIRECTORIES
List of preprocessor include file search directories.

This property specifies the list of directories given so far to the **include_directories()** command.

This property is used to populate the **INCLUDE_DIRECTORIES** target property, which is used by the generators to set the include directories for the compiler.

In addition to accepting values from that command, values may be set directly on any directory using the **set_property()** command. A directory gets its initial value from its parent directory if it has one. The intial value of the **INCLUDE_DIRECTORIES** target property comes from the value of this property. Both directory and target property values are adjusted by calls to the **include_directories()** command.

The target property values are used by the generators to set the include paths for the compiler.

Contents of **INCLUDE_DIRECTORIES** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

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.

INTERPROCEDURAL_OPTIMIZATION_<CONFIG>
Per-configuration interprocedural optimization for a directory.

This is a per-configuration version of INTERPROCEDURAL_OPTIMIZATION. If set, this property overrides the generic property for the named configuration.

INTERPROCEDURAL_OPTIMIZATION
Enable interprocedural optimization for targets in a directory.

If set to true, enables interprocedural optimizations if they are known to be supported by the compiler.

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.

VS_GLOBAL_SECTION_POST_<section>
Specify a postSolution global section in Visual Studio.

Setting a property like this generates an entry of the following form in the solution file:

```
GlobalSection(<section>) = postSolution
<contents based on property value>
EndGlobalSection
```

The property must be set to a semicolon-separated list of key=value pairs. Each such pair will be transformed into an entry in the solution global section. Whitespace around key and value is ignored. List elements which do not contain an equal sign are skipped.

This property only works for Visual Studio 7 and above; it is ignored on other generators. The property only applies when set on a directory whose CMakeLists.txt contains a project() command.

Note that CMake generates postSolution sections ExtensibilityGlobals and ExtensibilityAddIns by default. If you set the corresponding property, it will override the default section. For example, setting VS_GLOBAL_SECTION_POST_ExtensibilityGlobals will override the default contents of the ExtensibilityGlobals section, while keeping ExtensibilityAddIns on its default.

VS_GLOBAL_SECTION_PRE_<section>
Specify a preSolution global section in Visual Studio.

Setting a property like this generates an entry of the following form in the solution file:

```
GlobalSection(<section>) = preSolution
<contents based on property value>
EndGlobalSection
```

The property must be set to a semicolon-separated list of key=value pairs. Each such pair will be transformed into an entry in the solution global section. Whitespace around key and value is

ignored. List elements which do not contain an equal sign are skipped.

This property only works for Visual Studio 7 and above; it is ignored on other generators. The property only applies when set on a directory whose CMakeLists.txt contains a project() command.

PROPERTIES ON TARGETS

ALIASED_TARGET

Name of target aliased by this target.

If this is an *Alias Target*, this property contains the name of the target aliased.

ARCHIVE_OUTPUT_DIRECTORY_<CONFIG>

Per-configuration output directory for ARCHIVE target files.

This is a per-configuration version of ARCHIVE_OUTPUT_DIRECTORY, but multi-configuration generators (VS, Xcode) do NOT append a per-configuration subdirectory to the specified directory. This property is initialized by the value of the variable CMAKE_ARCHIVE_OUTPUT_DIRECTORY_<CONFIG> if it is set when a target is created.

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. Multi-configuration generators (VS, Xcode) append a per-configuration subdirectory to the specified directory.

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_<CONFIG>

Per-configuration output name for ARCHIVE target files.

This is the configuration-specific version of ARCHIVE_OUTPUT_NAME.

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.

AUTOGEN_TARGET_DEPENDS

Target dependencies of the corresponding **_automoc** target.

Targets which have their **AUTOMOC** target **ON** have a corresponding **_automoc** target which is used to autogenerate generate moc files. As this **_automoc** target is created at generate-time, it is not possible to define dependencies of it, such as to create inputs for the **moc** executable.

The **AUTOGEN_TARGET_DEPENDS** target property can be set instead to a list of dependencies for the **_automoc** target. The buildsystem will be generated to depend on its contents.

See the **cmake-qt(7)** manual for more information on using CMake with Qt.

AUTOMOC_MOC_OPTIONS

Additional options for moc when using **AUTOMOC**

This property is only used if the **AUTOMOC** property is **ON** for this target. In this case, it holds additional command line options which will be used when **moc** is executed during the build, i.e. it is equivalent to the optional **OPTIONS** argument of the **qt4_wrap_cpp()** macro.

By default it is empty.

See the **cmake-qt(7)** manual for more information on using CMake with Qt.

AUTOMOC

Should the target be processed with automoc (for Qt projects).

AUTOMOC is a boolean specifying whether CMake will handle the Qt **moc** preprocessor automatically, i.e. without having to use the **QT4_WRAP_CPP()** or QT5_WRAP_CPP() macro. Currently Qt4 and Qt5 are supported. When this property is set **ON**, CMake will scan the source files at build time and invoke moc accordingly. If an **#include** statement like **#include moc_foo.cpp** is found, the **Q_OBJECT** class declaration is expected in the header, and **moc** is run on the header file. If an **#include** statement like **#include foo.moc** is found, then a **Q_OBJECT** is expected in the current source file and **moc** is run on the file itself. Additionally, header files with the same base name (like **foo.h**) or **_p** appended to the base name (like **foo_p.h**) are parsed for **Q_OBJECT** macros, and if found, **moc** is also executed on those files. **AUTOMOC** checks multiple header alternative extensions, such as **hpp**, **hxx** etc when searching for headers. The resulting moc files, which are not included as shown above in any of the source files are included in a generated **<targetname>_automoc.cpp** file, which is compiled as part of the target. This property is initialized by the value of the **CMAKE_AUTOMOC** variable if it is set when a target is created.

Additional command line options for moc can be set via the **AUTOMOC_MOC_OPTIONS** property.

By enabling the **CMAKE_AUTOMOC_RELAXED_MODE** variable the rules for searching the files which will be processed by moc can be relaxed. See the documentation for this variable for more details.

The global property **AUTOGEN_TARGETS_FOLDER** can be used to group the automoc targets together in an IDE, e.g. in MSVS.

See the **cmake-qt(7)** manual for more information on using CMake with Qt.

AUTOUIC

Should the target be processed with autouic (for Qt projects).

AUTOUIC is a boolean specifying whether CMake will handle the Qt **uic** code generator automatically, i.e. without having to use the **QT4_WRAP_UI()** or **QT5_WRAP_UI()** macro. Currently Qt4 and Qt5 are supported.

When this property is **ON**, CMake will scan the source files at build time and invoke **uic** accordingly. If an **#include** statement like **#include ui_foo.h** is found in **foo.cpp**, a **foo.ui** file is expected next to **foo.cpp**, and **uic** is run on the **foo.ui** file. This property is initialized by the value of the **CMAKE_AUTOUIC** variable if it is set when a target is created.

Additional command line options for **uic** can be set via the **AUTOUIC_OPTIONS** source file property on the **foo.ui** file. The global property **AUTOGEN_TARGETS_FOLDER** can be used to group the autouic targets together in an IDE, e.g. in MSVS.

See the **cmake-qt(7)** manual for more information on using CMake with Qt.

AUTOUIC_OPTIONS
Additional options for uic when using **AUTOUIC**

This property holds additional command line options which will be used when **uic** is executed during the build via **AUTOUIC**, i.e. it is equivalent to the optional **OPTIONS** argument of the **qt4_wrap_ui()** macro.

By default it is empty.

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

The options set on the target may be overridden by **AUTOUIC_OPTIONS** set on the **.ui** source file.

This property may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions.

See the **cmake-qt(7)** manual for more information on using CMake with Qt.

AUTORCC
Should the target be processed with autorcc (for Qt projects).

AUTORCC is a boolean specifying whether CMake will handle the Qt **rcc** code generator automatically, i.e. without having to use the **QT4_ADD_RESOURCES()** or **QT5_ADD_RESOURCES()** macro. Currently Qt4 and Qt5 are supported.

When this property is **ON**, CMake will handle **.qrc** files added as target sources at build time and invoke **rcc** accordingly. This property is initialized by the value of the **CMAKE_AUTORCC** variable if it is set when a target is created.

Additional command line options for rcc can be set via the **AUTORCC_OPTIONS** source file property on the **.qrc** file.

The global property **AUTOGEN_TARGETS_FOLDER** can be used to group the autorcc targets together in an IDE, e.g. in MSVS.

See the **cmake-qt(7)** manual for more information on using CMake with Qt.

AUTORCC_OPTIONS
Additional options for **rcc** when using **AUTORCC**

This property holds additional command line options which will be used when **rcc** is executed during the build via **AUTORCC**, i.e. it is equivalent to the optional **OPTIONS** argument of the **qt4_add_resources()** macro.

By default it is empty.

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

The options set on the target may be overridden by **AUTORCC_OPTIONS** set on the **.qrc** source file.

See the **cmake-qt(7)** manual for more information on using CMake with Qt.

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.

BUNDLE_EXTENSION
The file extension used to name a BUNDLE target on the Mac.

The default value is bundle - you can also use plugin or whatever file extension is required by the host app for your bundle.

BUNDLE
This target is a CFBundle on the Mac.

If a module library target has this property set to true it will be built as a CFBundle when built on the mac. It will have the directory structure required for a CFBundle and will be suitable to be used for creating Browser Plugins or other application resources.

COMPATIBLE_INTERFACE_BOOL
Properties which must be compatible with their link interface

The **COMPATIBLE_INTERFACE_BOOL** property may contain a list of properties for this target which must be consistent when evaluated as a boolean with the **INTERFACE** variant of the property in all linked dependees. For example, if a property **FOO** appears in the list, then for each dependee, the **INTERFACE_FOO** property content in all of its dependencies must be consistent with each other, and with the **FOO** property in the depender.

Consistency in this sense has the meaning that if the property is set, then it must have the same boolean value as all others, and if the property is not set, then it is ignored.

Note that for each dependee, the set of properties specified in this property must not intersect with the set specified in any of the other *Compatible Interface Properties*.

COMPATIBLE_INTERFACE_NUMBER_MAX
Properties whose maximum value from the link interface will be used.

The **COMPATIBLE_INTERFACE_NUMBER_MAX** property may contain a list of properties for this target whose maximum value may be read at generate time when evaluated in the **INTERFACE** variant of the property in all linked dependees. For example, if a property **FOO** appears in the list, then for each dependee, the **INTERFACE_FOO** property content in all of its dependencies will be compared with each other and with the **FOO** property in the depender. When reading the **FOO** property at generate time, the maximum value will be returned. If the property is not set, then it is ignored.

Note that for each dependee, the set of properties specified in this property must not intersect with the set specified in any of the other *Compatible Interface Properties*.

COMPATIBLE_INTERFACE_NUMBER_MIN
Properties whose maximum value from the link interface will be used.

The **COMPATIBLE_INTERFACE_NUMBER_MIN** property may contain a list of properties for this target whose minimum value may be read at generate time when evaluated in the **INTERFACE** variant of the property of all linked dependees. For example, if a property **FOO** appears in the list, then for each dependee, the **INTERFACE_FOO** property content in all of its dependencies will be compared with each other and with the **FOO** property in the depender. When reading the **FOO** property at generate time, the minimum value will be returned. If the property is not set, then it is ignored.

Note that for each dependee, the set of properties specified in this property must not intersect with the set specified in any of the other *Compatible Interface Properties*.

COMPATIBLE_INTERFACE_STRING
Properties which must be string-compatible with their link interface

The **COMPATIBLE_INTERFACE_STRING** property may contain a list of properties for this target which must be the same when evaluated as a string in the **INTERFACE** variant of the property all linked dependees. For example, if a property **FOO** appears in the list, then for

each dependee, the **INTERFACE_FOO** property content in all of its dependencies must be equal with each other, and with the **FOO** property in the depender. If the property is not set, then it is ignored.

Note that for each dependee, the set of properties specified in this property must not intersect with the set specified in any of the other *Compatible Interface Properties*.

COMPILE_DEFINITIONS_<CONFIG>

Per-configuration preprocessor definitions on a target.

This is the configuration-specific version of **COMPILE_DEFINITIONS** where **<CONFIG>** is an upper-case name (ex. **COMPILE_DEFINITIONS_DEBUG**).

Contents of **COMPILE_DEFINITIONS_<CONFIG>** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

Generator expressions should be preferred instead of setting this property.

COMPILE_DEFINITIONS

Preprocessor definitions for compiling a targets 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).

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).

Disclaimer: 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. Known limitations include:

```
# - broken almost everywhere
; - broken in VS IDE 7.0 and Borland Makefiles
, - broken in VS IDE
% - broken in some cases in NMake
& | - broken in some cases on MinGW
 < > \" - broken in most Make tools on Windows
```

CMake does not reject these values outright because they do work in some cases. Use with caution.

Contents of **COMPILE_DEFINITIONS** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

The corresponding **COMPILE_DEFINITIONS_<CONFIG>** property may be set to specify per-configuration definitions. Generator expressions should be preferred instead of setting the alternative property.

COMPILE_FLAGS

Additional flags to use when compiling this targets sources.

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

This property is deprecated. Use the COMPILE_OPTIONS property or the target_compile_options command instead.

COMPILE_OPTIONS
List of options to pass to the compiler.

This property specifies the list of options specified so far for this property.

This property is intialized by the **COMPILE_OPTIONS** directory property, which is used by the generators to set the options for the compiler.

Contents of **COMPILE_OPTIONS** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

<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.

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 targets 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 or outside to properly setup dllexport/dllimport decorations.

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.

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 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. For non-DLL platforms that require all symbols to be resolved at link time, such as Mac OS X, the module will link to the executable using a flag like -bundle_loader. For other non-DLL platforms the link rule is simply ignored since the dynamic loader will automatically bind symbols when the module is loaded.

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.

EXCLUDE_FROM_DEFAULT_BUILD_<CONFIG>
Per-configuration version of target exclusion from Build Solution.

This is the configuration-specific version of EXCLUDE_FROM_DEFAULT_BUILD. If the generic EXCLUDE_FROM_DEFAULT_BUILD is also set on a target, EXCLUDE_FROM_DEFAULT_BUILD_<CONFIG> takes precedence in configurations for which it has a value.

EXCLUDE_FROM_DEFAULT_BUILD
Exclude target from Build Solution.

This property is only used by Visual Studio generators 7 and above. When set to TRUE, the target will not be built when you press Build Solution.

EXPORT_NAME
Exported name for target files.

This sets the name for the IMPORTED target generated when it this target is is exported. If not set, the logical target name is used by default.

FOLDER
Set the folder name. Use to organize targets in an IDE.

Targets with no FOLDER property will appear as top level entities in IDEs like Visual Studio. Targets with the same FOLDER property value will appear next to each other in a folder of that name. To nest folders, use FOLDER values such as GUI/Dialogs with / characters separating folder levels.

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

This property tells CMake whether the Fortran source files in a target use fixed-format or free-format. CMake will pass the corresponding format flag to the compiler. Use the source-specific Fortran_FORMAT property to change the format of a specific source file. If the variable CMAKE_Fortran_FORMAT is set when a target is created its value is used to initialize this property.

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 targets source directory. If the variable CMAKE_Fortran_MODULE_DIRECTORY is set when a target is created its value is used to initialize this property.

Note that some compilers will automatically search the module output directory for modules USEd during compilation but others will not. If your sources USE modules their location must be specified by INCLUDE_DIRECTORIES regardless of this property.

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

GENERATOR_FILE_NAME
Generators file for this target.

An internal property used by some generators to record the name of the project or dsp file associated with this target. Note that at configure time, this property is only set for targets created by include_external_msproject().

GNUtoMS
Convert GNU import library (.dll.a) to MS format (.lib).

When linking a shared library or executable that exports symbols using GNU tools on Windows (MinGW/MSYS) with Visual Studio installed convert the import library (.dll.a) from GNU to MS format (.lib). Both import libraries will be installed by install(TARGETS) and exported by install(EXPORT) and export() to be linked by applications with either GNU- or MS-compatible tools.

If the variable CMAKE_GNUtoMS is set when a target is created its value is used to initialize this property. The variable must be set prior to the first command that enables a language such as project() or enable_language(). CMake provides the variable as an option to the user automatically when configuring on Windows with GNU tools.

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

This is equivalent to setting the LINKER_LANGUAGE property to CXX. See that propertys 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_CONFIGURATIONS
Configurations provided for an IMPORTED target.

Set this to the list of 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_<CONFIG>
<CONFIG>-specific version of IMPORTED_IMPLIB property.

Configuration names correspond to those provided by the project from which the target is imported.

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

Set this to the location of the .lib part of a windows DLL. Ignored for non-imported targets.

IMPORTED_LINK_DEPENDENT_LIBRARIES_<CONFIG>
<CONFIG>-specific version of IMPORTED_LINK_DEPENDENT_LIBRARIES.

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_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. Set this property to the list of dependent shared libraries of an imported library. The list should be disjoint from the list of interface libraries in the INTERFACE_LINK_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_INTERFACE_LANGUAGES_<CONFIG>
<CONFIG>-specific version of IMPORTED_LINK_INTERFACE_LANGUAGES.

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.

Set this to the list of languages of source 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_LIBRARIES_<CONFIG>
<CONFIG>-specific version of IMPORTED_LINK_INTERFACE_LIBRARIES.

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.

This property is ignored if the target also has a non-empty INTERFACE_LINK_LIBRARIES property.

This property is deprecated. Use INTERFACE_LINK_LIBRARIES instead.

IMPORTED_LINK_INTERFACE_LIBRARIES
Transitive link interface of an IMPORTED target.

Set this to the list of 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.

This property is ignored if the target also has a non-empty INTERFACE_LINK_LIBRARIES property.

This property is deprecated. Use INTERFACE_LINK_LIBRARIES instead.

IMPORTED_LINK_INTERFACE_MULTIPLICITY_<CONFIG>
<CONFIG>-specific version of IMPORTED_LINK_INTERFACE_MULTIPLICITY.

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_LOCATION_<CONFIG>
<CONFIG>-specific version of IMPORTED_LOCATION property.

Configuration names correspond to those provided by the project from which the target is imported.

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

Set this to 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.

Projects may skip IMPORTED_LOCATION if the configuration-specific property IMPORTED_LOCATION_<CONFIG> is set. To get the location of an imported target read one of the LOCATION or LOCATION_<CONFIG> properties.

IMPORTED_NO_SONAME_<CONFIG>
<CONFIG>-specific version of IMPORTED_NO_SONAME property.

Configuration names correspond to those provided by the project from which the target is imported.

IMPORTED_NO_SONAME
Specifies that an IMPORTED shared library target has no soname.

Set this property to true for an imported shared library file that has no soname field. CMake may adjust generated link commands for some platforms to prevent the linker from using the path to the library in place of its missing soname. Ignored for non-imported targets.

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**F alse** for targets built within the project.

IMPORTED_SONAME_<CONFIG>
<CONFIG>-specific version of IMPORTED_SONAME property.

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.

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

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.

INCLUDE_DIRECTORIES
List of preprocessor include file search directories.

This property specifies the list of directories given so far to the **target_include_directories()** command. In addition to accepting values from that command, values may be set directly on any target using the **set_property()** command. A target gets its initial value for this property from the value of the **INCLUDE_DIRECTORIES** directory property. Both directory and target property values are adjusted by calls to the **include_directories()** command.

The value of this property is used by the generators to set the include paths for the compiler.

Relative paths should not be added to this property directly. Use one of the commands above instead to handle relative paths.

Contents of **INCLUDE_DIRECTORIES** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

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.

INTERFACE_AUTOUIC_OPTIONS
List of interface options to pass to uic.

Targets may populate this property to publish the options required to use when invoking **uic**. Consuming targets can add entries to their own **AUTOUIC_OPTIONS** property such as **$<TARGET_PROPERTY:foo,INTERFACE_AUTOUIC_OPTIONS>** to use the uic options specified in the interface of **foo**. This is done automatically by the **target_link_libraries()** command.

This property supports generator expressions. See the **cmake-generator-expressions(7)** manual for available expressions.

INTERFACE_COMPILE_DEFINITIONS
List of public compile definitions for a library.

Targets may populate this property to publish the compile definitions required to compile against the headers for the target. Consuming targets can add entries to their own **COMPILE_DEFINITIONS** property such as **$<TARGET_PROPERTY:foo,INTERFACE_COMPILE_DEFINITIONS>** to use the compile definitions specified in the interface of **foo**.

Contents of **INTERFACE_COMPILE_DEFINITIONS** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

INTERFACE_COMPILE_OPTIONS
List of interface options to pass to the compiler.

Targets may populate this property to publish the compile options required to compile against the headers for the target. Consuming targets can add entries to their own **COMPILE_OPTIONS** property such as **$<TARGET_PROPERTY:foo,INTERFACE_COMPILE_OPTIONS>** to use the compile options specified in the interface of **foo**.

Contents of **INTERFACE_COMPILE_OPTIONS** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

INTERFACE_INCLUDE_DIRECTORIES
List of public include directories for a library.

The **target_include_directories()** command populates this property with values given to the **PUBLIC** and **INTERFACE** keywords. Projects may also get and set the property directly.

Targets may populate this property to publish the include directories required to compile against the headers for the target. Consuming targets can add entries to their own **INCLUDE_DIRECTORIES** property such as **$<TARGET_PROPERTY:foo,INTERFACE_INCLUDE_DIRECTORIES>** to use the include directories specified in the interface of **foo**.

Contents of **INTERFACE_INCLUDE_DIRECTORIES** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

Include directories usage requirements commonly differ between the build-tree and the install-tree. The **BUILD_INTERFACE** and **INSTALL_INTERFACE** generator expressions can be used to describe separate usage requirements based on the usage location. Relative paths are allowed within the **INSTALL_INTERFACE** expression and are interpreted relative to the installation prefix. For example:

```
set_property(TARGET mylib APPEND PROPERTY INTERFACE_INCLUDE_DIRECTORIES
$<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include/mylib>
$<INSTALL_INTERFACE:include/mylib> # <prefix>/include/mylib
)
```

INTERFACE_LINK_LIBRARIES
List public interface libraries for a library.

This property contains the list of transitive link dependencies. 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. This property is overridden by the **LINK_INTERFACE_LIBRARIES** or **LINK_INTERFACE_LIBRARIES_<CONFIG>** property if policy **CMP0022** is **OLD** or unset.

Contents of **INTERFACE_LINK_LIBRARIES** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the

cmake-buildsystem(7) manual for more on defining buildsystem properties.

INTERFACE_POSITION_INDEPENDENT_CODE
Whether consumers need to create a position-independent target

The **INTERFACE_POSITION_INDEPENDENT_CODE** property informs consumers of this target whether they must set their **POSITION_INDEPENDENT_CODE** property to **ON**. If this property is set to **ON**, then the **POSITION_INDEPENDENT_CODE** property on all consumers will be set to **ON**. Similarly, if this property is set to **OFF**, then the **POSITION_INDEPENDENT_CODE** property on all consumers will be set to **OFF**. If this property is undefined, then consumers will determine their **POSITION_INDEPENDENT_CODE** property by other means. Consumers must ensure that the targets that they link to have a consistent requirement for their **INTERFACE_POSITION_INDEPENDENT_CODE** property.

INTERFACE_SYSTEM_INCLUDE_DIRECTORIES
List of public system include directories for a library.

Targets may populate this property to publish the include directories which contain system headers, and therefore should not result in compiler warnings. Consuming targets will then mark the same include directories as system headers.

Contents of **INTERFACE_SYSTEM_INCLUDE_DIRECTORIES** may use generator expressions with the syntax **$<...>**. See the cmake-generator-expressions(7) manual for available expressions. See the cmake-buildsystem(7) manual for more on defining buildsystem properties.

INTERPROCEDURAL_OPTIMIZATION_<CONFIG>
Per-configuration interprocedural optimization for a target.

This is a per-configuration version of INTERPROCEDURAL_OPTIMIZATION. If set, this property overrides the generic property for the named configuration.

INTERPROCEDURAL_OPTIMIZATION
Enable interprocedural optimization for a target.

If set to true, enables interprocedural optimizations if they are known to be supported by the compiler.

JOB_POOL_COMPILE
Ninja only: Pool used for compiling.

The number of parallel compile processes could be limited by defining pools with the global **JOB_POOLS** property and then specifying here the pool name.

For instance:

```
set_property(TARGET myexe PROPERTY JOB_POOL_COMPILE ten_jobs)
```

This property is initialized by the value of **CMAKE_JOB_POOL_COMPILE**.

JOB_POOL_LINK
Ninja only: Pool used for linking.

The number of parallel link processes could be limited by defining pools with the global **JOB_POOLS** property and then specifing here the pool name.

For instance:

```
set_property(TARGET myexe PROPERTY JOB_POOL_LINK two_jobs)
```

This property is initialized by the value of **CMAKE_JOB_POOL_LINK**.

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

Target label semantics are currently unspecified.

<LANG>_VISIBILITY_PRESET
Value for symbol visibility compile flags

The <LANG>_VISIBILITY_PRESET property determines the value passed in a visibility related compile option, such as -fvisibility= for <LANG>. This property only has an affect for libraries and executables with exports. This property is initialized by the value of the variable CMAKE_<LANG>_VISIBILITY_PRESET if it is set when a target is created.

LIBRARY_OUTPUT_DIRECTORY_<CONFIG>
Per-configuration output directory for LIBRARY target files.

This is a per-configuration version of LIBRARY_OUTPUT_DIRECTORY, but multi-configuration generators (VS, Xcode) do NOT append a per-configuration subdirectory to the specified directory. This property is initialized by the value of the variable CMAKE_LIBRARY_OUTPUT_DIRECTORY_<CONFIG> if it is set when a target is created.

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. Multi-configuration generators (VS, Xcode) append a per-configuration subdirectory to the specified directory.

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

This is the configuration-specific version of LIBRARY_OUTPUT_NAME.

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.

LINK_DEPENDS_NO_SHARED
Do not depend on linked shared library files.

Set this property to true to tell CMake generators not to add file-level dependencies on the shared library files linked by this target. Modification to the shared libraries will not be sufficient to relink this target. Logical target-level dependencies will not be affected so the linked shared libraries will still be brought up to date before this target is built.

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

LINK_DEPENDS
Additional files on which a target binary depends for linking.

Specifies a semicolon-separated list of full-paths to files on which the link rule for this target depends. The target binary will be linked if any of the named files is newer than it.

This property is ignored by non-Makefile generators. It is intended to specify dependencies on linker scripts for custom Makefile link rules.

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.

If this property is not set by the user, it will be calculated at generate-time by CMake.

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

This is the configuration-specific version of LINK_FLAGS.

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_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.

This property is overridden by the INTERFACE_LINK_LIBRARIES property if policy CMP0022 is NEW.

This property is deprecated. Use INTERFACE_LINK_LIBRARIES instead.

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 explicit 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 initialized by the value of the variable CMAKE_LINK_INTERFACE_LIBRARIES if it is set when a target is created. This property is ignored for STATIC libraries.

This property is overridden by the INTERFACE_LINK_LIBRARIES property if policy CMP0022 is NEW.

This property is deprecated. Use INTERFACE_LINK_LIBRARIES instead.

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_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_LIBRARIES
List of direct link dependencies.

This property specifies the list of libraries or targets which will be used for linking. In addition to accepting values from the **target_link_libraries()** command, values may be set directly on any target using the **set_property()** command.

The value of this property is used by the generators to set the link libraries for the compiler.

Contents of **LINK_LIBRARIES** may use generator expressions with the syntax **$<...>**. See the **cmake-generator-expressions(7)** manual for available expressions. See the **cmake-buildsystem(7)** manual for more on defining buildsystem properties.

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 CMake adds an option at the end of the library list (if necessary) to set the linker search type back to its starting type. This property switches the final linker search type to -Bstatic regardless of how it started. See also LINK_SEARCH_START_STATIC.

LINK_SEARCH_START_STATIC
Assume the linker looks for static libraries by default.

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 assumed to be -Bdynamic at the beginning of the library list. This property switches the assumption to -Bstatic. It is intended for use when linking an executable statically (e.g. with the GNU -static option). See also LINK_SEARCH_END_STATIC.

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

A read-only property that indicates where a targets 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.

Do not set properties that affect the location of a target after reading this property. These include properties whose names match (RUNTIME|LIBRARY|ARCHIVE)_OUTPUT_(NAME|DIRECTORY)(_<CONFIG>)?, **(IMPLIB_)?(PREFIX|SUFFIX)**, or LINKER_LANGUAGE. Failure to follow this rule is not diagnosed and leaves the location of the target undefined.

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 targets 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. In CMake 2.8.4 and above add_custom_command recognizes generator expressions to refer to target locations anywhere in the command. Therefore this property is not needed for creating custom commands.

Do not set properties that affect the location of a target after reading this property. These include properties whose names match (RUNTIME|LIBRARY|ARCHIVE)_OUTPUT_(NAME|DIRECTORY)(_<CONFIG>)?, **(IMPLIB_)?(PREFIX|SUFFIX)**, or LINKER_LANGUAGE. Failure to follow this rule is not diagnosed and leaves the location of the target undefined.

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

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

A 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.

MACOSX_RPATH
Whether to use rpaths on Mac OS X.

When this property is set to true, the directory portion of the install_name field of shared libraries will be **@rpath** unless overridden by **INSTALL_NAME_DIR**. Runtime paths will also be embedded in binaries using this target and can be controlled by the **INSTALL_RPATH** target property. This property is initialized by the value of the variable **CMAKE_MACOSX_RPATH** if it is set when a target is created.

Policy CMP0042 was introduced to change the default value of MACOSX_RPATH to ON. This is because use of **@rpath** is a more flexible and powerful alternative to **@executable_path** and **@loader_path**.

MAP_IMPORTED_CONFIG_<CONFIG>
Map from project configuration to IMPORTED targets configuration.

Set this to the list of configurations of an imported target that may be used for the current projects <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 this property is set and no matching configurations are available, then the imported target is considered to be not found. This property is ignored for non-imported targets.

This property is initialized by the value of the variable CMAKE_MAP_IMPORTED_CONFIG_<CONFIG> if it is set when a target is created.

NAME
Logical name for the target.

Read-only logical name for the target as used by CMake.

NO_SONAME
Whether to set soname when linking a shared library or module.

Enable this boolean property if a generated shared library or module should not have soname set. Default is to set soname on all shared libraries and modules as long as the platform supports it. Generally, use this property only for leaf private libraries or plugins. If you use it on normal shared libraries which other targets link against, on some platforms a linker will insert a full path to the library (as specified at link time) into the dynamic section of the dependent binary. Therefore, once installed, dynamic loader may eventually fail to locate the library for the binary.

NO_SYSTEM_FROM_IMPORTED
Do not treat includes from IMPORTED target interfaces as SYSTEM.

The contents of the INTERFACE_INCLUDE_DIRECTORIES of IMPORTED targets are treated as SYSTEM includes by default. If this property is enabled, the contents of the INTERFACE_INCLUDE_DIRECTORIES of IMPORTED targets are not treated as system includes. This property is initialized by the value of the variable CMAKE_NO_SYSTEM_FROM_IMPORTED if it is set when a target is created.

OSX_ARCHITECTURES_<CONFIG>
Per-configuration OS X binary architectures for a target.

This property is the configuration-specific version of **OSX_ARCHITECTURES**.

OSX_ARCHITECTURES
Target specific architectures for OS X.

The **OSX_ARCHITECTURES** property sets the target binary architecture for targets on OS X (**-arch**). This property is initialized by the value of the variable **CMAKE_OSX_ARCHITECTURES** if it is set when a target is created. Use **OSX_ARCHITECTURES_<CONFIG>** to set the binary architectures on a per-configuration basis, where **<CONFIG>** is an upper-case name (e.g. **OSX_ARCHITECTURES_DEBUG**).

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

This is the configuration-specific version of OUTPUT_NAME.

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.

PDB_NAME_<CONFIG>
Per-configuration output name for the MS debug symbol **.pdb** file generated by the linker for an executable or shared library target.

This is the configuration-specific version of **PDB_NAME**.

This property is not implemented by the **Visual Studio 6** generator.

PDB_NAME
Output name for the MS debug symbol **.pdb** file generated by the linker for an executable or shared library target.

This property specifies the base name for the debug symbols file. If not set, the logical target name is used by default.

NOTE:
This property does not apply to STATIC library targets because no linker is invoked to produce them so they have no linker-generated **.pdb** file containing debug symbols.

The compiler-generated program database files specified by the MSVC **/Fd** flag are not the same as linker-generated program database files and so are not influenced by this property.

This property is not implemented by the **Visual Studio 6** generator.

PDB_OUTPUT_DIRECTORY_<CONFIG>
Per-configuration output directory for the MS debug symbol **.pdb** file generated by the linker for an executable or shared library target.

This is a per-configuration version of **PDB_OUTPUT_DIRECTORY**, but multi-configuration generators (VS, Xcode) do NOT append a per-configuration subdirectory to the specified directory. This property is initialized by the value of the **CMAKE_PDB_OUTPUT_DIRECTORY_<CONFIG>** variable if it is set when a target is created.

This property is not implemented by the **Visual Studio 6** generator.

PDB_OUTPUT_DIRECTORY
Output directory for the MS debug symbols **.pdb** file generated by the linker for an executable or shared library target.

This property specifies the directory into which the MS debug symbols will be placed by the linker. This property is initialized by the value of the **CMAKE_PDB_OUTPUT_DIRECTORY** variable if it is set when a target is created.

NOTE:
This property does not apply to STATIC library targets because no linker is invoked to produce them so they have no linker-generated **.pdb** file containing debug symbols.

The compiler-generated program database files specified by the MSVC **/Fd** flag are not the

same as linker-generated program database files and so are not influenced by this property.

This property is not implemented by the **Visual Studio 6** generator.

POSITION_INDEPENDENT_CODE
Whether to create a position-independent target

The **POSITION_INDEPENDENT_CODE** property determines whether position independent executables or shared libraries will be created. This property is **True** by default for **SHARED** and **MODULE** library targets and **False** otherwise. This property is initialized by the value of the **CMAKE_POSITION_INDEPENDENT_CODE** variable if it is set when a target is created.

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 Studio.

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_<CONFIG>
Per-configuration output directory for RUNTIME target files.

This is a per-configuration version of RUNTIME_OUTPUT_DIRECTORY, but multi-configuration generators (VS, Xcode) do NOT append a per-configuration subdirectory to the specified directory. This property is initialized by the value of the variable CMAKE_RUNTIME_OUTPUT_DIRECTORY_<CONFIG> if it is set when a target is created.

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. Multi-configuration generators (VS, Xcode) append a per-configuration subdirectory to the specified directory.

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

This is the configuration-specific version of RUNTIME_OUTPUT_NAME.

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.

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. SOVERSION is ignored if NO_SONAME property is set. 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_<CONFIG>
Per-configuration flags for creating a static library.

This is the configuration-specific version of STATIC_LIBRARY_FLAGS.

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 target name.

A target property that can be set to override the suffix (such as .so or .exe) on the name of a library, module or executable.

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.

VISIBILITY_INLINES_HIDDEN
Whether to add a compile flag to hide symbols of inline functions

The VISIBILITY_INLINES_HIDDEN property determines whether a flag for hiding symbols for inline functions, such as -fvisibility-inlines-hidden, should be used when invoking the compiler. This property only has an affect for libraries and executables with exports. This property is initialized by the value of the **CMAKE_VISIBILITY_INLINES_HIDDEN** if it is set when a target is created.

VS_DOTNET_REFERENCES
Visual Studio managed project .NET references

Adds one or more semicolon-delimited .NET references to a generated Visual Studio project. For example, System;System.Windows.Forms.

VS_DOTNET_TARGET_FRAMEWORK_VERSION
Specify the .NET target framework version.

Used to specify the .NET target framework version for C++/CLI. For example, v4.5.

VS_GLOBAL_KEYWORD
Visual Studio project keyword for VS 10 (2010) and newer.

Sets the keyword attribute for a generated Visual Studio project. Defaults to Win32Proj. You may wish to override this value with ManagedCProj, for example, in a Visual Studio managed C++ unit test project.

Use the **VS_KEYWORD** target property to set the keyword for Visual Studio 9 (2008) and older.

VS_GLOBAL_PROJECT_TYPES
Visual Studio project type(s).

Can be set to one or more UUIDs recognized by Visual Studio to indicate the type of project. This value is copied verbatim into the generated project file. Example for a managed C++ unit testing project:

```
{3AC096D0-A1C2-E12C-1390-A8335801FDAB};{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}
```

UUIDs are semicolon-delimited.

VS_GLOBAL_ROOTNAMESPACE
Visual Studio project root namespace.

Sets the RootNamespace attribute for a generated Visual Studio project. The attribute will be generated only if this is set.

VS_GLOBAL_<variable>
Visual Studio project-specific global variable.

Tell the Visual Studio generator to set the global variable <variable> to a given value in the generated Visual Studio project. Ignored on other generators. Qt integration works better if VS_GLOBAL_QtVersion is set to the version FindQt4.cmake found. For example, 4.7.3

VS_KEYWORD
Visual Studio project keyword for VS 9 (2008) and older.

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

Use the **VS_GLOBAL_KEYWORD** target property to set the keyword for Visual Studio 10 (2010) and newer.

VS_SCC_AUXPATH
Visual Studio Source Code Control Aux Path.

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

VS_SCC_LOCALPATH
Visual Studio Source Code Control Local Path.

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.

VS_WINRT_EXTENSIONS
Visual Studio project C++/CX language extensions for Windows Runtime

Can be set to enable C++/CX language extensions.

VS_WINRT_REFERENCES
Visual Studio project Windows Runtime Metadata references

Adds one or more semicolon-delimited WinRT references to a generated Visual Studio project. For example, Windows;Windows.UI.Core.

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 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. This property is initialized by the value of the variable CMAKE_WIN32_EXECUTABLE if it is set when a target is created.

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

ATTACHED_FILES_ON_FAIL
Attach a list of files to a dashboard submission if the test fails.

Same as ATTACHED_FILES, but these files will only be included if the test does not pass.

ATTACHED_FILES
Attach a list of files to a dashboard submission.

Set this property to a list of files that will be encoded and submitted to the dashboard as an addition to the test result.

COST
Set this to a floating point value. Tests in a test set will be run in descending order of cost.

This property describes the cost of a test. You can explicitly set this value; tests with higher COST values will run first.

DEPENDS
Specifies that this test should only be run after the specified list of tests.

Set this to a list of tests that must finish before this test is run.

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: FAIL_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.

PROCESSORS
How many process slots this test requires

Denotes the number of processors that this test will require. This is typically used for MPI tests, and should be used in conjunction with the ctest_test PARALLEL_LEVEL option.

REQUIRED_FILES
List of files required to run the test.

If set to a list of files, the test will not be run unless all of the files exist.

RESOURCE_LOCK
Specify a list of resources that are locked by this test.

If multiple tests specify the same resource lock, they are guaranteed not to run concurrently.

RUN_SERIAL
Do not run this test in parallel with any other test.

Use this option in conjunction with the ctest_test PARALLEL_LEVEL option to specify that this test should not be run in parallel with any other tests.

SKIP_RETURN_CODE
Return code to mark a test as skipped.

Sometimes only a test itself can determine if all requirements for the test are met. If such a situation should not be considered a hard failure a return code of the process can be specified that will mark the test as Not Run if it is encountered.

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.

WORKING_DIRECTORY
The directory from which the test executable will be called.

If this is not set it is called from the directory the test executable is located in.

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.

AUTOUIC_OPTIONS

Additional options for **uic** when using **AUTOUIC**

This property holds additional command line options which will be used when **uic** is executed during the build via **AUTOUIC**, i.e. it is equivalent to the optional **OPTIONS** argument of the **qt4_wrap_ui()<FindQt4>** macro.

By default it is empty.

The options set on the **.ui** source file may override **AUTOUIC_OPTIONS** set on the target.

AUTORCC_OPTIONS

Additional options for **rcc** when using **AUTORCC**

This property holds additional command line options which will be used when **rcc** is executed during the build via **AUTORCC**, i.e. it is equivalent to the optional **OPTIONS** argument of the **qt4_add_resources()** macro.

By default it is empty.

The options set on the **.qrc** source file may override **AUTORCC_OPTIONS** set on the target.

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_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.

Disclaimer: 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. Known limitations include:

```
# - broken almost everywhere
; - broken in VS IDE 7.0 and Borland Makefiles
, - broken in VS IDE
```

```
% - broken in some cases in NMake
& | - broken in some cases on MinGW
 < > \" - broken in most Make tools on Windows
```

CMake does not reject these values outright because they do work in some cases. Use with caution.

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.

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

This property tells CMake whether a given Fortran source file uses fixed-format or free-format. CMake will pass the corresponding format flag to the compiler. Consider using the target-wide Fortran_FORMAT property if all source files in a target share the same format.

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 terms 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 if 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. Setting this property for a file means this file will be compiled. Do not set this for headers or files that should not be compiled.

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, CFBundle, 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. Module library targets with the BUNDLE property set are built as Mac OS X CFBundle bundles 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>. For cfbundles the content folder is <name>.bundle/Contents (unless the extension is changed). See the PUBLIC_HEADER, PRIVATE_HEADER, and RESOURCE target properties for specifying files meant for Headers, PrivateHeaders, 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 that 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.

COPYRIGHT