Professional cmake a practical guide review questions pdf download full

) PRIVATE Items listed after PRIVATE only affect the behavior of targetName itself. CMake adds the supplied paths to the linker command line and leaves the linker to find such libraries on its own. Contrary to what the option name suggests, it applies not just to macOS, but also to other Apple platforms like iOS as well. A library may initially be created without the GLOBAL keyword but later promoted to global visibility, a topic covered in detail in Section 16.3, “Promoting Imported Targets” further below. The INTERFACE keyword tends to be used mostly for imported or interface library targets, or occasionally for adding missing dependencies to a target defined in a part of the project which the developer may not be allowed to change. 82 10.3. General Information . I am also very grateful for the timely words of advice and encouragement delivered at just the right time during some of the more stressful and exhausting periods. Specifying Version Details Versioning is one of those things that frequently doesn’t get the attention it deserves. Flow Control . Note, however, that setting the STRINGS property of a cache variable does not guarantee that the cache variable will hold one of the defined values, it only controls how the variable is presented in the CMake GUI application. 407 28.7. Recommended Practices . 334 26.1. Packaging Basics . If the message is of a general informational nature, prefer to use STATUS rather than no mode keyword at all so that message output does not appear out of order in the build log. This variable is rarely needed, but may prove useful in some debugging scenarios. I’d like to make special mention of Brad King, the CMake project leader, who through his inclusive and encouraging approach to handling new CMake contributors has made people like myself very welcome and feel empowered to get involved. This is the typical usage of the get_property() command. Using this approach, developers do not need to know or specify the individual language features used by the code, they just need to set a single number indicating the standard the code assumes is supported. Testing the platform with the expression APPLE doesn’t express the right condition, the CMake generator should be tested instead: if(CMAKE_GENERATOR STREQUAL "Xcode") # Some Xcode-specific settings here... Nonetheless, script mode allows complex logic to be implemented and it comes with the advantage of not requiring any additional shell interpreter to be installed. If specifying multiple languages, separate each with a space. Note that functions do not introduce a new policy scope (see Section 12.3, “Recommended Practices” for further discussion of this). 7.2. include() The other method CMake provides for pulling in content from other directories is the include() command, which has the following two forms: 54 include(fileName [OPTIONAL] [RESULT_VARIABLE myVar] [NO_POLICY_SCOPE]) include(module [OPTIONAL] [RESULT_VARIABLE myVar] [NO_POLICY_SCOPE]) The first form is somewhat analogous to add_subdirectory(), but there are a number of important differences: • include() expects the name of a file to read in, whereas add_subdirectory() expects a directory and will look for a CMakeLists.txt file within that directory. The third implementation of printme is encountered. On the other, some CMake commands have restrictions which only allow them to operate on things defined within the current file scope, so include() works better for those cases. Directories following a PRIVATE keyword are added to the INCLUDE_DIRECTORIES target property, while directories following an INTERFACE keyword are added to the INTERFACE_INCLUDE_DIRECTORIES target property. There are a few more dependency-related options supported by add_custom_command(). For most of the other scopes, the particular entity within that scope must be named and the requested property attached to that entity will be retrieved. The ubiquitous if() command provides the expected if-then-else behavior and looping is provided through the foreach() and while() commands. are all part of what could be considered the CMake language. An important distinction to note is that if a source file is listed as a MAIN_DEPENDENCY, then the custom command becomes a replacement for how that source file would normally be compiled. Nevertheless, there are situations where imported targets need to be created with global visibility, such as to ensure that the same version or instance of a particular package is used consistently throughout the whole project. A number of modules are also provided to facilitate various aspects of testing and packaging. Developers should think carefully about what minimum CMake version their project should require. The range indicates that the CMake version in use must be at least the minimum and the behavior to use should be the lesser of the specified maximum and the running CMake version. ◦ The custom command is NOT executed, since onTheFly.cpp already exists. 10 3.2. The project() Command . An alternative is to base the compiler and linker flags on one of the other build types and add the extra flags needed. 352 26.4.4. NSIS . For these generators, the build type has to be chosen by setting the CMAKE_BUILD_TYPE cache variable. Similarly, there are directory properties which can be set in the child directory’s CMakeLists.txt file which will have no effect on the parent’s directory properties. There is, however, a specific case to be aware of which can result in unexpected behavior. The INACTIVITY_TIMEOUT option provides this capability, whereas TIMEOUT only allows the total time to be limited. This can be a useful way of finding files in a well-structured directory hierarchy. If a requested feature is not supported by the compiler, CMake will report an error. A variable cannot be read or modified outside of its scope. When implementing custom tasks, try to avoid those features which do not have universal support across all platforms. 203 21. For these reasons, developers are discouraged from using in-source builds where possible, even for simple projects. Again, CMake finds an existing command by that name, so it redefines the name _printme to point to the old command (which is the second implementation) and sets printme to point to the new definition. Whereas both configure_file() and file() are intended for use at configure time or possibly as part of a CMake script at install time, CMake’s command mode can be used for copying files and directories at build time. For this reason, the CMake developers are careful to ensure that changes are implemented in such a way as to preserve backward compatibility and to provide a straightforward, controlled migration path for projects to be updated to the new behavior. For subsequent CMake runs, the CC and CXX environment variables are not substituted in this way, which can lead to unexpected results if the commands perform actions that rely on CC and/or CXX having the same values every time execute_process() is called. It needs to know how to set up the policy behavior for all subsequent processing. In particular, if a project defines a chain of library dependencies with a mix of both old and new forms of the command, the old-style form will generally be treated as PRIVATE. TARGET_FILE_NAME Same as TARGET_FILE but without the path (i.e. it provides just the file name part). When CMake runs its configure step, it still treats the cache variable as an arbitrary string, so it is still possible to give the cache variable any value either at the cmake command line or via set() commands in the project. All other CMake generators will treat this as PRE_LINK instead. 225 22.4. Loadable Bundles . If the project is being scripted, invoke the build tool via cmake --build instead of invoking the build tool directly. While such manipulations range from trivial through to quite complex, the more common tasks include: • Constructing paths or extracting components of a path. While the CMake documentation doesn’t define the supported regular expression syntax for if() commands, it does define it elsewhere for other commands (e.g. see the string() command documentation). This too only affects the way the variable is displayed in cmake-gui, it does not in any way affect how CMake uses the variable during processing. The SOURCES option allows arbitrary files to be listed which will then be associated with the custom target. Neither the nominal type nor the documentation string affect how CMake treats the variable, they are only used by GUI tools to present the variable to the user in a more suitable form. Variables defined or modified inside a function have no effect on variables of the same name outside of the function. 390 28.1. Superbuild Structure . 70 9.1. General Property Commands . It also includes a number of modules which provide support for languages in a more extensible and configurable way. CMake supports such tasks through custom commands and custom targets. CMake provides the math() command for performing basic mathematical evaluation: math(EXPR outVar mathExpr) The first argument must by the keyword EXPR, while mathExpr defines the expression to be evaluated and the result will be stored in outVar. 60 8.3. Keyword Arguments . Hovering the mouse over any of the cache variables will show a tooltip containing the docstring for that variable. Previous Value(s) New Value(s) No Keyword APPEND APPEND_STRING foo bar bar foo;bar foobar a;b c;d c;d a;b;c;d a;b;c;d a;bc;d The get_property() command follows a similar form: get_property(resultVar entitySpecific PROPERTY propName [DEFINED | SET | BRIEF_DOCS | FULL_DOCS]) The PROPERTY keyword is always required and is always followed by the name of the property to retrieve. To move a file into a directory, the file name must be specified as part of the destination. This works regardless of the platform or CMake generator being used, thereby freeing the project from having to work out the various platform and generator differences that lead to a range of different output directory structures, file names, etc. Even experienced users should find at least the Recommended Practices section at the end of each chapter to be a useful read. 3 Chapter 1. The syntax for these operations all share the same form: 34 string(LENGTH string(TOLOWER string(TOUPPER string(STRIP input input input input outVar) outVar) outVar) outVar) CMake provides other operations, such as string comparison, hashing, timestamps and more, but their use is less common in everyday CMake projects. • The name of a CMake library target. If CMAKE_WARN_DEPRECATED is defined to a boolean true, the message will be treated as a warning. SHOW_PROGRESS When present, this option causes progress information to be logged as status messages. 8.4. Scope A fundamental difference between functions and macros is that functions introduce a new variable scope, whereas macros do not. Generator expressions are also a perfect fit for handling content which changes depending on the target or the build type. For some multi configuration generator types, CMake will check that the required variables exist and will fail with an error if they are not set. Matches can be referred to in replaceWith using the usual notation \1, \2, etc., but note that the backslashes themselves must be escaped in CMake. These warnings are the most common way developers are exposed to CMake’s policy functionality. Projects may choose to use this more specific form where it is convenient and use the generic form elsewhere, or for consistency the more generic form may be used everywhere. remove_directory dir make_directory dirs... Because they are object files, they cannot be linked to as a single unit (although CMake 3.12 relaxes this restriction). endforeach() In this more general form, individual arguments can still be specified using the ITEMS keyword, but the LISTS keyword allows one or more list variables to be specified. In earlier versions of CMake, target properties were much less prominent and properties were often specified at the directory level instead. 186 20.3. Shared Library Versioning . The other type of property CMake supports is for installed files. By convention, the postfix for Release builds is typically empty. It can also be useful if interactive input is required for non-IDE builds. Alternatively, cmake can invoke the build tool on the developer’s behalf like so: 8 cmake --build /some/path/build --config Debug --target MyApp This works even for project types the developer may be more accustomed to using through an IDE like Xcode or Visual Studio. Multiple calls to add_custom_command() can be made to append multiple sets of custom commands to a particular target. To my friend and designer, V, the way you somehow managed to make sense of my random, disconnected ideas and conflicting snippets still baffles me. Variables can be referenced in the mathExpr with the usual ${myVar} notation. On the other hand, if the project will itself be a dependency for other projects, then choosing a more recent CMake version may present 11 a hurdle for adoption. The include() command does not introduce a new directory scope and neither does a find_package() call, so imported targets defined by files brought into the build that way can be promoted. 356 26.4.7. RPM . To avoid unexpected behavior, also avoid calling return() from inside a macro. Normally, each time target_include_directories() is called, the specified directories are appended to the relevant target properties. They do, however, have related configuration-specific properties: • LINK_FLAGS_ • STATIC_LIBRARY_FLAGS_ These flags will be used in addition to the non-configuration-specific flags when the matches the configuration being built. The COMMENT and WORKING_DIRECTORY options can also be specified for each call and they will take effect for the commands being added in that call. It is possible, however, to send different streams to different places, such as the output stream to a variable and the error stream to a file or vice versa. file(WRITE multi.txt [[First line Second line]]) file(WRITE userCheck.sh [=[#!/bin/bash [[-n "${USER}"]] && echo "Have USER"]=]) In the above, the contents to be written to multi.txt consist only of simple text with no special characters, so the simplest bracket syntax where = characters can be omitted is sufficient, leaving just a pair of square brackets to mark the start and end of the content. The CMake GUI and the ccmake tool both provide an option to show or hide advanced cache variables, allowing the user to 76 choose whether to focus on just the main basic variables or to see the full set of non-internal variables. 404 28.5.3. Windows Specific Issues . 180 19.4. Recommended Practices . Therefore, in the first run, a normal variable would be hidden, but in subsequent runs it would not. Before rushing to start using source properties, developers should be aware of an implementation detail which may present a strong deterrent to their use in some situations. The above description also happens to map very closely to the behavior of the other target_...() commands which manipulate compiler flags. When running cmake from the command line, both stages are executed automatically, but in the GUI application, they are triggered separately with the Configure and Generate buttons. PUBLIC This is equivalent to combining the effects of PRIVATE and INTERFACE. If multiple values are given, the values will be joined together with a semicolon separating each value - the resultant string is how CMake represents lists. This can be especially useful as a step before performing a string comparison. Projects should keep in mind that source properties are only visible to targets defined in the same directory scope. 57 8. 67 8.6. Recommended Practices . In fact, this is the main use case for which the ability to promote imported targets was created. All of the techniques discussed above merely allow a custom build type to be selected, they don’t define anything about that build type. The cmake_minimum_required() command implicitly calls this form to set CMake’s behavior. cmake_parse_arguments() takes the arguments supplied as the argsToParse parameter and processes them according to the specified sets of keywords. Another type of pseudo-library is also supported for collecting together object files without going as far as creating a static library. As discussed back in Section 9.6, “Cache Variable Properties”, however, cache variables can have their STRINGS property defined to hold a set of valid values. 68 9. Properties Properties affect just about all aspects of the build process, from how a source file is compiled into an object file, right through to the install location of built binaries in a packaged installer. 36 The above describes most of the available list() subcommands. If a macro argument is used in an if() statement, it would be treated as a string rather than a variable. That target can be an ordinary executable, a library target or it can even be a custom target. Every feature will have a macro of the form prefix_COMPILER_UPPERCASEFEATURE whose value will be 1 or 0 depending on whether the feature is supported or not for the compiler being used. Furthermore, the environment at the time CMake is run can change compared to when the build 37 itself is invoked. Pay special attention to dependencies for the inputs and outputs of custom tasks. Some of the more problematic anti-patterns are discussed in this chapter. COMPILE_DEFINITIONS This holds a list of definitions to be set on the compile command line. Unfortunately, it is very common to see tutorials and examples use file(GLOB) and file(GLOB_RECURSE) to collect the set of sources to pass to commands like add_executable() and add_library(). This particular topic is covered in more detail in Section 13.2, “Common Errors”. The set of properties supporting generator expressions have expanded over time, with some CMake releases also expanding the set of supported expressions. This file is a platform independent description of the whole project, which CMake then turns into platform specific build tool project files. Different parts of the build can be turned on or off, paths to external packages can be set, flags for compilers and linkers can be modified and so on. Even then, such logic is likely to be fragile and could probably be expressed with more generality and robustness using generator expressions instead. As a special case, cache variables of type STRING can be given a set of values to show in a combobox in CMake GUI instead of showing a simple text entry widget. A work like this doesn’t happen without the generosity, patience and insight of others, nor does it succeed without being challenged, tested and reworked. While the introduction of new capabilities is unlikely to cause problems for existing projects built with CMake, any change in behavior has the potential to break projects if they are relying on the old behavior. Developers can then turn that option on/off or set the variable to non-default values without having to edit the CMakeLists.txt file directly. Non-cache variables of the same name can be defined and they will override the cache variables for the current directory scope and its children (i.e. those created by add_subdirectory()). The more direct method of employing modules uses the include() command to essentially inject the module code into the current scope. A few examples help demonstrate typical usage. These allow any command or set of commands to be executed at build time to perform whatever arbitrary tasks a project requires. Unless the project specifically requires a particular type of library, the preferred practice is to not specify it and leave the choice up to the developer when building the project. But there are situations where understanding the separation into two distinct phases is particularly important. 376 27.1.4. Common Issues . It must be one of the following: GLOBAL DIRECTORY TARGET SOURCE INSTALL TEST CACHE [dir] [target1 [source1 [file1 [test1 [var1 [target2 [source2 [file2 [test2 [var2 [...]]] [...]]] [...]]] [...]]] [...]]] The first word of each of the above defines the type of entity whose property is being set. These variables are: CMAKE_PROJECT_VERSION CMAKE_PROJECT_VERSION_MAJOR CMAKE_PROJECT_VERSION_MINOR CMAKE_PROJECT_VERSION_PATCH CMAKE_PROJECT_VERSION_TWEAK This same pattern is also followed to provide variables for the project name, description and homepage url, the latter two being added in CMake versions 3.9 and 3.12 respectively. 392 28.3. Common Top Level Subdirectories . It aims to give performance close to a Release build, but still allow some level of debugging. That said, for better robustness, it is still recommended to use at least CMake 3.11 if custom build types are going to be defined. With one exception discussed below, adding the GLOBAL keyword to yield the keyword combination INTERFACE IMPORTED GLOBAL results in a library with little practical difference compared to INTERFACE alone. Success is measured by the exit code of the executable created from the source provided, with 0 being treated as success and all other values indicating failure. cmake_pop_check_state() is the opposite, it discards the current values of the CMAKE_REQUIRED_... variables and restores them to the previous stack level’s values. The project will also define move constructors if the compiler supports rvalue references. Chapter 14, Compiler And Linker Essentials covers those commands and how they relate to target properties in depth. The above could be more accurately implemented as follows: if(MSVC) set(platformImpl source_msvc.cpp) else() set(platformImpl source_generic.cpp) endif() Another example involves conditional behavior based on the CMake generator being used. 363 26.4.11. If the CMakeLists.txt file does not call project() or does not call it early enough, CMake will implicitly call it internally for the default languages C and CXX to ensure compilers and linkers are properly set up for other commands which rely on them. As a general guide, most simple projects are probably better off preferring to use add_subdirectory() over include(). The CheckFunctionExists module is already documented as being deprecated and the CheckVariableExists module offers nothing that CheckSymbolExists doesn’t already provide. 94 12. It also allows a single configuration generator to effectively act like a multi configuration generator, with IDE environments like Qt Creator supporting switching between build directories just as easily as Xcode or Visual Studio allow switching between build schemes or configurations. When scripting builds, a common deficiency is to assume a particular generator is used or to not properly account for differences between single and multi configuration generators. It is relatively common to see platform specific commands used with add_custom_command(), add_custom_target() and execute_process(). As a general rule of thumb, choose the most recent CMake version that won’t present significant problems for those building the project. This is because imported targets are assumed to be coming from outside the project and therefore their associated headers should be treated in a similar way to other system-provided headers. Acknowledgments It is only when you come to thank all those who have contributed to the process of getting your book released that you realize just how many people have been involved. Before CMake 3.8, equivalent logic would have to be expressed in the following more verbose way that requires the expression to be given twice: $$ Generator expressions can be nested, allowing expressions of arbitrary complexity to be constructed. For these reasons, adding linker search directories should be avoided where possible. 165 The contents for userCheck.sh are much more interesting and highlight the features of bracket syntax. wildcards, but care needs to be taken to avoid deleting more than was intended and leaving the cache in an unbuildable state. They are always attached to a specific entity, whether that be a directory, target, source file, test case, cache variable or even the overall build process itself. 4.1. Executables The more complete form of the basic add_executable() command is as follows: add_executable(targetName [WIN32] [MACOSX_BUNDLE] [EXCLUDE_FROM_ALL] source1 [source2 ...]) The only differences to the form shown previously are the new optional keywords. It is possible to have a normal variable and a cache variable with the same name but holding different values. In addition, a file can be specified for the input stream to the first command with the INPUT_FILE option. $ Identifies the platform for which the target is being built. If the CMAKE_ERROR_DEPRECATED variable is defined to a boolean true value, the message will be treated as an error. Because CMake handles all the various platform differences and paths on the project’s behalf, using a CMake target name is generally the preferred method. This is a direct consequence of how the COMPILE_DEFINITIONS directory property is used by CMake. This flag prefix is stripped off by CMake before the define is stored in the COMPILE_DEFINITIONS directory property, so it doesn’t matter which prefix is used, regardless of the compiler or platform on which the project is built. Section 16.2, “Libraries” discusses additional behaviors for static libraries which may result in further linking to satisfy dependency relationships, including cyclic dependencies. This includes things like verifying that the compilers work, determining the set of supported compiler features and various other tasks. Relative paths will be automatically converted to absolute paths where needed, with paths being treated as relative to the current source directory. In the following example, foo would be built with C++14, bar with C++17 and guff with C++14: set_target_properties(foo PROPERTIES CXX_STANDARD 11) target_compile_features(foo PUBLIC cxx_std_14) set_target_properties(bar PROPERTIES CXX_STANDARD 17) target_compile_features(bar PRIVATE cxx_std_11) set_target_properties(guff PROPERTIES CXX_STANDARD 11) target_link_libraries(guff PRIVATE foo) Note that this may mean a more recent language standard could be used than what the project expected, which in some cases can result in compilation errors. They are also a cornerstone of CMake’s own module system (covered separately in Chapter 11, Modules). 210 21.5. Compiler Checks . When working with projects that support CMake versions earlier than 3.0, it is sometimes the case that they will define their own version-related variables which clash with those automatically defined by CMake 3.0 and later. If the code cannot be built, this is also treated as a failure. 77 9.8. Recommended Practices . • Since each of the supported keywords has to be passed to cmake_parse_arguments() and it is typically called near the top of the function, it is generally very clear what arguments the function supports. The goal is to incrementally build up the base knowledge needed to use CMake effectively, with an emphasis on being able to put that knowledge into practice right away. The copy is performed immediately, so it is a configure-time operation. bar[0-9].txt Matches all files of the form barX.txt where X is a single digit. This is a common pattern used by CMake to tailor command behavior or to modify defaults so they don’t have to be repeated for every command, target definition, etc. • Both commands introduce a new policy scope by default, but the include() command can be told not to do so with the NO_POLICY_SCOPE option (add_subdirectory() has no such option). When understood this way, VARIABLE is consistent with how the other entity types are handled. 96 12.1. Policy Control . Generator Expressions When running CMake, developers tend to think of it as a single step which involves reading the project’s CMakeLists.txt file and producing the relevant set of generator-specific project files (e.g. Visual Studio solution and project files, an Xcode project, Unix Makefiles or Ninja input files). By setting CMAKE_MODULE_PATH appropriate in each project, those reusable CMake building blocks are then made available for use as widely as needed. }; In addition to the override keyword, a number of other features can also have a similar conditionally defined symbol used in much the same way. Because AND, OR and NOT require that their expressions evaluate to only 0 or 1, consider wrapping those expressions in a $ to force more tolerant logic of what is considered a true or false expression. In this section, a number of techniques for copying files are presented. For all generator types, CMake will carry out a series of tests and interrogate the system to determine how to set up the project files. The first form allows for the extraction of the different parts of a path or file name: get_filename_component(outVar input component [CACHE]) The result of the call is stored in the variable named by outVar. 14.1.1. Compiler Flags The most fundamental target properties for controlling compiler flags are the following, each of which hold a list of items: INCLUDE_DIRECTORIES This is a list of directories to be used as header search paths, all of which must be absolute paths. One of the effects of calling add_subdirectory() is that CMake creates a new scope for processing that directory’s CMakeLists.txt file. 355 26.4.6. productbuild . Some of the more frequently used functionality is presented below, but the CMake reference documentation should be considered the canonical source of all available operations and their behavior. Since no CMakeCache.txt file is updated in script mode, -D options can be used freely without affecting the main build’s cache. Any directory can be added, with sourceDir being specified as either an absolute or relative path, the latter being relative to the current source directory. If neither variable is defined, the message will not be shown. Building Simple Targets . If the timestamps should not be updated if the files already match, then the copy_if_different command may be more appropriate: cmake -E copy_if_different file1 [file2...] destination This functions exactly like the copy command except if a source file already exists at the destination and is the same as the source, no copy is performed and the timestamp of the target is left alone. Generator expressions are supported in all cases and there would usually be no need to handle build and install situations differently. These commands not only manipulate the properties on specific targets, they also set up dependency relationships between targets so that CMake can propagate some properties automatically. If iterating over items, consider whether using the IN LISTS or IN ITEMS forms communicate more clearly what is being done rather than a bare foreach(loopVar item1 item2 ...) form. The ccmake tool offers all the same functionality as the cmake-gui application, but it does so through a text-based interface Rather than selecting the source and build directories like with cmake-gui, the source or build directory has to be specified on the ccmake command line, just like for the cmake command. file(DOWNLOAD url fileName [options...]) file(UPLOAD fileName url [options...]) The DOWNLOAD form downloads a file from the specified url and saves it to fileName. This includes when a TIMEOUT option is being used, since TIMEOUT on its own will not generate an error, it will only ensure the command doesn’t run longer than the nominated timeout period. They can, however, still be used as inputs for the build phase, such as generated sources or headers. 384 27.2.2. Other Uses For FetchContent . This is called the configure step. If the command runs longer than the timeout threshold, the result variable will hold an error string indicating that the command was terminated due to timeout, which is why printing the result variable is recommended. These other modules are either from earlier versions of CMake or are for a language other than C or C++. In such cases, it may be beneficial to instead require the oldest CMake version that still provides the minimum CMake features needed, but make use of features from later CMake versions if available (Chapter 12, Policies presents techniques for achieving this). If any of the listed files do not exist, the file() command does not report an error. The discussions in Chapter 7, Using Subdirectories and Chapter 8, Functions And Macros further clarify this behavior, explaining how a variable’s scope can influence what value ${myVar} would return. For reference, the above is equivalent to: set(optVar initialValue CACHE BOOL helpString) 25 Compared to set(), the option() command more clearly expresses the behavior for boolean cache variables, so it would generally be the preferred command to use. It would be unusual, however, for a project module to change its behavior based on a policy setting. Consider the following example: macro(inner) message("From inner") return() # Usually dangerous within a macro message("Never printed") endmacro() function(outer) message("From outer before calling inner") inner() message("Also never printed") endfunction() outer() The output from the above would be: From outer before calling inner From inner To highlight why the second message in the function body is never printed, paste the contents of the macro body into where it is called: 66 function(outer) message("From outer before calling inner") # === Pasted macro body === message("From inner") return() message("Never printed") # === End of macro body === message("Also never printed") endfunction() outer() It is now much clearer why the return() statement causes processing to leave the function, even though it was originally called from inside the macro. The contents of this file are covered in great detail in subsequent chapters, but for now, it is enough to know that this is what controls everything that CMake will do in setting up and performing the build. Earlier in this chapter, it was noted that the COMMENT option for add_custom_target() and add_custom_command() isn’t always reliable. Problems with the compiler and linker setup are then caught very early. Using a .cmake file name suffix allows the include() command to find the files as modules, a topic covered in detail in Chapter 11, Modules. 75 9.5. Source Properties . The following partial example demonstrates one of the motivating cases for this option: file(DOWNLOAD " myfile.tar.gz EXPECTED_HASH SHA1=${myfileHash} HTTPHEADER "Host: somebucket.s3.amazonaws.com" HTTPHEADER "Date: ${timestamp}" HTTPHEADER "Content-Type: application/x-compressed-tar" HTTPHEADER "Authorization: AWS ${s3key}:${signature}") The file()-based download and upload commands tend to find use more as part of install steps, packaging or test reporting, but they can also occasionally find use for other purposes. It can be very hard to overcome the inertia of existing processes and change how version numbers are handled later in the life of a project. Good practice is to re-run the configure stage until there are no changes, since this ensures robust behavior for more complex projects where enabling some options may add further options which could require another configure pass. 15.3. Recommended Practices Projects should avoid setting compiler and linker flags directly to control the language standard used. ⑤ myVar is modified in the child scope. When the myApp sources are compiled, they will have /some/path/include as a header search path and will also have a compiler definition COOL_FEATURE=1 provided on the compiler command line. Some network connections may be of poor quality or may simply be very slow. Also keep in mind that it is possible for one source file to be compiled into multiple targets, so any source properties that are set should make sense for all targets the source is added to. CMake also supports executing tasks at configure time, enabling various techniques that rely on tasks being completed before the build stage or even before processing later parts of the current CMakeLists.txt file. target_compile_options(targetName [BEFORE] item1 [item2 ...] [item3 [item4 ...]] ... 38 Chapter 6. When working with CMake, it helps to understand its view of the world. When developers move to a more recent CMake release, the newer CMake version will sometimes issue such warnings to highlight how the project should be updated to use a new behavior. By default, imported libraries are defined as local targets, meaning they are only visible in the current directory scope and below. Custom Tasks . If the project can require CMake 3.0 or later, it is also good to force thinking about project version numbers early and start incorporating version numbering into the project() command as soon as possible. The command line option only needs to be given once, since it is stored in the cache for subsequent runs and therefore does not need to be provided every time cmake is run. It’s only effect is to end processing of the current scope. For either form of the get_directory_property() command, if the DIRECTORY argument is used, the named directory must have already been processed by CMake. No other mechanism in CMake offers the same degree of flexibility and generality for handling the multitude of factors which may contribute to the final content needed for a particular target property. Toolchains And Cross Compiling......Page 21421.1. Toolchain Files......Page 21521.2. Defining The Target System......Page 21621.3. Tool Selection......Page 21721.4. System Roots......Page 21921.5. Compiler Checks......Page 22021.6.2. GCC With 32-bit Target On 64-bit Host......Page 22121.6.3. Android......Page 22221.7. Recommended Practices......Page 22722.1. CMake Generator Selection......Page 22822.2. Application Bundles......Page 22922.3. Frameworks......Page 23422.4. Loadable Bundles......Page 23722.5. Build Settings......Page 23822.6. Code Signing......Page 24022.7. Creating And Exporting Archives......Page 24222.8. Limitations......Page 24422.9. Recommended Practices......Page 245Part III: The Bigger Picture......Page 24823.1. Finding Files And Paths......Page 24923.1.2. Cross-compilation Controls......Page 25323.3. Finding Programs......Page 25523.4. Finding Libraries......Page 25623.5. Finding Packages......Page 25823.5.1. Package Registries......Page 26423.5.2. FindPkgConfig......Page 26623.6. Recommended Practices......Page 26824.1. Defining And Executing A Simple Test......Page 27224.2. Pass / Fail Criteria And Other Result Types......Page 27524.3. Test Grouping And Selection......Page 27824.4. Parallel Execution......Page 28224.5. Test Dependencies......Page 28424.7. Build And Test Mode......Page 28724.8.1. Key CDash Concepts......Page 29024.8.2. Executing Pipelines And Actions......Page 29124.8.3. CTest Configuration......Page 29324.8.4. Test Measurements And Results......Page 29724.9. GoogleTest......Page 29824.10. Command mode is the preferred way to copy content as part of add_custom_target() and add_custom_command() rules, since it provides platform independence (see Section 17.5, “Platform Independent Commands”). 7.1.1. Source And Binary Directory Variables Sometimes a developer needs to know the location of the build directory corresponding to the current source directory, such as when copying files needed at run time or to perform a custom build task. The first matching file will be used, or if no matching file is found or if CMAKE_MODULE_PATH is empty or undefined, CMake will then search in its own internal module directory. This is generally achieved using the message() command. This undocumented behavior has existed since early versions of CMake, even as far back as the now deprecated exec_program() command which execute_process() replaced. When PARENT_SCOPE is used, the variable being set is the one in the parent scope, not the one in the current scope. Just like executables, libraries may also be defined as imported targets. else() # Things for other CMake generators here... 8.3. Keyword Arguments The previous section illustrated how the ARG... variables can be used to handle a variable set of arguments. 143 The directory in which the commands are executed is the current binary directory by default. Modules . 323 25.7.2. Config Files For Non-CMake Projects . A simplified version of that function showing only the compulsory options can be described as follows. New features for tests (e.g. environment modification) More detailed release notes are available. CMake captures this richer set of dependency relationships with its target_link_libraries() command, not just the simplistic idea of needing to link. Therefore, setting environment variables within the CMakeLists.txt file like this is rarely useful. 278 24.7. Build And Test Mode . Generator expressions can be used, but there would usually be no need to handle build and install situations differently. It also goes without saying that the many contributors to CMake over the years also deserve much praise for their efforts, often made on a purely voluntary basis. In the case of policies, each add_subdirectory() call creates a new scope in which policy changes can be made without affecting the policy settings of the parent. In addition to the TARGET_FILE expressions, CMake also provides some library-specific expressions which have similar roles except they handle file name prefix and/or suffix details slightly differently. 117 add_compile_options(opt1 [opt2 ...]) The add_compile_options() command is used to provide arbitrary compiler options. There are other modules which aim to provide functionality that is similar to or a subset of that covered by CheckSymbolExists. An example of where this is useful is when working with library targets defined using the INTERFACE form of add_library(), such as when using a target to represent a header-only library’s dependencies (see Chapter 16, Target Types). The exact effects of this option vary somewhat between platforms. There are three commands related to copying, the first of which is used to copy individual files: cmake -E copy file1 [file2...] destination If only one source file is provided, then destination is interpreted as the name of the file to copy to, unless it names an existing directory. Enabling the Advanced option shows all cache variables except those marked INTERNAL (the only way to see INTERNAL variables is to edit the CMakeCache.txt file with a text editor, since they are not intended to be manipulated directly by developers). A number of different approaches can be used, each with their own strengths and weaknesses. Parentheses can be used to capture parts of the matched value. Note, however, that a bug existed prior to CMake 3.10 which resulted in the RESET option being ignored, so for projects that need to work with versions before 3.10, it is better to use a separate call to cmake_reset_check_state() instead. These more direct expressions take care of extracting out parts of some properties or computing values based on raw properties. 314 25.4. Installing Files And Directories . Ultimately, lists are just a single string with list items separated by semicolons, which can make it less convenient to manipulate individual list items. 273 24.5. Test Dependencies . Note, however, that CMake does not typically raise an error if either operand is not a number and its behavior does not fully conform to the official documentation when values contain more than just digits. To have a newline written, it must be included in the content passed to the file() command. 19.1. Project Version A project version often needs to be defined near the beginning of the top level CMakeLists.txt file so that various parts of the build can refer to it. 281 24.8.1. Key CDash Concepts . In the CMake GUI, this is normally presented as a text edit field, so the developer can edit it to contain whatever arbitrary content they wish. This relates to how CMake parses each item to check if it is a compiler definition or not. Parentheses are also supported and have their usual mathematical meaning. The brief docs should generally be kept to one relatively short line, while the full docs can be longer and span across multiple lines if required. Although not recommended, it is also possible to specify compiler flags other than definitions with these commands. REPLACE will return the entire input string with each match replaced by replaceWith. Ultimately, the long term solution is to address the underlying problem highlighted by the warning. 125 # Use C++14 if available and allow compiler extensions for all targets set(CMAKE_CXX_STANDARD 14) set(CMAKE_CXX_STANDARD_REQUIRED OFF) set(CMAKE_CXX_EXTENSIONS ON) The situation for C is very similar. Some are aimed at meeting a particular need, while others are intended to be more generic and can be used in a variety of situations. These commands largely behave in a comparable way to their file()-based counterparts, with only slight variations. The set of options and features associated with find_package() is considerably richer than what is provided for include() and Chapter 23, Finding Things is dedicated to covering the topic in detail. 13.1.2. Multiple Configuration Generators Some generators, notably Xcode and Visual Studio, support multiple configurations in a single build directory. Note that setting the STRINGS property on the variable doesn’t prevent that variable from having other values assigned to it, it only affects the widget used by cmake-gui when editing it. 94 11.5. Recommended Practices . The following example and its output demonstrate the difference: function(func arg) if(DEFINED arg) message("Function arg is a defined variable") else() message("Function arg is NOT a defined variable") endif() endfunction() macro(macr arg) if(DEFINED arg) message("Macro arg is a defined variable") else() message("Macro arg is NOT a defined variable") endif() endmacro() func(foobar) macr(foobar) Function arg is a defined variable Macro arg is NOT a defined variable Aside from that difference, functions and macros both support the same features when it comes to argument processing. Note that in CMake 3.1 and earlier, an error was reported if length pointed past the end of the string. If it is more desirable to break up the file contents line-by-line, the STRINGS form may be more convenient. Note that newlines are not automatically added, so the text from the APPEND line in the above example continues directly after the WRITE line’s text without a break. The input strings are concatenated before substitution. Testing . This ensures the configure step doesn’t block indefinitely and allows an unexpectedly long configure step to be treated as an error. Most projects will probably find specifying language requirements this way to be tedious and fragile, so they should only be used if the situation clearly warrants it. This allows the project to effectively say "I need at least CMake X, but I am safe to use with policies from up to CMake Y". The basic form of this command is: add_executable(targetName source1 [source2 ...]) This creates an executable which can be referred to within the CMake project as targetName. 275 24.6. Cross-compiling And Emulators . • Generation of CMakeLists.txt or other files which need to be included or processed as part of the current configure step. Such options must be placed before -P. It should also be noted that once an imported target has been promoted to have global visibility, it is able to support the creation of an alias referring to it. The reader should expect to encounter plenty of examples and tutorials online and elsewhere which still recommend patterns and approaches employing the older methods using variables and directory property commands, but it should be understood that the target_...() commands should be the preferred approach in the CMake 3.0+ era. Instead of specifying most things through 139 variables which then had to be managed manually by the project, or by directory level commands that would apply to all targets in a directory and below without much discrimination, each target gained the ability to carry all the necessary information in its own properties. All the same variables that affect how the code is built for check_c_source_compiles() and check_cxx_source_compiles() also have the same effect for these two modules’ macros as well. Rather than repeating the steps for every test case, the function allows the steps to be defined once and then each test case becomes a simple one-line definition. They always contain absolute paths. 87 cmake_print_properties([TARGETS target1 [target2...]] [SOURCES source1 [source2...]] [DIRECTORIES dir1 [dir2...]] [TESTS test1 [test2...]] [CACHE_ENTRIES var1 [var2...]] PROPERTIES property1 [property2...]) This macro essentially combines get_property() with message() into a single call. Temporary debugging messages frequently use no mode keyword for convenience, but if they are likely to remain part of the project for any length of time, it is better that they too specifying a mode (typically STATUS). CMAKE_SOURCE_DIR The top-most directory of the source tree (i.e. where the top-most CMakeLists.txt file resides). If return() is called within a function, processing returns immediately to the caller, i.e. the rest of the function is skipped. CMake gives the cache variables suitable default values the first time it runs in a build directory. 3 1. Alternatively, setting the standard via a compile meta feature like cxx_std_11 also avoids many of the problems of setting individual features. Some file types need to have the ${someVar} form preserved without substitution. Once set, cache variables remain set until something explicitly removes them from the cache. Again, the test is case-insensitive. TLS_VERIFY value This option accepts a boolean value indicating whether to perform server certificate verification when downloading from a https:// url. As explained above, return() does not return a value, it only returns processing to the parent scope.) get_source_file_property(resultVar sourceFile propertyName) Again, no APPEND functionality is provided for the setter, while the getter is really just syntax shorthand for the generic get_property() command and offers no new functionality. These alias executables are covered in detail in Chapter 16, Target Types. The build directory will be called build, or some variation thereof. For strings, values are compared lexicographically. Chapter 12, Policies discusses policy settings in more detail and explains how to tailor this behavior as needed. This is somewhat analogous to the way configure_file() 166 substitutes CMake variables except this time the substitution is for generator expressions. Recommended Practices . The most generic of these, set_property() and get_property(), allow setting and getting any property on any type of entity. The name will be treated case-insensitively, so upper/lowercase conventions are more a matter of style (the CMake documentation follows the convention that command names are all lowercase with words separated by underscores). The following examples demonstrate the required syntax: set(longStr abcdefabcdef) string(REGEX MATCHALL "[ace]" matchVar ${longStr}) string(REGEX REPLACE "([de])" "X\\1Y" replVar ${longStr}) message("matchVar = ${matchVar}") message("replVar = ${replVar}") The resultant output of the above is: matchVar = a;c;e;a;c;e replVar = abcXdYXeYfabcXdYXeYf Extracting a substring is also possible: string(SUBSTRING input index length outVar) The index is an integer defining the start of the substring to extract from input. These custom commands are inherently platform specific, which works against one of the main reasons many projects use CMake in the first place, i.e. to abstract away platform differences or to at least support a range of platforms with minimal effort. Therefore, prefer to pass information directly to CMake through cache variables instead wherever possible. These follow a similar pattern to the other property-specific setter and getter commands: 75 set_source_files_properties(file1 [file2...] PROPERTIES propertyName1 value1 [propertyName2 value2] ... 155 18. It allows the added directory to change whatever variables it wants without affecting variables in the calling scope. Relative paths are converted to absolute paths automatically and are treated as relative to the current source directory. This can sometimes lead to unexpected matches and typically developers would provide just the one input string in most situations. To help with this, CMake provides the CMakePushCheckState module which defines following three macros: cmake_push_check_state([RESET]) cmake_pop_check_state() cmake_reset_check_state() These macros allow the various CMAKE_REQUIRED_... variables to be treated as a set and to have their state pushed and popped onto/from a virtual stack. 300 25.2. Installing Targets . Conversely, it is easy to go overboard and to try to make everything a generator expression. The $ and $ generator expressions allow different paths to be specified for building and installing. 144 17.3. Commands That Generate Files . Consider a C++ project which can make use of the override, final and nullptr keywords if available and which aims to support the GNU, Clang, Visual Studio and Intel compilers on any of the platforms those compilers support. The directories added by link_directories() only have an effect when CMake is given a bare library name to link to. 59 8.2. Argument Handling Essentials . This option stores the result of each command in the variable named by resultsVar as a list. No such chaining functionality is provided for VARIABLE or CACHE, since these already chain to the parent variable scope by design. A missing component is treated as zero. They can be useful for representing the details of header-only libraries, collections of resources and many other scenarios and should be strongly preferred over trying to achieve the same result with variables or directorylevel commands alone. The expanded basic form of the command is therefore: add_library(targetName [STATIC | SHARED | MODULE | OBJECT] [EXCLUDE_FROM_ALL] source1 [source2 ...]) Prior to CMake 3.12, object libraries cannot be linked like other library types (i.e. they cannot be used with target_link_libraries()), they require using a generator expression of the form $ as part of the list of sources of another executable or library target. 48 6.3. Recommended Practices . That culminated in the test fixtures feature that is now a part of CMake, but more importantly I was really struck by the positive experience I had while making that contribution. This occasionally leads to a misunderstanding by some developers that an empty build type is equivalent to Debug, but this is not the case. The result of the call will be a true value if no matching diagnostic message is issued. For reasons that will become clear in later chapters, CMake generally uses the term binary directory, but among developers, the term build directory tends to be in more common use. The names CMake uses for related properties and variables usually follow the same pattern, with the variable name being the property name with CMAKE_ prepended. These are each analogous to their same-named target properties except that they apply only to the individual source file on which they are set. CMAKE_CURRENT_LIST_LINE Holds the line number of the file currently being processed. It is the second of the above points which tends to surprise many developers. The new command’s name and treatment of the definition items is consistent with the analogous target_compile_definitions() command. ◦ The myExe target is built. 14.3. Compiler And Linker Variables Properties are the main way that projects should seek to influence compiler and linker flags. 135 Another use of interface libraries is to provide a convenience for linking in a larger set of libraries, possibly encapsulating logic that selects which libraries should be in the set. Projects may want to use certain compiler flags only with a particular compiler or when building for a particular platform, for example. Logically, directory properties sit somewhere between global properties which apply everywhere and target properties which only affect individual targets. This build type is therefore most useful when the performance of a Debug build is not acceptable even for a debugging session. One situation is where a project wants to set a low minimum CMake version, but still take advantage of later features if a later version is used. CMake will always treat the variable as a string during processing, the type is just to improve the user experience in GUI tools. Prior to CMake 3.10, the file(GENERATE...) command had different handling of relative paths compared to most other commands provided by CMake. Of the optional keywords, all but SET have little value unless the project has explicitly called define_property() to populate the requested information for the particular entity. When setting a variable’s value, CMake doesn’t require those values to be quoted unless the value contains spaces. 175 19.1. Project Version . • Source properties offer a fine granularity on the level of control of compiler options, etc. These are examples of what CMake refers to as the build type. With CMake 3.7 or later, the following options are also available for both DOWNLOAD and UPLOAD: USERPWD username:password Provides authentication details for the operation. Rather than requiring all projects to 10 immediately deal with this, CMake provides policy mechanisms which allow the project to say “Behave like CMake version X.Y.Z”. A variety of information will be logged before CMake finishes with lines like the following upon success: -- Configuring done -- Generating done -- Build files have been written to: /some/path/build The above highlights that project file creation actually involves two steps; configuring and generating. Loosely speaking, the start to end process according to CMake looks something like this: CMake CTest CPack Project File Generation Build Configure Test Package Generate The first stage takes a generic project description and generates platform-specific project files suitable for use with the developer’s regular build tool of choice (e.g. make, Xcode, Visual Studio, etc.). For functions, each argument is a CMake variable and has all the usual behaviors of a CMake variable. For example, conditionally including a source file depending on the build type can be done relatively concisely, as the example given earlier for $ showed. 77 • For target properties, use of the various target_...() commands is strongly recommended over manipulating the associated target properties directly. Cygwin . Both of these are covered in more detail in their own respective chapters: Chapter 12, Policies and Chapter 9, Properties. If the value matches, the expression evaluates to true. endmacro() Once defined, the function or macro is called in exactly the same way as any other CMake command. An important exception to this is that the first time CMake is run on a project, the CC and CXX environment variables of the child process are explicitly set to the C and C++ compilers being used by the main build (if the main project has enabled the C and C++ languages). When a variable is watched, all attempts to read or modify it are logged. Projects should carefully consider whether the use of a compiler detection header is worth the complexity before diving in and making use of the WriteCompilerDetectionHeader module.) 115 The target_compile_options() command is also quite straightforward. Much like their role in other programming languages, functions and macros are the primary mechanism for projects and developers to extend CMake’s functionality and to encapsulate repetitive tasks in a natural way. Generator expressions cannot be used everywhere, but they are supported in many places. 51 7.1.2. Scope . For example, C++17 removed std::auto_ptr, so if code expects to be compiled with an older language standard and still uses std::auto_ptr, it could fail to compile if the toolchain strictly enforces this removal. This support has been extended in subsequent versions and from CMake 3.6 covers most common compilers (CMake 3.2 added most of the compiler support, 3.6 added the Intel compiler). FILES_MATCHING REGEX .*_private\\.h EXCLUDE PATTERN *.h PATTERN *.sh PERMISSIONS OWNER_READ OWNER_WRITE OWNER_EXECUTE) If the whole source should be copied but permissions need to be overridden for just a subset of matched files, the FILES_MATCHING keyword can be omitted and the patterns and regular expressions are used purely to apply permission overrides. 22 5.1. Variable Basics . The TO_CMAKE_PATH form converts all directory separators in input to forward slashes. Since functions do not introduce a new policy scope, a policy change can affect the caller if the change is not properly isolated using the appropriate push-pop logic. The USES_TERMINAL option is supported for CMake 3.2 and later. This will be discussed in more detail shortly. Attempting to delete a directory with the REMOVE form will have no effect. Prefer to use looping commands which avoid ambiguous or misleading code. Examples of this include sub-parts of the project that were written for older CMake versions and therefore don’t use the target_...() commands, or external libraries with imported targets that omit some important flags needed by targets linking to them. It produces a header file that the project’s sources can #include to pick up appropriately specified compiler defines. On its own, this is already useful, but CMake comes with a whole host of other features which bring great flexibility and convenience. Rather than listing out each item explicitly, the arguments can also be specified by one or more list variables using the more general form of the command: foreach(loopVar IN [LISTS listVar1 ...] [ITEMS item1 ...]) # ... This includes the named arguments, so the first named argument could also be referenced via ARG1, etc. A typical developer could be forgiven for losing at least some of their sanity trying to keep on top of the whole picture. Unlike the compiler properties, only LINK_LIBRARIES has an equivalent interface property, INTERFACE_LINK_LIBRARIES. If providing passwords with this option, the content should come from outside the project, such as from an appropriately protected file read from the user’s local machine at configure time. On non-Apple platforms, the MACOSX_BUNDLE keyword is ignored. It is more than just shorthand for get_property(), although it can be used simply to retrieve the value of any global property. While they can be convenient in a 122 few specific circumstances, consistent use of the target_...() commands instead will establish clear patterns that all developers in a project can follow. This applies whether the old name is for a builtin command or a custom function or macro. 6.1.1. Basic Expressions The most basic of all expressions is a single constant: if(value) CMake’s logic for what it considers true and false is a little more involved than most programming languages. A CMake function or macro is defined as follows: function(name [arg1 [arg2 [...]]]) # Function body (i.e. commands) ... cmake -DOPTION_A=1 -DOPTION_B=foo -P myCustomScript.cmake 17.6. Combining The Different Approaches The following example demonstrates many of the features introduced in this chapter. add_library(builtElsewhere STATIC IMPORTED) set_target_properties(builtElsewhere PROPERTIES IMPORTED_LOCATION /path/to/libSomething.a) # Promote the imported target to global visibility set_target_properties(builtElsewhere PROPERTIES IMPORTED_GLOBAL TRUE) It is important to note that an imported target can only be promoted if it is defined in exactly the same scope as the promotion. 114 If the SYSTEM keyword is specified, the compiler will treat the listed directories as system include paths on some platforms. 6.2.1. foreach() CMake provides the foreach() command to enable projects to iterate over a set of items or values. Sometimes, however, a project may want more fine-grained control than what the cmake_minimum_required() command provides. Logical operations are also supported: $ $ $ Each expr is expected to evaluate to either 1 or 0. Build artifacts may need to be archived or a notification service contacted. Items listed with DEPENDS can be CMake targets or files (compare this with the DEPENDS option for add_custom_target() which can only list files). • Each cache variable has a type, which must be one of BOOL, FILEPATH, PATH, STRING or INTERNAL. The third case is even more problematic because every time CMake is run, the flags will be appended again, leading to an ever growing and repeating set of flags. CMake provides two equivalent GUI tools, cmake-gui and ccmake, which allow developers to manipulate cache variables interactively. Therefore, developers should expect that a DEPENDS section will typically be needed unless the custom command doesn’t require anything created by the build or any of the project’s source files. 116 14.3. Compiler And Linker Variables . The message will be printed and processing will stop immediately. 89 11.4. Other Modules . Furthermore, the basic argument handling described above is quite rigid compared to many of CMake’s own built-in commands which support keyword-based arguments and flexible argument ordering. Most optimizations for speed are typically applied, but most debug functionality is also enabled. External Content . This means that the generated files won’t exist when the file(GENERATE...) command returns, so the files cannot be used as inputs to something else during the configure phase. Developers who are aware of this behavior are sometimes tempted to exploit it to try to create a wrapper around an existing command like so: function(someFunc) # Do something... This becomes particularly problematic if developers are using a version control system and are switching between branches, etc. In effect, the same thing is being done as the multiple build directory arrangement described above for single configuration generators, it’s just that the IDE is handling the directory structure on the developer’s behalf. The keyword being present specifies one thing, its absence another. It is possible to omit the keyword defining what type of library to build. The more recent target-focused features should generally be preferred instead, since they greatly improve the robustness of the build and offer much greater precision of control over compiler and linker behavior. Developers can be tempted to get too over-zealous with checking all manner of things, which can result in slowing down the configure stage for sometimes questionable gains. Specifying the build configuration is covered in depth in Chapter 13, Build Type. Another related trap is when an include() is used after targets have been created and the included file(s) modify the compiler or linker variables. A range of different generators are supported, with the more commonly used ones listed in the table below. Substitution of CMake variables with either the ${someVar} or @ form can also be performed on strings, not just files. • Any new variable created in the child scope will not be visible to the calling scope. Targets created in child directory scopes will only be affected if created after the call. Understand when variable values are used One of the more obscure aspects of the compiler and linker flag variables is the point in the build process at which their value actually gets used. The following example shows how to define a custom target which uses two other targets as part of the command and argument list: add_executable(hasher hasher.cpp) add_library(myLib api.cpp) add_custom_target(createHash COMMAND hasher $) When a target is used as the command to execute, CMake automatically creates a dependency on that executable target to ensure it is built before the custom target. CMake provides dedicated functions for modifying them in a more convenient and robust manner which also encourages 113 clear specification of dependencies and transitive behavior between targets. Consider the following scenarios: • A project wants to set a low minimum CMake version, but it also wants to take advantage of newer behavior if it is available. The first implementation of printme is created and made available as a command of that name. 50 The optional EXCLUDE_FROM_ALL keyword is intended to control whether targets defined in the subdirectory being added should be included in the project’s ALL target by default. Except within a quoted string, anything after a # on a line within a CMakeLists.txt file is also treated as a comment. These directory properties and the commands typically used to manipulate them lack the consistency shown by their target-based equivalents, which is another reason they should generally be avoided by projects where possible. 156 18.2. Copying Files . CMake uses this file to save details so that if it is re-run again, it can re-use information computed the first time and speed up the project generation. 10.2. Target Details Another common use of generator expressions is to provide

information about targets. CMake supports building a variety of different kinds of libraries, taking care of many of the platform differences, but still supporting the native idiosyncrasies of each. Unlike normal variables which have a lifetime limited to the processing of the CMakeLists.txt file, cache variables are stored in the special file called CMakeCache.txt in the build directory and they persist between CMake runs. It burrowed its way into my brain and sat there for a few months until one day I decided to explore how hard it would be to add that feature myself. Some kind of placeholder is needed to tell CMake "For whichever configuration is being built, use the directory of the final executable". Quite often, however, such commands can instead be expressed in a platform independent manner using CMake's command mode (-E). Without -f, a non-zero exit code is returned, whereas with -f, a zero exit code will be returned. Even the name itself can cause some confusion among developers, since an object library is not a true library, but rather just a set of uncombined object files, but developers sometimes expect it to behave like a real library. In order to rigorously require the variable to have one of the defined values, a project must explicitly perform that test itself. During the configuring phase, CMake reads in the CMakeLists.txt file and builds up an internal representation of the entire project. The target could have been defined in any directory, as long as it is known at the point where the if test is performed. To copy the directory's contents into the destination instead, append a forward slash (/) to the source directory like so: file(COPY base/srcDir DESTINATION destDir) file(COPY base/srcDir/ DESTINATION destDir) # --> destDir/srcDir # --> destDir By default, the COPY form will result in all files and directories keeping the same permissions as the source from which they are copied, whereas the INSTALL form will not preserve the original permissions. Sometimes the policy warnings cannot be addressed immediately, but the warnings could be undesirable. The developer selects the configuration at build time, well after CMake has finished. This is unfortunate, since the more intuitive expectation is probably that the loop body executes value times. # Generate unique files for all but Release file(GENERATE OUTPUT ${CMAKE_CURRENT_BINARY_DIR}/outFile-$.txt INPUT ${CMAKE_CURRENT_SOURCE_DIR}/input.txt.in CONDITION $) # Embedded content, bracket syntax does not # prevent the use of generator expressions file(GENERATE OUTPUT ${CMAKE_CURRENT_BINARY_DIR}/details-$.txt CONTENT [[Built as "$" for platform "$". They do not need to be escaped to prevent them being interpreted as the start or end of an argument. As a project grows in complexity, these three keywords have a stronger impact on how inter-target dependencies are handled. This type can be obtained using get_property() with the property name TYPE. CMake offers the traditional boolean logic as well as various other conditions such as file system tests, version comparison and testing for the existence of things. A number of these features are frequently misused, some due to such misuse being prevalent in online tutorials and examples, leading to the belief that it is the right way to do things. The following should help to demonstrate the behavior. There are situations, however, where the user will want to add their own compiler or linker flags. This allows the script to easily switch between generator types without having to be modified. 386 27.3. ExternalData . To help reduce these differences, the VERBATIM option can be used to ensure that the only escaping done is that by CMake itself when parsing the CMakeLists.txt file. It also allows the function to be more easily extended in the future because there is no reliance on argument ordering or for all arguments to always be provided, even if not relevant. It implements all of the same options as the TARGET form as well as some additional options related to dependency handling and appending to a previous OUTPUT command set. PATH Like FILEPATH, but GUI tools present a dialog that selects a directory rather than a file. • Builds can be composed of subcomponents which are defined independently from the top level project consuming them. Furthermore, it is strongly recommended that projects set all three properties/variables rather than just some of them. In some cases, a deprecation warning may be issued even if the policy has been explicitly set, but this is typically only for a policy that has already been documented as deprecated for a long time (many releases). Compile feature requirements are controlled by the target properties COMPILE_FEATURES and INTERFACE_COMPILE_FEATURES, but these properties are typically populated using the target_compile_features() command rather than being manipulated directly. 185 20.2. Linking Static Libraries . The find_package() command also does a similar thing to include(), pushing and popping upon starting and finishing processing of its associated FindXXX.cmake module file respectively. If the opening brackets are immediately followed by a newline character, that first newline is ignored, but subsequent newlines are not. This is the purpose of the PARENT_SCOPE keyword in the set() command. 5.6. String Handling As project complexity grows, in many cases so too does the need to implement more involved logic for how variables are managed. 80 10.2. Target Details . The ability to filter the variables shown is also not provided and the methods for editing a variable are not as rich as with cmake-gui. 193 20.5.1. Specifying Default Visibility . For example, a special tool may need to be run to produce source files or to post-process a target after it has been built. Multiple permissions can be (and usually are) listed together. INTERFACE_... properties can be set with set_property() or set_target_properties() with any CMake version. Another way to filter the list of variables shown is to enter text in the Search area, which results in only showing variables with the specified text in their name or value. Static libraries are simpler, have more complete and robust support from earlier CMake versions and they are well understood by most developers. These properties are specific to the type of packaging being used and are typically not needed by most projects. Chapter 19, Specifying Version Details covers this in depth and explains how to refer to this version information later in the CMakeLists.txt file. As a temporary way to work around such situations, CMake provides the CMAKE_POLICY_DEFAULT_CMPxxxx and CMAKE_POLICY_WARNING_CMPxxxx variables where xxxx is the usual four-digit policy number. The newer features also tend to be more consistent in their behavior and in the way they are meant to be used. 212 21.6.3. Android . The interaction between the two types of variables can be unexpected for developers new to CMake. Exactly one of the property types must be specified and each of the named properties will be printed for each entity listed. 220 22.3. Frameworks . Attempting to use any tool before understanding at least the basics of what it does and how it is meant to be used is most likely going to result in frustration. The modules follow a similar structure to the compilation case, each providing a single macro implementing the check: 90 include(CheckCSourceRuns) check_c_source_runs(code resultVar) include(CheckCXXSourceRuns) check_cxx_source_runs(code resultVar) There is no FAIL_REGEX option for these macros, as success or failure is determined purely by the test process' exit code. If using the RANGE form of foreach(), always specify both the start and end values. Only those symbols that satisfy the naming requirements for a preprocessor symbol can be used. One or both of ITEMS and/or LISTS must be provided when using this more general form. Files may need to be copied, verified or a hash value computed. The add_definitions() command has a number of shortcomings. target_link_libraries(targetName item [item...]) The above form is generally equivalent to the items being defined as PUBLIC, but in certain 19 situations, they may instead be treated as PRIVATE. Manipulation of the build type is the subject of this chapter, while the next chapter presents more specific details of controlling compiler and linker options. Like the projectName, the version details are used by CMake to populate some variables and as default package metadata, but other than that, the version details don't have any other significance. Further check modules continue to be added to CMake as it evolves, so consult the CMake module documentation to see the full set of available functionality. Another category of target is the utility or custom target. The following table demonstrates the differences. Furthermore, unlike add_subdirectory() where the fileName will always be CMakeLists.txt, the name of the file can be anything when using include(), so it can be difficult for the included file to determine its own name. The most significant benefit is that it can be used in contexts where CMake automatically replaces a target name with its location on disk, such as when executing commands for tests or custom tasks (both covered in later chapters). Compiler/linker variables are single strings, not lists If multiple compiler flags need to be set, they need to be specified as a single string, not as a list. Up to length characters will be extracted, or if length is -1, the returned substring will contain all characters up to the end of the input string. • A cache variable can be marked as advanced with the mark_as_advanced() command, which is really just setting the boolean ADVANCED cache variable property. A project may support some compiler features being optional, such as keywords intended to guide the developer or provide an increased ability for the compiler to catch common mistakes. This is covered in more detail in Chapter 10, Generator Expressions. Similarly, if a listed file is modified, the custom command will be executed if anything requires any of the custom command's output files. helpers:FILEPATH=subdir/helpers.txt ... CMake follows similar commenting conventions as Unix shell scripts. This command follows a very similar form to the various other target_...() commands provided by CMake: target_compile_features(targetName feature1 [feature2 ...] [feature3 [feature4 ...]] ... While developers will typically invoke their selected build tool directly in day-to-day development, invoking it via the cmake command as shown above can be more useful in scripts driving an automated build. The NO_POLICY_SCOPE option was added as a way for projects using later CMake versions to revert back to the old behavior for specific parts of the project, but its use is generally discouraged and should be unnecessary for new projects. • A quoted string. In general, it is fine to override a function or macro as long as it does not try to call the previous implementation like in the above discussion. For Unix-based platforms where shared libraries typically have version details in their file name, these will also be included. Conversely, if creating a project intended to be built and distributed as part of the operating system itself (common for Linux), the minimum CMake version is likely to be dictated by the version of CMake provided by that same distribution. When used this way, the last item in the list has index -1, the second last -2 and so on. Be aware that for many compilers/linkers, the same flag is used to control both the language standard and whether or not extensions are enabled. If the output of the test compilation and linking matches the regex regular expression, the check will be deemed to have failed, even if the code compiles and links successfully. ② The parent's myVar has been modified by the set() call in the child scope. When CMake has completed its run, it will have saved a CMakeCache.txt file in the build directory. Those policies that were added after the specified version will effectively be reset to empty. ④ childVar is still undefined in the child scope before it is set. Because of this delayed nature of the compiler and linker variables, they can be fragile to work with. For Visual Studio compilers before VS2015 Update 3, no flags would be added since the compiler either supports C++11 by default or it has no support for C++11 at all. Again, module files are one of the more common places where the policy stack might be manipulated like this. This is useful when the generated file is used as an input in a build target, such as a generated header file, since it can prevent unnecessary rebuilds. This ensures that a particular minimum set of CMake functionality is available before proceeding. These commands define not only properties for a particular target, they also define how that information might be propagated to other targets that link against it. The entire process from start to finish can be driven from CMake itself, with the testing and packaging steps available simply as additional targets in the build. Furthermore, if any listed file is itself an output file of another custom command in the same directory scope, that other custom command will be executed first. ◦ The custom command is executed to create onTheFly.cpp. This applies strictly only to the targets created in the current CMakeLists.txt file or other files pulled in via include(), but not to any targets created in parent or child directory scopes. 15.1. Setting The Language Standard Directly The simplest way for a project to control the language standards used by a build is to set them directly. In addition to checking whether code can be built, CMake also provides modules that test whether C or C++ code can be executed successfully. If no LANGUAGES option is provided, CMake will default to C and CXX. Target Types CMake supports a wide variety of target types, not just the simple executables and libraries introduced back in Chapter 4, Building Simple Targets. 344 26.4.1. Simple Archives . Using Subdirectories . Developers should consult the CMake reference documentation for the full set of supported conditional expressions. NAME_WE Extract the base file name only. Strive to avoid embedding any sort of authentication details (usernames, passwords, private keys, etc.) in any file stored in a version control system for the project's sources. It also ensures the build uses the same CMake version during the build stage as was used in the configure stage, even if the user's PATH environment variable changes. In situations where a target has both its _STANDARD property set and compile features specified (directly or transitively as a result of INTERFACE features from something it links to), CMake will enforce the stronger standard requirement. The if() command conveniently treats both non-empty error strings and integer values other than 0 as boolean true (unless a project is unlucky enough to have an error string that satisfies one of the special cases, see Section 6.1.1, "Basic Expressions"). In the case of VARIABLE scope queries, the result will only be true if the named variable has been explicitly defined with the define_property() command (see below). Core language features, key concepts and important building blocks were presented, providing a solid foundation for a deeper exploration of CMake's functionality. $ $ $ $ $ Another very useful conditional expression is testing the build type: $ This will evaluate to 1 if arg corresponds to the build type actually being built and 0 for all other build types. A good example of the latter is adding a build type for profiling or code coverage, both of which require specific compiler and linker settings. Subsequent chapters expand that knowledge to demonstrate how to get the most out of what CMake has to offer. 18.5. Downloading And Uploading The file() command has a number of other forms which carry out different tasks. For convenience, CMake provides dedicated commands for setting and getting directory properties which are a little more concise than their generic counterparts. This variable never changes its value. This means that using ${CMAKE_CURRENT_BINARY_DIR} as part of the working directory should not be necessary, since a relative path already implies it. 83 10.4. Recommended Practices . To avoid creating situations which may lead to such confusion, if the include_directories() command must be used, 116 prefer to call it early in a CMakeLists.txt file before any targets have been created or any subdirectories have been pulled in with include() or add_subdirectory(). The propName would normally match one of the properties defined in the CMake documentation, a number of which are discussed in later chapters. These custom targets are defined using the add_custom_target() command: add_custom_target(targetName [ALL] [command1 [args1...]] [COMMAND command2 [args2...]] [DEPENDS depends1...] [BYPRODUCTS [files...]] [WORKING_DIRECTORY dir] [COMMENT comment] [VERBATIM] [USES_TERMINAL] [SOURCES source1 [source2...]]) A new target with the specified targetName will be available to the build. In fact, they all follow the same pattern and apply the PRIVATE, PUBLIC and INTERFACE keywords in the same way. The only difference is that the PARENT_SCOPE keyword should not be used within the macro since it already modifies the variables in the caller's scope. They can also contain quotes, which require escaping with backslashes. ② Ends the current innerVar iteration and moves on to the next innerVar item. from full paths, or convert between absolute and relative paths. Developers should consult the CMake reference documentation for the available properties and their intended purpose. Category Generator Examples Multi-config Visual Studio Visual Studio 15 2017 Yes Visual Studio 14 2015 ⋮ Xcode Xcode Yes Ninja Ninja No Makefiles Unix Makefiles No MSYS Makefiles MinGW Makefiles NMake Makefiles Some of the generators produce projects which support multiple configurations (e.g. Debug, 7 Release, etc.). Use of an undefined variable simply results in an empty string being substituted with no error or warning, much like Unix shell scripts. It is particularly convenient when a large number of language features are being used, since each feature does not have to be explicitly specified. CMake provides a variety of features related to working with files and directories. When a target is created, the initial value of this property is taken from the CMAKE__STANDARD_REQUIRED variable. function(func myArg) message("myArg = ${myArg}") endfunction() macro(macr myArg) message("myArg = ${myArg}") endmacro() func(foobar) macr(foobar) 60 Both the call to func() and the call to macr() print the same thing: myArg = foobar In addition to the named arguments, functions and macros come with a set of automatically defined variables which allow processing of arguments in addition to or instead of the named ones: ARGC This will be set to the total number of arguments passed to the function. The _EXTENSIONS target property controls whether those extensions are enabled for that particular target. It can be tempting to focus on using one particular type of project generator in the early stages of a project, especially if the developer is not accustomed to writing cross-platform software. Similarly, if a target is referred to in a generator expression anywhere in the command or its arguments, a dependency is automatically created on that target too. 213 21.7. Recommended Practices . Starting with CMake 3.0, these modules are progressively being updated to also provide imported targets where appropriate. The BYPRODUCTS option can be used to list other files that are created as part of running the command(s). Instead, the files are written as part of the generation phase, which occurs after all of the CMakeLists.txt files have been processed. If the configure stage is successful, the generate stage can then be executed to create the build tool's project files in the build directory. The custom target is always considered out of date, so bringing any target that depends on it up to date will result in the commands being executed. The above form ensures the command is available regardless of what CMake version is being used. Replacing existing flags like this forces developers to go digging into the project files to find where and how to modify any lines which modify the relevant flags. Thus, a C++ file being built with a Debug configuration would have compiler flags from both CMAKE_CXX_FLAGS and CMAKE_CXX_FLAGS_DEBUG. For example, to add a compile feature which ensures that a target and anything that links against it has C++14 support enabled, the following could be used: target_compile_features(targetName PUBLIC cxx_std_14) If a project needs to support CMake versions earlier than 3.8, then the above meta-feature will not be available. A continuous integration system performing incremental builds is a prime candidate for being caught out by such use. The CMake documentation provides a complete list of all available modules, each with its own help section explaining what the module provides and how it can be used. 18.6. Recommended Practices A range of CMake functionality related to file handling has been presented in this chapter. [PARENT_SCOPE]) The name of the variable, varName, can contain letters, numbers and underscores, with letters being case-sensitive. It supports variables, if-then-else logic, looping and inclusion of other files to be processed. On Windows, the default library name would be targetName.dll, on Apple platforms it would be libtargetName.dylib and on other Unix-like platforms it would typically be libtargetName.so. CMake allows a library target to be defined as an object library, meaning it isn't a library in the usual sense, it is just a collection of object files that CMake associates with a target but doesn't actually result in a final library file being created. Some of the more fundamental Check... modules are those that compile and link a short test file into an executable and return a success/fail result. PUBLIC Public dependencies specify that not only does library A use library B internally, it also uses B in its interface. If an item is preceded by the debug keyword, then it will only be added if the build is a debug build. As a project evolves, it may start to use include() for some directories where this makes sense. The underlying string value held by the variable will conform to one of the ways CMake represents booleans as strings (ON/OFF, TRUE/FALSE, 1/0, etc. In particular, the IS_NEWER_THAN operator returns true if either file is missing or if both files have the same timestamp (which includes if the same file is given for both file1 and file2). Consider the project name and executable name as being separate, even if initially they start out the same. # Preserves the cache variable contents, appends new flags safely set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Werror") Prefer appending over replacing flags As touched on above, developers are sometimes tempted to unilaterally set compiler flags in their CMakeLists.txt files like so: # Not ideal, discards any developer settings from cache set(CMAKE_CXX_FLAGS "-Wall -Werror") Because this discards any value set by the cache variable, developers lose their ability to easily inject their own flags. If a dependency on any other target needs to be specified, the add_dependencies() command can be used to define that relationship. The singleValueKeywords each require exactly one additional argument after the keyword when they are used, whereas multiValueKeywords require zero or more additional arguments after the keyword. Source code may want to embed the project version so that it can be displayed to the user or recorded in a log file, packaging steps may need it to define release version details and so on. Therefore, checking for the success of a call to execute_process() is generally relatively simple: execute_process(COMMAND runSomeScript RESULT_VARIABLE result) if(result) message(FATAL_ERROR "runSomeScript failed: ${result}") endif() 150 From CMake 3.10, if the result of each individual command is required rather than just the last one, the RESULTS_VARIABLE option can be used instead. For example: # Any sort of real library (SHARED, STATIC, MODULE # or possibly OBJECT) add_library(myRealThings SHARED src1.cpp ...) add_library(otherThings STATIC srcA.cpp ...) # Aliases to the above with special names add_library(BagOfBeans::myRealThings ALIAS myRealThings) add_library(BagOfBeans::otherThings ALIAS otherThings) Within the project itself, other targets would link to either the real targets or the namespaced targets (both have the same effect). The effect of this is to restrict the default visibility of the library to the current directory scope instead of global. With CMake 3.8 and later, if-then-else logic can also be expressed very conveniently using a dedicated $ expression: $ As usual, the expr must evaluate to 1 or 0. Furthermore, any path part of the destination must already exist, the RENAME form will not create intermediate directories. • include() does not introduce a new variable scope, whereas add_subdirectory() does. This means that A cannot be used without B, so anything that uses A will also have a direct dependency on B. Intuitively, one may expect that the variable's value at the time of the call to add_library() is what CMake uses, but in fact it is the variable's value at the end of processing for that directory scope that gets used. 126 15.2.1. Detection And Use Of Optional Language Features . The following example demonstrates how to implement the technique: function(func resultVar1 resultVar2) set(${resultVar1} "First result" PARENT_SCOPE) set(${resultVar2} "Second result" PARENT_SCOPE) endfunction() func(myVar otherVar) message("myVar: ${myVar}") message("otherVar: ${otherVar}") 65 The output of the above would be: myVar: First result otherVar: Second result Another alternative is for a function to document the variables it sets rather than allowing the caller to specify the variable names. The next chapter discusses this area in more detail. FILE_PERMISSIONS OWNER_READ OWNER_WRITE OWNER_EXECUTE GROUP_READ GROUP_EXECUTE WORLD_READ WORLD_WRITE) The COPY and INSTALL signatures both also preserve the timestamps of the files and directories being copied. Since the underlying directory properties support generator expressions, so do these two commands, with some caveats. # Common pattern, often used with variables defined # by commands such as option(enableSomething "...") if(enableSomething) # ... 10.3. General Information Generator expressions can provide information about more than just targets. It is not unusual for some or all of the linker flags to be empty strings on some platforms. • If control over file or directory permissions is required as part of the copy, file(COPY) or file(INSTALL) must be used. 6.1.3. Comparison Tests CMake separates comparison tests into three distinct categories: numeric, string and version numbers, but the syntax forms all follow the same pattern: if(value1 OPERATOR value2) The two operands, value1 and value2, can be either variable names or (possibly quoted) values. 164 18.3. Reading And Writing Files Directly CMake offers more than just the ability to copy files, it also provides a number of commands for reading and writing file contents. 11.1. Useful Development Aids The CMakePrintHelpers module provides two macros which make printing the values of properties and variables more convenient during development. For ultimate control, however, a command can be given which will be executed every time the variable is read or modified. 7.4. Recommended Practices The best choice between using add_subdirectory() or include() to bring another directory into the build is not always obvious. 7 2.4. Running The Build Tool . In contrast, regular executable targets built by the project are always global. The requirement to prefix each item with -D or /D to have it treated as a definition is not consistent with other CMake behavior. 56 7.4. Recommended Practices . For the most part, cache variable properties are aimed more at how the cache variables are handled in the CMake GUI and the console-based ccmake tool rather than affecting the build in any tangible way. Generator Expressions......Page 8810.1. Simple Boolean Logic......Page 8910.2. Target Details......Page 9110.3. General Information......Page 9210.4. Recommended Practices......Page 93Chapter 11. Custom build types can be defined by adding them to CMAKE_CONFIGURATION_TYPES and unwanted build types can be removed from that list. The GLOBAL keyword can be given to make them have global visibility instead, just like other regular targets. The $ generator expression is the only way to robustly 81 provide such functionality which works for all CMake project generators, including multi configuration generators like Xcode or Visual Studio. In recognition of this, CMake 3.12 introduced the add_compile_definitions() command as a replacement for add_definitions(): add_compile_definitions(SomeSymbol Foo=Value ...) The new command handles only compile definitions, it does not require any prefix on each item and generator expressions can be used without the VALUE-only restriction. Omitting CMAKE__STANDARD_REQUIRED or CMAKE__EXTENSIONS can often lead to unexpected behavior, as the defaults may not be what some developers intuitively expect. While this isn't the same as returning a value from the function, it does allow a value (or multiple values) to be passed back to the caller. CMake also creates a special file called CMakeCache.txt in the build directory to store various information for reuse on subsequent runs. All but the first of the above points also hold true for this second form. In cases where a project does need to manipulate a specific policy, it should check whether the policy is available using if(POLICY...) rather than testing the CMAKE_VERSION variable. If the target name already begins with lib, the resultant library file names end up with the form liblibsomething...., which people often assume to be a mistake. Each time cmake_push_check_state() is called, it effectively begins a new virtual variable scope for just the CMAKE_REQUIRED_... variables (and also the CMAKE_EXTRA_INCLUDE_FILES variable which is only used by the CheckTypeSize module). Keep in mind though that over time, the project may itself become a child of another project, at which point it would no longer be the top level of the build and the suitability of this compromise may be reduced. Unlike the include_directories(), add_definitions(), remove_definitions() and add_compile_definitions() commands, its behavior is very straightforward and predictable. The simplest method for controlling language standard requirements is to use the CMAKE__STANDARD, CMAKE__STANDARD_REQUIRED and CMAKE__EXTENSIONS variables. Consult the CMake documentation for details on those areas. Rather than simply removing the FORCE keyword, the correct behavior is to set a non-cache variable rather than the cache variable. This is not particularly robust, since cmake could be invoked from any directory, not just the build directory. 211 21.6. Examples . If using system-provided packages on Linux rather than the official releases, note that many distributions split cmake-gui out into its own package. Using Subdirectories For simple projects, keeping everything in one directory is fine, but most real world projects tend to split their files across multiple directories. The next chapter shows how to manipulate files directly with CMake instead of having to resort to such tools and methods. • If a file just needs to be copied but its name will change, the syntax of configure_file() is slightly shorter than file(COPY...), but either would be suitable. CMAKE_DEBUG_POSTFIX is often set to values like d or _debug, especially for Visual Studio builds where different runtime DLLs must be used for Debug and non-Debug builds, so packages may need to include libraries for both build types. The interested reader should consult the CMake reference documentation for the string() command for full details. set(list1 A B) set(list2) set(foo WillNotBeShown) foreach(loopVar IN LISTS list1 list2 ITEMS foo bar) message("Iteration for: ${loopVar}") endforeach() The output from the above would be: Iteration Iteration Iteration Iteration for: for: for: for: A B foo bar The foreach() command also supports the more C-like iteration over a range of numerical values: foreach(loopVar RANGE start stop [step]) When using the RANGE form of foreach(), the loop is executed with loopVar set to each value in the range start to stop (inclusive). ALGO can be any one of the hashing algorithms CMake supports, the most commonly used being MD5 and SHA1. Querying the target properties can, however, be useful from time to time when investigating unexpected compiler or linker command line flags. If CMake does not recognize a particular item as looking like a compiler define, that item will instead be added unmodified to the COMPILE_OPTIONS directory property. I'm humbled by the scale of contributions that have been made by so many and by the positive impact on the world of software development. This can lead to the error only becoming apparent much later at build time. As that pattern repeated itself, the idea of writing a book was born. The former provides a macro which builds a test C executable and the latter does the same as a C++ executable. The type can only be set to OBJECT with CMake 3.9 or later (imported object libraries were not supported before that version). # Restore earlier policy settings cmake_policy(POP) 99 Some commands implicitly push a new policy state onto the stack and pop it again at a well defined point later. This book tends to prefer the latter term since it is generally more intuitive. • The help string of a cache variable is typically set as part of a call to the set() command, but it can also be modified or read using the HELPSTRING cache variable property. The various methods can be used very effectively to carry out a range of tasks in a platform independent way, but they can also be misused. The type affects how the CMake GUI and ccmake present that cache variable in the UI and what kind of widget is used for editing its value. The command is expected to be the name of a CMake function or macro (see Chapter 8, Functions And Macros) and it will be passed the following arguments: variable name, type of access, the variable's value, the name of the current list file and the list file stack (list files are discussed in Chapter 7, Using Subdirectories). Properties . $, $ Any content can be converted to lower or upper case via these expressions. Release This build type typically provides full optimizations for speed and no debug information, although some platforms may still generate debug symbols in certain circumstances. 17.4. Configure Time Tasks Both add_custom_target() and add_custom_command() define commands to be executed during the build stage. It promotes cleaner definition of the project and allows the CMakeLists.txt for a given directory to focus more on just what that directory needs to define. They are designed to be noisy but informative, encouraging developers to update the project to the new behavior. Using them from the beginning of a project also forces developers to think about the dependencies between targets, which can help to highlight structural problems within the project much earlier. The use is best demonstrated by example. In these situations, use an alias target to provide an old name for a renamed target so that those external projects can continue to build and be updated at their convenience. 103 13.1. Build Type Basics . Periodically checking the build with a different project generator than the one a developer usually uses can save considerable future pain by discouraging generator-specific code where it isn't required. When regular expression matching in if(xxx MATCHES regex) commands and the group capture variables are needed, it is generally advisable to store the CMAKE_MATCH_ match results in ordinary variables as soon as possible. Compilers, linkers, testing frameworks, packaging systems and more all contribute to the complexity of deploying high quality, robust software. If a particular keyword isn't present in the argsToParse, its corresponding variable will be empty. 70 9.2. Global Properties . These enable fine-grained manipulation of compiler flags on a file-by-file basis rather than for all of a target's sources. Again, the old-style form has the same confusion over which target properties it affects and the PRIVATE/PUBLIC keyword form should be preferred for new projects. 8 2.5. Recommended Practices . On many platforms (i.e. just about all except Windows), a leading lib will be prefixed automatically when constructing the actual library name to make it conform to the platform's usual convention. These do not usually represent a physical library, instead they primarily serve to collect usage requirements and dependencies to be applied to anything that links to them. While this method is usually accurate, it is not unusual to see projects unilaterally set CMAKE_CONFIGURATION_TYPES even if using a single configuration generator. Those mentioned are all supported since at least CMake 3.0, so projects should generally be able to expect them to be available. From CMake 3.8, a per language meta-feature is available to indicate a particular language standard rather than a specific compile feature. 334 26.2. Components . 331 26. Again, in CMake versions prior to 2.8.0, the condition had to be repeated in the endwhile() command, but this is no longer necessary and is actively discouraged for new projects. All methods presented work exactly the same way on all platforms. The following read-only variables are updated automatically as each file is processed by CMake. Given the limited support for this option, projects should aim for a structure which does not require a PRE_BUILD custom command. For those situations where a project needs to refer to an external tool or library, prefer to do so through an imported target if one is available. The last form of the add_library() command is for defining an alias library: add_library(aliasName ALIAS otherTarget) A library alias is mostly analogous to an executable alias. Instead, the build tool would treat original.cpp as an input file used to create transformed.cpp. The full form shown previously with PRIVATE, PUBLIC and INTERFACE sections should be preferred, as it expresses the nature of dependencies with more accuracy. It also makes working with version control systems more difficult, since there are lots of files created by the build which either the source control tool has to know to ignore or the developer has to manually exclude during commits. DEFINED Returns true if a variable of the specified name exists. 347 26.4.3. WIX . Note that this means any flag that results in a compiler warning but successful compilation will still be deemed to have failed the check. In between, the project can modify the settings of any policies it needs to without having to explicitly save each one first. These properties behave as though they were PRIVATE, meaning they only specify requirements on that target and not on anything linking to it. As an example, for one particular compiler, the following variables for the C++ language are defined by default: CMAKE_CXX_FLAGS CMAKE_CXX_FLAGS_DEBUG -g -O0 CMAKE_CXX_FLAGS_RELEASE -O3 -DNDEBUG CMAKE_CXX_FLAGS_RELWITHDEBINFO -O2 -g -DNDEBUG CMAKE_CXX_FLAGS_MINSIZEREL -Os -DNDEBUG The handling of linker flags is similar. It can be a useful technique in specific situations, but it can reduce the portability of the project if it relies too heavily on such expressions. MODULE Specifies a library that is somewhat like a shared library, but is intended to be loaded dynamically at run-time rather than being linked directly to a library or executable. Fellow CMake developers Gregor Jasny and Christian Pfeiffer have been valuable contributors throughout the review process and I am truly grateful for their suggestions and insights. CMAKE_REQUIRED_QUIET If this option is present, no status messages will be printed by the macro. 9.3. Directory Properties Directories also support their own set of properties. 84 11. By default, cmake-gui only shows non-advanced variables, which typically presents just the main variables a developer would be interested in viewing or modifying. 99 12.3. Recommended Practices . The CMake reference documentation for each command typically lists any variables which can affect that command's behavior. Each of the files can provide functions, macros, variables and other features as appropriate. The prepending of one underscore to "save" the previous command only applies to the current name, it is not applied recursively to all previous overrides. It can sometimes be desirable to check for this and only include the file once, returning early for subsequent inclusions to prevent reprocessing the file multiple times. Alternatively, some action may need to be performed repeatedly until a particular condition is met. ⑥ childVar was defined in the child scope, so it is not visible to the parent and evaluates to an empty string. The component to extract from input is specified by component, which must be one of the following: DIRECTORY Extract the path part of input without the file name. These expressions also invite misuse, since they may appear to provide a way to do things like construct paths to things which could otherwise have been obtained using more robust methods like using TARGET_FILE expressions or other CMake features. IMPLICIT_DEPENDS 148 directs CMake to invoke a C or C++ scanner to determine dependencies of the listed files, while DEPFILE can be used to provide a Ninja-specific .d dependency file. When printme() is called, execution enters the third implementation, which calls _printme(). Policies . Cache variables have more information attached to them than a normal variable, including a nominal type and a documentation string. There are two primary ways developers can do this, either from the cmake command line or using a GUI tool. This can help avoid problems with linking inconsistent standard libraries and other linkage issues. 19 4.6. Recommended Practices . STATIC Specifies a static library or archive. Normally, the binaryDir does not need to be specified. Paths may also contain generator expressions. This will cause the generator executable to be rebuilt because its source file was modified. 3.2. The project() Command Every CMake project should contain a project() command and it should appear after cmake_minimum_required() has been called. CMAKE_REQUIRED_INCLUDES Specifies directories to search for headers. For GLOB_RECURSE, on the other hand, directory names are not included by default but this can be controlled with the LIST_DIRECTORIES option. Consider the following scenario: • The onTheFly.cpp file initially does not exist. The example in the previous screenshot shows typical log output after the configure stage has been run and no changes were made to any of the cache variables. For example, to configure and then build a project with Ninja, one might use commands like this: cmake -G Ninja -DCMAKE_BUILD_TYPE:STRING=Debug ../source cmake --build . There have been wrong turns and many technical discussions on how to improve things from both a usability and robustness perspective. 366 27.1. ExternalProject . This can lead to overly complex expressions which ultimately obscure the logic and which can be difficult to debug. In particular, generator expressions can be used, a feature which becomes much more important when installing targets and creating packages. This can be used to limit which files are copied and to override the permissions just for the matched files. The implications are stronger for check_cxx_symbol_exists(), since it means only non-template functions or variables in the global namespace can be checked because any scoping (::) or template markers () would not be valid for a preprocessor symbol. The remove_directory command can strictly only be used with a single directory, whereas file(REMOVE_RECURSE...) can remove multiple items and both files and directories can be listed. The names may be returned with different upper/lower case than the way they were originally defined. It will also mean the project is likely to be better placed to adapt to new platform or operating system releases, which inevitably introduce new things for build systems to deal with. When comparing two version numbers, the major part is compared first. • The caller gets to choose the order in which the arguments are given. It will be used for the librarian or archiver tool. This can lead to some unexpected results. Using this approach, a simple scripted build might look something like this: mkdir build cd build cmake -G "Unix Makefiles" ../source cmake --build . The CMAKE_BUILD_TYPE cache variable can also be changed in the CMake GUI application instead of from the command line, but the end effect is the same. This is explicitly discouraged by the CMake documentation for precisely the above reasons. This would be common for libraries provided by the system. This can lead to unexpected results in a variety of situations for the unaware. 130 16. Some of the more common ones and a couple of utility expressions are listed here as a starting point for further reading: $ Evaluates to the build type. Once a command has been overridden more than once, its original implementation is no longer accessible. Each version component is expected to be an integer, but the comparison result is essentially undefined if this restriction does not hold. Thus, attaching additional compiler interface dependencies to a target can be done from any part of the project, but linker interface dependencies can only be done from the same directory scope in which a target is created. 18.1. Manipulating Paths One of the most basic parts of file handling is manipulating file names and paths. The APPEND keyword changes the behavior to append the value(s) to the existing one, forming a list, whereas the APPEND_STRING keyword takes the existing value and appends the new value(s) by concatenating the two as strings rather than as a list (see also the special note for inherited properties further below). This behavior is much closer to what developers would intuitively expect compared to the other directory property commands. set(allowableBuildTypes Debug Release Profile) # WARNING: This logic is not sufficient if(NOT CMAKE_BUILD_TYPE IN_LIST allowableBuildTypes) message(FATAL_ERROR "${CMAKE_BUILD_TYPE} is not a defined build type") endif() The default value for CMAKE_BUILD_TYPE is an empty string, so the above would cause a fatal error for both single and multi configuration generators unless the developer explicitly set it. If the destination is not an absolute path, it is interpreted as being relative to the current binary directory (i.e. CMAKE_CURRENT_BINARY_DIR). Versions before that lack substantial features that would make using many modern CMake techniques impossible. Prefer defining static libraries over object libraries. Rather than hard-coding the value in the CMakeLists.txt file as a normal variable, a cache variable can be used so that the developer can override the value without having to edit the CMakeLists.txt file. This chapter discusses the fundamentals of how that compiler and linker behavior is controlled. Many projects deem this to be an acceptable trade-off, while some developers consider this undesirable or even an anti-pattern. They are normally cache variables to allow the user to easily view and modify them, but they can also be set as regular CMake variables within the project's CMakeLists.txt files (something projects should aim to avoid). This family of modules all work in fundamentally the same way, writing a short amount of test code and then attempting to compile and possibly link and run the resultant executable to confirm whether what is being tested in the code is supported. The NO_SOURCE_PERMISSIONS and USE_SOURCE_PERMISSIONS options can be used to override these defaults, or the permissions can be explicitly specified with the FILE_PERMISSIONS and DIRECTORY_PERMISSIONS options. Put another way, upon entry into the child scope, it receives a copy of all of the variables defined in the calling scope at that point in time. There are also efforts underway to remove some of the restrictions associated with calling some commands from different directories, so add_subdirectory() is likely to become more flexible and be the more preferred of the two methods. Regular expressions are also well supported by the REGEX operation, with a few different variants available as determined by the second argument: string(REGEX MATCH regex outVar input [input...]) string(REGEX MATCHALL regex outVar input [input...]) string(REGEX REPLACE regex replaceWith outVar input [input...]) The regular expression to match, regex, can make use of typical basic regular expression syntax (see the CMake reference documentation for the full specification), although some common features such as negation are not supported. 205 21.1. Toolchain Files . ⑥ childVar is has been set in the child scope. 6.2.2. while() The other looping command offered by CMake is while(): while(condition) # ... The CMake documentation warns that these should only be used for PRIVATE items, since they would be carried through to other targets if defined as PUBLIC or INTERFACE and this may not always be safe. On the one hand, add_subdirectory() is simpler and does a better job of keeping directories relatively self contained because it creates its own scope. If simulator details have not been provided, the macros will instead expect a predetermined result to be provided through a set of variables and will not try to run anything. The following is a minimal, well-formed CMakeLists.txt file producing a basic executable. This is understandable, since the platform is one of the simplest things to understand and test, but using it instead of the more accurate constraint can unnecessarily limit the generator choices available to developers, or it may result in the wrong behavior entirely. They have their own dedicated commands and unique behaviors and are covered in depth in the next chapter. As a result, projects may still find it easier and more robust to prefer using the project-wide variables instead. Certain build tools may contain bugs which a project may later be affected by, so it can be useful to have alternative generators to fall back on until such bugs are fixed. The permission values are based on those supported by Unix systems: OWNER_READ OWNER_WRITE OWNER_EXECUTE GROUP_READ GROUP_WRITE GROUP_EXECUTE WORLD_READ WORLD_WRITE WORLD_EXECUTE SETUID SETGID If a particular permission is not understood on a given platform, it is simply ignored. ARGN Like ARGV, except this only contains arguments beyond the named ones (i.e. the optional, unnamed arguments). The use of this option should be avoided. Other higher level checks are also possible, as provided by CheckLanguage, CheckLibraryExists and the various CheckIncludeFile... modules. cmake_minimum_required(VERSION 3.0) project(PolicyExample) if(CMAKE_VERSION VERSION_GREATER 3.1) # Enable stronger checking of break() command usage cmake_policy(SET CMP0055 NEW) endif() Testing the CMAKE_VERSION variable is one way of determining whether a policy is available, but the if() command provides a more direct way using the if(POLICY...) form. CMake's modules have the benefit of a potentially large pool of developers and projects using them across a diverse set of platforms and situations, so they may offer a more compelling alternative to projects doing their own manual logic in many cases. The remove command accepts an optional -f flag which changes the behavior when an attempt is made to remove a file that does not exist. 240 23.1. Finding Files And Paths . The items specified in the link_libraries() command can be CMake targets, library names, full paths to libraries or even linker flags. In the case of Windows platforms, two properties should be set: IMPORTED_LOCATION should hold the location of the DLL and IMPORTED_IMPLIB should hold the location of the associated import library, which usually has a .lib file extension (the ..._ variants of these properties can also be set and will take precedence). In some situations, CMake may select a more recent standard due to compile feature requirements (discussed next). One last existence test is available in CMake 3.5 and later: if(value IN_LIST listVar) This expression will return true if the variable listVar contains the specified value, where value follows the usual variable-or-string rules but listVar must be the name of a list variable. It is initially empty when the target is created and it supports generator expressions. It can be used to include content from a file which has already been included in the build previously. With add_subdirectory(), both the source and the build trees' directory structures can be arbitrarily complex. A common but incorrect practice is to query the LOCATION target property to work out a target's output file name. All three commands provide the traditional behavior as implemented by most programming languages, but they also have added features specific to CMake. This differs from a public dependency in that library A doesn't require B internally, it only uses B in its interface. Not only does this command specify the minimum CMake version a project requires, it also sets CMake's behavior to match that of the version given. Developers should be comfortable with using imported targets, but understanding all the ins and outs of how they are defined is not usually necessary unless actually writing Find modules or manually creating config files for a package. This behavior is consistent with that of most operating systems' native copy commands, but it also means that the behavior is dependent on the state of the file system before the copy operation. For custom targets, PRE_LINK is not supported. 4.2. Defining Libraries Creating simple executables is a fundamental need of any build system. In the absence of any other options, the output of the last command is sent to the output of the CMake process itself but the standard error of every command is sent to the standard error stream of the CMake process. A universally supported alternative is presented in Section 17.5, "Platform Independent Commands" below. Projects should simply assume that the new implementation replaces the old one, with the old one considered to be no longer available. The following table summarizes the supported operators: 41 Numeric String Version numbers LESS STRLESS VERSION_LESS GREATER STRGREATER VERSION_GREATER EQUAL STREQUAL 1 LESS_EQUAL 1 STRLESS_EQUAL 1 VERSION_LESS_EQUAL 1 GREATER_EQUAL 1 VERSION_EQUAL 1 STRGREATER_EQUAL 1 VERSION_GREATER_EQUAL Only available with CMake 3.7 and later. Instead of storing the entire file's contents as a single string, this form stores it as a list 167 with each line being one list item. This shift in focus to a target-centric model has also led to a family of pseudo target types that facilitate expressing intertarget relationships more flexibly and accurately. By creating a target to represent the executable, other parts of the project can treat it just like it would any other executable target that the project built itself (with some restrictions). Every PUSH is required to eventually have a matching POP. 13.1. Build Type Basics The build type has the potential to affect almost everything about the build in one way or another. CMAKE_CURRENT_SOURCE_DIR The directory of the CMakeLists.txt file currently being processed by CMake. In almost all cases, CMake offers better alternatives to such approaches. Other targets then simply link to algo_all and the conditional linking of algo_beta is handled by the interface library. A little bit of time spent getting familiar with it will simplify working with more complex projects later. For this reason, it is more robust to always explicitly specify the target file name when copying a single file unless it is guaranteed that the destination is a directory that will already exist. message([mode] msg1 [msg2]...) If more than one msg is specified, they will be joined together into a single string without any separators. The REGEX option is a particularly useful way to extract only specific lines of interest from a file. Only if a specific bug fix is needed should a project specify a patch part. When using the Ninja generator, this has the effect of placing the command in the console pool. In particular, the various CMAKE_REQUIRED_... variables often need to be saved and restored. 6.2. Looping Another common need in many CMake projects is to perform some action on a list of items or for a range of values. A slightly reduced form of the command is as follows: configure_file(source destination [COPYONLY | @ONLY] [ESCAPE_QUOTES]) The source must be an existing file and can be an absolute or relative path, with the latter being relative to the current source directory (i.e. CMAKE_CURRENT_SOURCE_DIR). Some platforms may typically come with CMake already installed (eg most Linux distributions have CMake available through their package manager), but these versions are often quite old. The RANGE form also accepts just one argument like so: foreach(loopVar RANGE value) This is equivalent to foreach(loopVar RANGE 0 value), which means the loop body will execute (value + 1) times. These are heavily used by config files created during packaging or by Find module implementations (covered in Chapter 23, 133 Finding Things and Chapter 25, Installing), but have limited use outside of those contexts. If such headers are part of the project, SYSTEM is not typically an appropriate option to use. 176 19.2. Source Code Access To Version Details Once the version details are defined in the CMakeLists.txt file, a very common need is to make them available to source code compiled by the project. CMake's command mode also supports a very similar set of capabilities which can be used at build time rather than configure time: cmake cmake cmake -E -E -E -E rename source destination remove [-f] files... # Logical operators if(NOT expression) if(expression1 AND expression2) if(expression1 OR expression2) # Example with parentheses if(NOT (expression1 AND (expression2 OR expression3))) Following usual conventions, expressions inside parentheses are evaluated first, beginning with the innermost parentheses. This prevents project code from accidentally (or deliberately) replacing an official module with one of their own and changing the documented behavior. MACOSX_BUNDLE When present, this option directs CMake to build an app bundle when building on an Apple platform. cmake_policy(GET CMPxxxx outVar) The value stored in outVar will be OLD, NEW or empty. They provide a way to encode some logic which is not evaluated at configure time, the evaluation is instead delayed until the generation phase when the project files are being written. Another common use is to obtain a path related to some other target and use it as part of an include directory. Compare this to an ordinary name which CMake will treat as a library assumed to be provided by the system if it doesn't know of a target by that name. 119 MODULE Targets created with add_library(name MODULE ...). They control and provide information about everything from the flags used to compile source files through to the type and location of the built binaries and intermediate files. add_library(myHeaderOnlyToolkit INTERFACE) target_include_directories(myHeaderOnlyToolkit INTERFACE /some/path/include) target_compile_definitions(myHeaderOnlyToolkit INTERFACE COOL_FEATURE=1 $) add_executable(myApp ...) target_link_libraries(myApp PRIVATE myHeaderOnlyToolkit) In the above example, the myApp target links against the myHeaderOnlyToolkit interface library. The second case demonstrates how combining bracket syntax with embedded content can be a particularly convenient way of defining file contents inline, even when generator expressions and quoting are involved. The build should be predictable, reliable and easy to set up, but if it relies on environment variables being set for things to work correctly, this can be a point of frustration for new developers as they wrestle to get their build environment set up. False values could be an empty string, an error message, etc. These cached details are stored in the build directory in the CMakeCache.txt file. Either of the INPUT or CONTENT options must be present, but not both. ⑦ When processing returns to the parent scope, myVar still has the value from before the call to add_subdirectory(). • Build the myExe target again, which this time results in the following sequence: ◦ The generator target is brought up to date. They define the content to be written to the specified output file. If a separate toolchain file is available, that can be used to customize not just the compilers but also the target environment, compiler flags and various other things. In such cases, no configuration-specific compiler or linker flags are used, which often simply results in invoking the 105 compiler and linker with minimal flags and hence the behavior is determined by the compiler's and linker's own default behavior. When splitting up a library, define an interface library with the old target name and have it define link dependencies to the new split out libraries. The following examples show how to make use of generator expressions to customize the contents and file names depending on the build type. Setting Up A Project Without a build system, a project is just a collection of files. This reinforces better habits which will be important when working on more complex multi-target projects. Those policies introduced at the specified CMake version or earlier are reset to NEW. The console log almost always shows the -- Generating done message immediately after the configure step completes, so this is understandable. For the above example, the implementation of dangerous() could be changed to use foreach(arg IN ITEMS ${ARGN}) instead. A Minimal Project All CMake projects start with a file called CMakeLists.txt and it is expected to be placed at the top of the source tree. Importantly, it does not mean set the variable in both the parent and the current scope. The developer selects the type of project file to be created by choosing a particular project file generator. This allows different parts of a project hierarchy to pull in the same packages and modules with different settings, yet not interfere with each other. # See details in Chapter 23: Finding Things find_package(BagOfBeans REQUIRED) # Define an executable that links to the imported # library from the installed package add_executable(eatLunch main.cpp ...) target_link_libraries(eatLunch PRIVATE BagOfBeans::myRealThings) If at some point the above project wanted to incorporate the BagOfBeans project directly into its own build instead of finding an installed package, it could do so without changing its linking relationship because the BagOfBeans project provided an alias for the namespaced name: # Add BagOfBeans directly to this project, making # all of its targets directly available add_subdirectory(BagOfBeans) # Same definition of linking relationship still works add_executable(eatLunch main.cpp ...) target_link_libraries(eatLunch PRIVATE BagOfBeans::myRealThings) Another important aspect of names having a double-colon (::) is that CMake will always treat them as the name of an alias or imported target. If a target is listed, it will be brought up to date any time the custom command's output 147 files are required to be brought up to date. The optional LANGUAGES argument defines the programming languages that should be enabled for the project. The only other difference between COPY and INSTALL apart from the default permissions is that the INSTALL form prints status messages for each copied item, whereas COPY does not. 367 27.1.2. Step Management . 259 24. A variable can be unset either by calling unset() or by calling set() with no value for the named variable. # Define an object library add_library(objLib OBJECT src1.cpp src2.cpp) # Define two executables which each have their own source # file as well as the object files from objLib add_executable(app1 app1.cpp $) add_executable(app2 app2.cpp $) In the above example, no separate library is created for objLib, but the src1.cpp and src2.cpp source files are still only compiled once.) These commands find all files whose names match any of the provided expressions, which can be thought of as simplified regular expressions. That new scope acts like a child of the calling scope and there are a number of effects: • All variables defined in the calling scope will be visible to the child scope and the child scope can read their values like any other variable. Developers should think carefully before relying on the more general information generator expressions as a way to solve a problem. The source and destination must be the same type, i.e. both files or both directories. The following shows how to safely put these command mode commands together: add_custom_target(copyOne COMMAND ${CMAKE_COMMAND} -E copy a.txt output/textfiles/a.txt) add_custom_target(copyTwo COMMAND ${CMAKE_COMMAND} -E make_directory output/textfiles COMMAND ${CMAKE_COMMAND} -E copy a.txt b.txt output/textfiles) The copy command will always copy the source to the destination, even if the destination is already identical to the source. Conveniently, single configuration generators will ignore any build-time specification and multi configuration generators will ignore the CMAKE_BUILD_TYPE variable, so by specifying both, a script can account for both cases. Without 120 the FORCE keyword, the set() command only updates a cache variable if it is not already defined. In particular, longlived projects may find it a useful stepping stone in updating their code base to more recent language features while they still need to support older compilers on some platforms. While some platforms have a dominant IDE environment that simplifies some aspects of this (e.g. Xcode and Visual Studio), projects that need to support multiple platforms cannot always make use of their features. If the configure and generation time is non-trivial and the source file is being modified frequently, this can be a source of frustration for developers. 59 8.2. Argument Handling Essentials The argument handling of functions and macros is the same except for one very important difference. The TestBigEndian module provides the test_big_endian() macro which compiles a small test program to determine the endianness of the target. An example should help illustrate the problem. This can be exploited to do things like abstract away completely different library structures on different platforms, switch library implementations based on some condition (variables, generator expressions, etc.), provide an old library target name where the library structure has been refactored (e.g. split up into separate libraries) and so on. Projects should always set all three variables together to make clear how the language standard requirements should be enforced and whether compiler extensions are permitted. A definition has the form VAR or VAR=VALUE, which CMake will convert to the appropriate form for the compiler being used (typically -DVAR... or /DVAR...). There are also a variety of variables and properties which could simplify or replace the need for generator expressions altogether. 171 INACTIVITY_TIMEOUT seconds This is a more specific kind of timeout. On the other hand, spending all one's time learning the theory about something without getting hands-on makes for a rather boring experience and often leads to an overly idealistic understanding. This is why the source file in the above example used "@" rather than @ as the filename to write the output to. Cache variables defined on the command line have an empty docstring. The expressions in if() and elseif() commands can take a variety of different forms. Consider first whether CMake already provides a dedicated facility 84 to achieve the same result. This is especially true when specifying multiple commands, since it makes each command use a consistent form. • Being keyword-based, the calling site has improved readability, since the arguments essentially become self-documenting. The flexibility to use any number of = characters between the [and] markers also means embedded square brackets do not get misinterpreted as markers. A common approach is to allow a variable name to be passed in as a function argument so that the caller is still in control of the name of variables where function results are set. Projects often need to extract file names, file suffixes, etc. By specifying NEW or OLD, a project tells CMake to use the new or old behavior for that particular policy. Plain library name If just the name of the library is given with no path, the linker command will search for that library (e.g. foo becomes -lfoo or foo.lib, depending on the platform). Executables, libraries, test output and packages are all created within the build directory. For example: function(print_me) message("Hello from inside a function") message("All done") endfunction() # Called like so: print_me() As shown above, the name argument defines the name used to call the function or macro and it should only contain letters, numbers and underscores. 353 26.4.5. DragNDrop . Being written and packaged as ordinary CMake code and therefore being human readable, modules can also be a useful resource for learning more about how to get things done in CMake. Some developers may find object libraries more natural if coming from a background where nonCMake projects defined their targets based on sources or object files rather than a related set of static libraries. By establishing the habit of using CMAKE_CURRENT_LIST_DIR early, it is much easier to switch between add_subdirectory() and include() as a project grows in complexity and to move entire directories to restructure a project. This can be more convenient for some builds because it can avoid the build time cost of creating a static library or the run time cost of symbol resolution for a dynamic library, yet still avoid having to compile the same sources multiple times. For example: set(ENV{PATH} "$ENV{PATH}:/opt/myDir") Note, however, that setting an environment variable like this only affects the currently running CMake instance. Alternatively, the binaryDir can be explicitly specified as either an absolute or relative path, with the latter being evaluated relative to the current binary directory (discussed in more detail shortly). Later chapters also discuss the use of more complex source directory hierarchies. Different target types can be defined that act as a reference to other entities rather than being built themselves. The benefits should be weighed against the increased complexity and potential reduction in readability and less natural style for most developers. In comparison, the following example requests a later C++ version and enables compiler extensions, resulting in a GCC/Clang compiler flag like -std=gnu++14 instead. target_link_libraries(targetName lib [lib...] [lib [lib...]]) Similar to the previous old-style form, this one is a pre-cursor to the PRIVATE and PUBLIC keyword versions of the newer form. Chapter 7, Using Subdirectories and Chapter 8, Functions And Macros introduce the situations where local scopes arise and show how the PARENT_SCOPE keyword is used to promote the visibility of a variable into the enclosing scope. 17.1. Custom Targets Library and executable targets are not the only kinds of targets CMake supports. Using a toolchain file is typical when cross-compiling, which is covered in detail in Chapter 21, Toolchains And Cross 30 Compiling. 218 22. Packaging . CMake provides the execute_process() command for running tasks like these during the configure stage: execute_process(COMMAND command1 [args1...] [COMMAND command2 [args2...]] [WORKING_DIRECTORY directory] [RESULT_VARIABLE resultVar] [RESULTS_VARIABLE resultsVar] [OUTPUT_VARIABLE outputVar] [ERROR_VARIABLE errorVar] [OUTPUT_STRIP_TRAILING_WHITESPACE] [ERROR_STRIP_TRAILING_WHITESPACE] [INPUT_FILE inFile] [OUTPUT_FILE outFile] [ERROR_FILE errorFile] [OUTPUT_QUIET] [ERROR_QUIET] [TIMEOUT seconds]) Similar to add_custom_command() and add_custom_target(), one or more COMMAND sections specify the tasks to be executed and the WORKING_DIRECTORY option can be used to control where those commands are run. This is where the OUTPUT form of add_custom_command() can be used. The output of the last command in the set of commands can be captured by specifying the name of a variable to store it in with the OUTPUT_VARIABLE option. The executable name can be customized with target properties, a CMake feature introduced in Chapter 9, Properties. The --target option can be used to tell the build tool what to build, or if omitted, the default target will be built. need to be carried forward to anything using the headers. It also tends to allow IDE tools to recognize the file type and apply

CMake syntax highlighting. which may not be available by default on all platforms. C++ keywords like final, constexpr, noexcept and more can all potentially be used if available or omitted if not supported by the compiler and still produce valid and correct code. Specifying Version Details . Note also that copy_directory copies the contents of the source directories into the destination, not the source directories themselves. This is particularly true when developing for Apple platforms where tools like Xcode and its SDKs change rapidly and where app store requirements evolve over time. PRE_LINK The commands will be run after sources are compiled, but before they are linked. This leads to greater consistency of the code. The second form retrieves the value of a variable, which may not seem all that useful, but it provides a means of obtaining a variable's value from a different directory scope other than the current one (when the DIRECTORY argument is used). This forces projects to manually apply such dependencies themselves, which increases the opportunity for errors and omissions. Care needs to be taken, however, to avoid setting CMAKE_CONFIGURATION_TYPES if it is not already defined. These special library types support only the INTERFACE_... target properties and not the non-INTERFACE_... properties. The -P option to the cmake command puts CMake into script processing mode: cmake [options] -P filename The filename argument is the name of the CMake script file to execute. 343 26.4. Package Generators . I am indeed a very fortunate human being. 2 Part I: Fundamentals Attempting to use any tool before understanding at least the basics of what it does and how it is meant to be used is most likely going to result in frustration. 105 13.3. Custom Build Types . CMake treats all variables as strings. set(base /usr/share) file(GLOB_RECURSE images RELATIVE ${base} ${base}/*/*.png) The above will find all images below /usr/share and include the path to those images, except with the /usr/share part stripped off. As a general guide, the PROJECT_... variables can be useful for generic code (especially modules) as a way to define sensible defaults for things like packaging or documentation details. But because they add object files rather than actual libraries, their transitive nature is more restricted to prevent object files from being added multiple times to consuming targets. Any task not already directly supported by CMake can be implemented using commands provided by the host operating system instead. Each of the three target properties above has a related target property with the same name, only with INTERFACE_ prepended. For projects with many files spread across multiple directories, there are better ways to collect the set of source files which do not suffer from such problems. Later chapters also highlight other common errors and misuses of regular variables that share the same name as cache variables. Try to avoid relying on environment variables being defined, apart from perhaps the ubiquitous PATH or similar operating system level variables. These allow compiler/linker flags and header search paths to also carry through from one target to another when they are connected by target_link_libraries(). For example, a Unix shell script might be copied to the current binary directory as follows: file(COPY whoami.sh DESTINATION . In most projects, specifying the patch and tweak parts is not necessary, since new features typically only appear in minor version updates (this is the official CMake behavior from version 3.0 onward). 207 21.3. Tool Selection . It can be as simple as a few lines or it can be quite complex and pull in more files from other directories. They may run the code under a simulator if the necessary details have been provided, which would likely slow down the CMake stage considerably. A surprisingly powerful pair of subcommands provide the ability to download files from and upload files to a URL. 59 8.1. The Basics . 110 Chapter 14. Furthermore, the policy settings for the function implementation are taken from the scope in which the function was defined, not the one from which it is called. The main situation where the current policy setting may need to be read is in a module file, which may be one provided by CMake itself or by the project. 98 If CMake detects that the project is doing something that either relies on the old behavior, conflicts with the new behavior or whose behavior is ambiguous, it may warn if the relevant policy is unset. The cmake-gui user interface is shown in the figure below. The main highlights for this edition include: The material covering Android has been rewritten. Project Organization. In such cases, substitution can be limited to only the @ form with the @ONLY keyword: 160 CMakeLists.txt set(USER_FILE whoami.txt) configure_file(whoami.sh.in whoami.sh @ONLY) whoami.sh.in #!/bin/sh echo ${USER} > "@" whoami.sh #!/bin/sh echo ${USER} > "whoami.txt" Substitution can also be disabled entirely with the COPYONLY keyword. There could even be multiple build trees being used with the same source tree. The destination directory structure is created as necessary. Once again though, quoted values have ambiguous behavior similar to that in basic unary expressions. add_custom_command(OUTPUT output1 [output2...] COMMAND command1 [args1...] [COMMAND command2 [args2...]] [WORKING_DIRECTORY dir] [BYPRODUCTS files...] [COMMENT comment] [VERBATIM] [USES_TERMINAL] [APPEND] [DEPENDS [depends1...] [MAIN_DEPENDENCY depend] [IMPLICIT_DEPENDS depend1 [depend2...]] [DEPFILE depfile]) Instead of specifying a target and pre/post build stage, this form requires one or more output file names to be given after the OUTPUT keyword. Note that from CMake version 3.3, the linker command always uses the full path specified, but prior to version 3.3, there were some situations where CMake may ask the linker to search for the library instead (e.g. replace /usr/lib/libfoo.so with -lfoo). The CACHE keyword has the same meaning as the other forms of get_filename_component(). Policy names are usually of the form CMPxxxx, where the xxxx part is always a four digit number. 17.5. Platform Independent Commands The add_custom_command(), add_custom_target() and execute_process() commands provide projects with a great deal of freedom. 3.3. Building A Basic Executable To complete our minimal example, the add_executable() command tells CMake to create an executable from a set of source files. Developers who need to manually write config packages should refer to the CMake reference documentation to understand the other IMPORTED_... target properties which may be relevant to their situation. In order to force the custom command to be re-run if the generator target is rebuilt, an explicit dependency has to be specified rather than relying on the dependency CMake automatically creates. As noted in the previous section, different parts of a project may include the same file from multiple places. The official CMake packages can be downloaded and unpacked to a directory on the developer's machine without interfering with any system-wide CMake install. 228 22.5. Build Settings . Without the COPYONLY or @ONLY options, anything in the source file that looks like a use of a CMake variable (i.e. has the form ${someVar}) will be replaced by the value of that variable. The path from author to reader is now so much shorter and simpler than even just a few years ago, which brings it within reach of more potential writers and delivers benefits to the wider community. The other two dependency-related options, IMPLICIT_DEPENDS and DEPFILE, are not supported by most project generators. If compiler flags need to be manipulated at the individual source file level, target properties are not granular enough. This can be useful to help diagnose problems when a download or upload fails. CMake then takes care of managing the dependencies throughout the chain of libraries linked in this fashion. To delete directories and all of their contents, use the REMOVE_RECURSE form instead. For a single unquoted constant, the rules are as follows: • If value is an unquoted constant with value 1, ON, YES, TRUE, Y or a non-zero number, it is treated as true. CMake is particularly flexible in that it is also possible to use this form recursively or to specify the name of another variable to set. variable_watch(myVar [command]) 32 For the vast majority of cases, simply listing the variable to be watched without the optional command is sufficient, as it logs all accesses to the nominated variable. The second implementation of printme is encountered. _STANDARD_REQUIRED While the _STANDARD property specifies the language standard the project wants, 124 _STANDARD_REQUIRED determines whether that language standard is treated as a minimum requirement or as just a "use if available" guideline. 24 5.4. Manipulating Cache Variables . For example, new C++ STL types, functions, etc. FATAL_ERROR Denotes a hard error. In the material below, this output variable will generally be named outVar. 62 8.4. Scope . The potential confusion around quoted values means it is generally advisable to avoid using quoted arguments with the if(something) form. Note that the default settings for RelWithDebInfo will disable assertions. This combination usually arises when an INTERFACE library is exported or installed for use outside of the project. 169 18.5. Downloading And Uploading . A number of people generously agreed to review the material in this book, without whom both technical accuracy and readability would have suffered. This first part of the book follows a logical progression through CMake's more fundamental features and concepts and is structured to enable the reader to immediately experiment and to do increasingly useful things with each chapter. Strings of the form @ are also substituted in the same way. 102 Chapter 13. The same stream cannot be captured in a variable and sent to a file at the same time. 27 5.4.2. CMake GUI Tools Setting cache variables via the command line is an essential part of automated build scripts and anything else driving CMake via the cmake command. The following table shows the automatically populated version variables when the VERSION keyword is used with the project() command: 175 PROJECT_VERSION projectName_VERSION PROJECT_VERSION_MAJOR projectName_VERSION_MAJOR PROJECT_VERSION_MINOR projectName_VERSION_MINOR PROJECT_VERSION_PATCH projectName_VERSION_PATCH PROJECT_VERSION_TWEAK projectName_VERSION_TWEAK The two sets of variables serve slightly different purposes.) 169 file(GLOB_RECURSE outVar [LIST_DIRECTORIES true|false] [RELATIVE path] [FOLLOW_SYMLINKS] [CONFIGURE_DEPENDS] # Requires CMake 3.12 or later expressions... Projects should confirm that for the minimum CMake version they require, the properties being modified do indeed support the generator expressions used. GUI tools present a file dialog to the user for modifying the variable's value. CMake versions prior to 3.0 do not support the LANGUAGES keyword, but languages can still be specified after the project name using the older form of the command like so: project(myProj C CXX) New projects are encouraged to specify a minimum CMake version of at least 3.0 and use the new form with the LANGUAGES keyword instead. set(myVar "goes here") set(multiLine "First line ${myVar} Second line with a \"quoted\" word") If using CMake 3.0 or later, an alternative to quotes is to use the lua-inspired bracket syntax where the start of the content is marked by [=[and the end with]=]. Where a version string is provided with some parts omitted (the tweak part is often left out, for example), the corresponding variables are left empty. Each feature must be one of the features supported by the underlying compiler. This enters the second implementation which also calls _printme(), but _printme() points back at the second implementation again and infinite recursion results. A related error is to assume a particular build output directory structure in custom commands (see Chapter 17, Custom Tasks). The primary workhorse is the -D option, which is used to define the value of a cache variable. 9.5. Source Properties CMake also supports properties on individual source files. If sourceDir contains any path components, these will be mirrored in the binaryDir created by CMake. Once these checks have passed, CMake sets up a number of variables and properties which control the build for the enabled languages. have to be manually carried across to the targets they are added to. It is ignored for targets being built as a static library. This chapter covers one of the most fundamental parts of CMake, namely the use of variables. Instead of capturing the output and error streams in a variable, they can be sent to files. Note the /*/ in the expression to allow any directory below the base point to be matched. Furthermore, note how the embedded contents contain a pair of square brackets. This will allow CMake to treat the imported target just like a regular library target of the named type in various situations. This was fairly common practice in the pre CMake 3.0 era, but with the focus shifting to a target-centric model with CMake 3.0 and later, such examples are no longer a good model to follow. For many larger projects, the ability to create and work with libraries is also essential to keep the project manageable. 35 5.8. Math. The docstring should be short and consist of plain text (i.e. no HTML markup, etc.). 170 The file(GLOB) and file(GLOB_RECURSE) commands are some of the most misused parts of CMake. Typical situations where the ARG... variables are used include supporting optional arguments and implementing a command which can take an arbitrary number of items to be processed. 118 CMake provides a set of variables that specify compiler and linker flags to be merged with those provided by the various directory, target and source file properties. • Generating a file from string contents. Once a generator has been selected for a project, it cannot be changed without deleting the cache and starting again, which can be done from the File menu if required. This is achieved by setting a cache variable's STRINGS property (covered in detail in Section 9.6, "Cache Variable Properties", but shown here for convenience): set(trafficLight Green CACHE STRING "Status of something") set_property(CACHE trafficLight PROPERTY STRINGS Red Orange Green) In the above, the trafficLight cache variable will initially have the value Green. FULL_DOCS Retrieves the full documentation for the named property. set_property(entitySpecific [APPEND] [APPEND_STRING] PROPERTY propName [value1 [value2 [...]]]) entitySpecific defines the entity whose property is being set. If the OUTPUT_STRIP_TRAILING_WHITESPACE option is present, any trailing whitespace will be omitted from the content stored in the output variable, while the ERROR_STRIP_TRAILING_WHITESPACE option does a similar thing for the content stored in the error variable. If the Ninja generator is being used, this option is required if another target depends on any of the files created as a by-product of running this set of custom commands. Working with variables is covered in detail in Chapter 5, Variables, but for now, one way to set this variable is by including a -D option on the cmake command line like so: cmake -DBUILD_SHARED_LIBS=YES /path/to/source It could be set in the CMakeLists.txt file instead with the following placed before any add_library() commands, but that would then require developers to modify it if they wanted to change it (i.e. it would be less flexible): set(BUILD_SHARED_LIBS YES) Just as for executables, library targets can also be defined to refer to some existing binary or target rather than being built by the project. 50 7.1. add_subdirectory() . The quality does, however, vary from one module to another. These details are covered in the next few sections, but before launching into those discussions, it is essential to understand how to select the build type and how to avoid some common problems. Any number of = characters can appear between the square brackets, including none at all, but the same number of = characters must be used at the start and the end. I would also like to express my gratitude to the people behind Asciidoctor, the software used to compile and prepare this book. 194 20.5.2. Specifying Individual Symbol Visibilities . If a dependency exists on a file rather than a target, the DEPENDS keyword can be used to specify that relationship as part of the add_custom_target() call directly. For this scenario, optional support for specific language features may help reduce compiler warnings and catch coding errors when using more modern compilers. Even if that variable existed in the calling scope, the calling scope's variable is left unchanged. These commands require the type of entity to be specified as command arguments along with some entity-specific information. 109 13.4. Recommended Practices Developers should not assume a particular CMake generator is being used to build their project. If the library file changes, CMake will detect that change and re-link the target. Because it evaluates generator expressions, it cannot write out the files immediately. 13.2. Common Errors Note how for single configuration generators, the build type is specified at configure time, whereas for multi configuration generators, the build type is specified at build time. Before jumping in and getting their hands dirty with CMake, developers will first need to ensure CMake is installed on their system. Setting policies to match a particular CMake release's behavior makes the project easier to understand and update, whereas changes to individual policies can be harder to trace through multiple directory levels, especially because of their interaction with version-level policy changes where they are always reset. While this may often be similar to the Debug build type's behavior, it is by no means guaranteed. Also consider that the project may one day become a sub-part of some larger project, so a name beginning with the project name or something closely associated with the project may be desirable. All three comparison categories support the same set of operations, but the OPERATOR names are different for each category. The distinction between configuring and generating doesn't matter so much for basic CMake usage, but in later chapters the separation of configuration and generation becomes important. Another variable which may sometimes be defined for a custom build type is CMAKE__POSTFIX. This will prevent other projects from being forced to require a more recent version than their target environment typically allows or provides. If not called from inside a function, return() ends processing of the current file regardless of whether it was brought in via include() or add_subdirectory(). ARGV This is a list variable containing each of the arguments passed to the function, including both the named arguments and any additional unnamed arguments that were given. Even parts of CMake itself did this prior to 3.11. If provided, the initialValue must conform to one of the boolean values accepted by the set() command. Be aware that only the Visual Studio generator supports this option and only for Visual Studio 7 or later. Consider the target_link_libraries() command: 62 target_link_libraries(targetName item1 [item2 ...] [item3 [item4 ...]] ... GUI tools use a checkbox or similar to represent the variable. This can be useful in cross-compiling situations, especially where a build may support multiple platforms (e.g. device and simulator builds). This also has the benefit of making the project well placed to take advantage of any new generator type in the future. One could simply set a variable near the start of the CMakeLists.txt file to record a version number in whatever form is needed like so: cmake_minimum_required(VERSION 3.0) project(FooBar) set(FooBar_VERSION 2.4.7) If individual components need to be extracted, a slightly more involved set of variables may need to be defined, one example of which may look something like this: cmake_minimum_required(VERSION 3.0) project(FooBar) set(FooBar_VERSION_MAJOR 2) set(FooBar_VERSION_MINOR 4) set(FooBar_VERSION_PATCH 7) set(FooBar_VERSION ${FooBar_VERSION_MAJOR}.${FooBar_VERSION_MINOR}.${FooBar_VERSION_PATCH}) Different projects may use different conventions for the naming of variables. Compared to other languages, variable scope is a little more flexible in CMake, but for now, in the simple examples in this chapter, consider the scope of a variable as being global. These do, however, have the potential for undesirable negative impacts on the build behavior of a project. It now focuses on using NDK r23 and CMake 3.21 as minimum versions. If using the output or error variables' contents for any sort of string comparison, a common problem is failing to account for trailing whitespace, so its removal is often desirable. If the ALL option is not provided, then the target is only ever built if it is explicitly requested or if building some other target that depends on it. That earlier discussion focused on the dependency relationships between targets, but following the above discussion of target properties, the exact effects of those keywords can now be made more precise. Depending on the mix of digits and non-digits, the result of the expression may be true or false. 101 Part II: Builds In Depth In the preceding chapters, the most fundamental aspects of CMake were progressively introduced. If the downloaded file's hash cannot be known in advance, then the TLS_VERIFY option is highly recommended to ensure the integrity of the content. Where possible, it is better to use the above method to give the caller the control over variable names being set or modified. Before CMake 3.11, alias libraries could not be created for imported targets, but as with other changes made for imported targets in CMake 3.11, this restriction was relaxed and it has become possible to create aliases for globally visible imported targets. In such cases, projects can set a variable named CMAKE_GENERATOR_NO_COMPILER_ENV to a boolean true value and then CMake will not set CC and CXX for any execute_process() call, even the initial invocation. They can be used to perform conditional logic, output strings providing information about various aspects of the build like directories, names of things, platform details and more. Not only is this easy to understand and use, it also has the advantage that it is relatively straightforward to ensure that the same standard is used throughout a project. This would seem to present a problem if the CMakeLists.txt file wants to do something like copy a file to the same directory as the final executable for a given target, since the location of that directory depends on which configuration is being built. Similar to the situation for the if() and endif() commands, in very early versions of CMake (i.e. prior to 2.8.0), all forms of the foreach() command required that the loopVar also be specified as an argument to endforeach(). In short, target properties are where most of the details about how to actually turn source files into binaries are collected and applied. For example: mkdir build cd build cmake -G Ninja -DCMAKE_BUILD_TYPE=Release ../source cmake --build . Another aspect of this issue is that developers may add their own types to the CMAKE_CONFIGURATION_TYPES cache variable and/or remove some of the ones they are not interested in. 397 28.5. Defining Targets . Using the default native compilers is the usual choice for ordinary desktop development and selecting that option requires no further details. The greatest difficulty is typically experienced by projects that need to support older platforms where the system-provided version of CMake may be quite old. 87 11.2. Endianness . Be forewarned though that the quality of the documentation does vary from module to module. This is a limitation of how the dependency details are handled in the Makefile setup, where testing whether each individual source's compiler flags have changed brings with it a prohibitively big performance hit, so the CMake developers chose to implement the dependency at the target level instead. An item can be preceded by the keyword general to indicate that it applies to all build types, but since general is the default anyway, there is little benefit to doing so. The BUILD_SHARED_LIBS variable can be used to change the default in one place instead of having to modify every call to add_library(). That directory must have its own CMakeLists.txt file which will be processed at the point where add_subdirectory() is called and a corresponding directory will be created for it in the project's build tree. Tests are discussed in detail in Chapter 24, Testing. 14.4. Recommended Practices This chapter has covered areas of CMake which have undergone some of the most significant improvements since earlier versions. Projects supporting the Windows platform can also obtain details about PDB files for a given target. Note that DEPENDS should not be used for target dependencies, only file dependencies. Without either keyword, the variables will only be marked as advanced if they don't already have an advanced/non-advanced state set. • In general, prefer to avoid the directory property commands. Macros can be handled the same way as functions, specifying the names of variables to be set by passing them in as arguments. • Before CMake 3.1, if the value of the string matched the name of an existing variable, then the quoted string is effectively replaced by that variable name (unquoted) and the test is then repeated. If working with fast-moving platforms such as iOS, quite recent versions of CMake may be needed in order to support the latest OS releases, etc. It is also permitted for a project to create new properties apart from those 70 already defined by CMake. else() # commands ... If required, the BEFORE keyword can be used to prepend the listed directories to existing contents of the target properties instead. There are times, however, where it is desirable for a variable change made in an added directory to be visible to the caller. 151 17.6. Combining The Different Approaches . When OFF, if the requested standard is not supported by the compiler, CMake will decay the request to an earlier standard rather than halting with an error. 27 5.4.2. CMake GUI Tools . Section 28.5.1, "Target Sources" presents some alternative strategies which not only avoid these problems, they also encourage a more modular and self-contained directory structure. As for add_custom_target(), always use an absolute path if listing a file for DEPENDS to avoid ambiguous legacy behavior. foo?.txt Files like foo2.txt, fooB.txt, etc. This in turn has a strong impact on other areas such as symbol visibility (discussed in detail in Section 20.5, "Symbol Visibility"). The most general of these is the TARGET_FILE set of generator expressions: TARGET_FILE This will yield the absolute path and file name of the target's binary, including any file prefix and suffix if relevant for the platform (e.g .exe, .dylib). Developers should become familiar with interface libraries in particular, as they open up a range of techniques for capturing and expressing relationships without needing to create or refer to a physical file. One of the few differences compared to a regular target is that imported targets cannot be installed, a topic covered in Chapter 25, Installing. New projects should prefer to use INTERFACE_LINK_LIBRARIES instead. Another option is to use different project generators for the different build directories, such as Unix Makefiles and Xcode. 219 22.1. CMake Generator Selection . • A part of the project is not able to be modified (e.g. it might come from an external read-only code repository) and it relies on old behavior which has been changed in newer CMake versions. 4.3. Linking Targets When considering the targets that make up a project, developers are typically used to thinking in terms of library A needing library B, so A is linked to B. These can be used to execute arbitrary commands and define custom build rules, allowing projects to implement just about any sort of behavior needed. There is a special case for handling values initially declared without a type on the cmake command line. The effect of these keywords is to further refine when the item following it should be included based on whether or not the build is configured as a debug build (see Chapter 13, Build Type). Adding the GLOBAL keyword when creating the imported library achieves this, but the project may not be in control of the command that does the creation. Dependencies can be manually specified with the DEPENDS option. Occasionally, however, a project may need to explicitly convert between CMake and native paths, such as when working with custom commands and needing to pass a path to a script which requires native paths. In such cases, CMake will retrieve the normal variable's value rather than the cache variable when using ${myVar}, or put another way, normal variables take precedence over cache variables. This makes it easy to add multiple paths in a natural, progressive manner. The second argument is always the list to operate on and it must be a variable (i.e. passing a raw list like a;b;c is not permitted). The form acting at the coarsest level is a close relative to cmake_minimum_required(): cmake_policy(VERSION major[.minor[.patch[.tweak]]]) 96 In this form, the command changes CMake's behavior to match that of the specified version. # This could be in an external file brought in # by an include() call rather than in the same # file as the lines further below. The MATCH operation finds just the first match and stores it in outVar. In particular, if source properties are used to modify the compiler flags for specific source files rather than for a whole target, changing the source's compiler flags will still result in all of the target's sources being rebuilt, not just the affected source file. 5 Chapter 2. 19 4.5. Old-style CMake . Those who have already been using CMake for some time may find the topics relatively familiar, but the material also includes hard-won knowledge from real world projects and interaction with the CMake community. Neither is right or wrong, although a common mistake is to use the property-specific commands to replace a property value instead of appending to it. 46 6.2.1. foreach() . Problems can arise, however, when a project tries to force updating the cache variable instead of a local variable. Strongly prefer to set a minimum CMake version of at least 3.1 to disable the old behavior that allowed implicit conversion of quoted string values to variable names. The above three TARGET_FILE expressions are especially useful when defining custom build rules for copying files around in post build steps (see Section 17.2, "Adding Build Steps To An Existing Target"). It is also possible to remove variables from the cache with the -U option, which can be repeated as necessary to remove more than one variable. For most developers, however, imported targets are generally created for them behind the scenes and they appear to act more or less like regular targets. The behavior is essentially as though the variable was being assigned using the set() command with the CACHE and FORCE options. No assumptions are made about the contents of the strings, but be mindful of the potential for the variable/string substitution situation described earlier. Without bracket syntax, CMake would see the ${USER} part and treat it as a CMake variable substitution, but because bracket syntax performs no such substitution, it is left as is. When the project is built, an executable will be created in the build directory with a platform-dependent name, the default name being based on the target name. That being the case, any return() statement from a macro will actually be returning from the scope of whatever called the macro, not from the macro itself. Most of the output to the console log is produced during this stage, including any content from message() commands. It focuses purely on providing a simple way to obtain the value of a target property and is basically just a shorthand version of the generic command. The CMAKE_COMMAND variable is populated by CMake and it contains the full path to the cmake executable being used in the main build. This allows switching between build types without forcing a complete recompile each time. The specified message is logged just before running the commands, so if the commands silently fail for some reason, the comment can be a useful marker to indicate where the build failed. In addition, consider adding an alias namespace::... target for each target that is not private to the project (i.e. every target that may end up being installed or packaged). The required flags vary from compiler to compiler, so it is more robust, more maintainable and more convenient to use the features CMake provides and allow it to populate the flags appropriately. file(RENAME source destination) file(REMOVE files...) file(REMOVE_RECURSE filesOrDirs...) file(MAKE_DIRECTORY dirs...) The RENAME form renames a file or directory, silently replacing the destination if it already exists. The following table summarizes what the various keyword combinations support: 136 Keywords Visibility Imported Location Set Interface Properties Installable INTERFACE Global Prohibited Any method Yes IMPORTED * Local Required * No * No * No Restricted IMPORTED GLOBAL Global Required Restricted INTERFACE IMPORTED Local Prohibited Restricted INTERFACE IMPORTED GLOBAL Global Prohibited Restricted The various target_...() commands can be used to set INTERFACE_... properties only if using CMake 3.11 or later. 5.2. Environment Variables CMake also allows the value of environment variables to be retrieved and set using a modified form of the normal variable notation. If the file calling include() is itself inside CMake's own internal module directory, then the internal module directory will be searched first before consulting CMAKE_MODULE_PATH. The CMakeLists.txt file will also more clearly express the intent, since human readable variables and properties are used instead of often cryptic raw compiler and linker flags. The debug, optimized and general keywords should be avoided for new projects as there are clearer, more flexible and more robust ways to achieve the same thing with today's CMake features. Furthermore, the Xcode project file format does not support configuration specific source file properties at all, so if targeting Apple platforms, $ or $ should not be used in source file properties. 153 CMakeLists.txt cmake_minimum_required(VERSION 3.0) project(Example) # Define an executable which generates various files in a # directory passed as a command line argument add_program(generateFiles generateFiles.cpp) # Create a custom target which invokes the above executable # after creating an empty output directory for it to populate, # then invoke a script to archive that directory's contents # and print the MD5 checksum of that archive set(outDir "foo") add_custom_target(archiver COMMAND ${CMAKE_COMMAND} -E echo "Archiving generated files" COMMAND ${CMAKE_COMMAND} -E remove_directory "${outDir}" COMMAND ${CMAKE_COMMAND} -E make_directory "${outDir}" COMMAND generateFiles "${outDir}" COMMAND ${CMAKE_COMMAND} "-DTAR_DIR=${outDir}" -P "${CMAKE_CURRENT_SOURCE_DIR}/archiver.cmake") archiver.cmake cmake_minimum_required(VERSION 3.0) if(NOT TAR_DIR) message(FATAL_ERROR "TAR_DIR must be set") endif() # Create an archive of the directory set(archive archive.tar) execute_process(COMMAND ${CMAKE_COMMAND} -E tar cf ${archive} "${TAR_DIR}" RESULT_VARIABLE result) if(result) message(FATAL_ERROR "Archiving ${TAR_DIR} failed: ${result}") endif() # Compute MD5 checksum of the archive execute_process(COMMAND ${CMAKE_COMMAND} -E md5sum ${archive} OUTPUT_VARIABLE md5output RESULT_VARIABLE result) if(result) message(FATAL_ERROR "Unable to compute md5 of archive: ${result}") endif() # Extract just the checksum from the output string(REGEX MATCH "^ *[^]*" md5sum "${md5output}") message("Archive MD5 checksum: ${md5sum}") 154 17.7. Recommended Practices When custom tasks need to be executed, it is preferable that they be done during the build stage rather than the configure stage. Many developers are not aware that the cmake command itself provides a command mode which abstracts away many of these platform specific tasks: cmake -E cmd [args...] 151 The full set of supported commands can be listed using cmake -E help, but some of the more commonly used ones include: • compare_files • copy • copy_directory • copy_if_different • echo • env • make_directory • md5sum • remove • remove_directory • rename • tar • time • touch Consider the example of a custom task to remove a particular directory and all its contents: set(discardDir "${CMAKE_CURRENT_BINARY_DIR}/private") # Naive platform specific implementation (not robust) if(WIN32) add_custom_target(myCleanup COMMAND rmdir /S /Q "${discardDir}") elseif(UNIX) add_custom_target(myCleanup COMMAND rm -rf "${discardDir}") else() message(FATAL_ERROR "Unsupported platform") endif() # Platform independent equivalent add_custom_target(myCleanup COMMAND "${CMAKE_COMMAND}" -E remove_directory "${discardDir}") The platform specific implementation shows how projects typically try to implement a scenario such as this, but the if-else conditions are testing the target platform rather than the host platform. When using some CMake generators (notably the Unix Makefiles generator), the dependencies between sources and source properties are stronger than one might expect. 263 24.1. Defining And Executing A Simple Test . In general, SYSTEM is intended for paths outside of the project, but even then it should rarely be needed. This is particularly true in relation to project logic which tries to do different things for different build types. Establishing good habits and patterns and applying them consistently throughout a project will help ensure new developers are exposed to better practices. Imported targets are covered in detail in Chapter 16, Target Types. When cmake_parse_arguments() returns, for every keyword, a corresponding variable will be available whose name consists of the specified prefix, an underscore and the keyword name. The inheriting behavior of INHERITED properties only applies to the get_property() command and its analogous get_... functions for specific property types (covered in the sections below). These read-only pseudo properties are technically not global properties (they cannot be retrieved using get_property(), for example), but they are notionally very similar. 12.2. Policy Scope Sometimes a policy setting only needs to be applied to a specific section of a file. If the step option is provided, then this value is added to the previous one after each iteration and the loop stops when the result of that is greater than stop. This clearly expresses the nature of the relationships between targets and communicates unambiguously to all of a project's developers how targets are related. Subsequent builds will not re-run the custom command unless the output file is removed, even if any of the automatically detected dependency targets are rebuilt. One of the main drawbacks to the use of the module is a potential reduction in source code readability. Working With Files . It might be desirable to allow an operation to continue as long as it is making some sort of progress, but if it stalls for more than some acceptable limit, the operation should fail. In such cases, the library will be either STATIC or SHARED, with the choice determined by the value of a CMake variable called BUILD_SHARED_LIBS. Using FORCE to update the cache like this for compiler and linker flags is rarely a good idea. For the latter case, it is preferable to use cmake_minimum_required(VERSION) because the projects using the module may enforce their own minimum CMake version, but the module may have specific minimum version requirements of its own. When defining an imported executable target, certain target properties need to be set before it can be useful. 317 25.5. Custom Install Logic . The most basic of these is the following: add_library(targetName [STATIC | SHARED | MODULE] [EXCLUDE_FROM_ALL] source1 [source2 ...]) This form is analogous to how add_executable() is used to define a simple executable. Expressions starting with TARGET_PDB_FILE follow an analogous pattern as for TARGET_PROPERTY, providing path and file name details for the PDB file used for the target on which the generator expression is being used. This leads to better usability and improved robustness of calling code (e.g. little chance of getting arguments mixed up). In contrast, modules are pre-built chunks of CMake code built on top of the core language features. string(FIND inputString subString outVar [REVERSE]) FIND searches for subString in inputString and stores the index of the found subString in outVar (the first character is index 0). New projects should leave the else() and endif() arguments empty, as shown above. Projects should avoid the NO_POLICY_SCOPE keyword of these commands, as it is intended only for addressing a change in behavior of very early CMake versions and its use is rarely appropriate for new projects. Together, these generator expressions provide a simple yet powerful way to optionally include content. This allows greater flexibility in choosing between static or dynamic libraries as an overall project-wide strategy. This name is used for the top level of a project with some project generators (eg Xcode and Visual Studio) and it is also used in various other parts of the project, such as to act as defaults for packaging and documentation metadata, to provide project-specific variables and so on. • Reading in the contents of a file. 113 14.2. Directory Properties And Commands . • Permission is given to use any of the code samples in this work without restriction. This has the potential to lead to infinite recursion, as the following contrived example demonstrates: 67 function(printme) message("Hello from first") endfunction() function(printme) message("Hello from second") _printme() endfunction() function(printme) message("Hello from third") _printme() endfunction() printme() One would naively expect the output to be as follows: Hello from third Hello from second Hello from first But instead, the first implementation is never called because the second one ends up calling itself in an infinite loop. This amounts to ensuring the correct directory separator is used (backslash on Windows, forward slash everywhere else). 104 13.2. Common Errors . Lastly, for ultimate control, developers can specify the full set of options for crosscompiling, but this is not recommended for normal use. For example: # Old library previously defined like this: add_library(deepCompute SHARED ...) # Now the library has been split in two, so define # an interface library with the old name to effectively # forward on the link dependency to the new libraries add_library(computeAlgoA SHARED ...) add_library(computeAlgoB SHARED ...) add_library(deepCompute INTERFACE) target_link_libraries(deepCompute INTERFACE computeAlgoA computeAlgoB) 141 Chapter 17. NuGet . User-defined functions and macros can support the same flexibility by using the cmake_parse_arguments() command: include(CMakeParseArguments) # Needed only for CMake 3.4 and earlier cmake_parse_arguments(prefix noValueKeywords singleValueKeywords multiValueKeywords argsToParse) The cmake_parse_arguments() command used to be provided by the CMakeParseArguments module, but it became a built-in command in CMake 3.5. The include(CMakeParseArguments) line will do nothing in CMake 3.5 and later, while for earlier versions of CMake it will define the cmake_parse_arguments() command (see Chapter 11, Modules for more on this sort of usage of include()). Furthermore, for GLOB_RECURSE, symlinks to directories are normally reported as entries in outVar rather than descending into them, but the FOLLOW_SYMLINKS option directs CMake to descend into the directory instead of listing it. For example, in CMake 3.2, policy CMP0055 was introduced to provide strict checking on usage of the break() command. The defaults for _STANDARD_REQUIRED and _EXTENSIONS have proven to be relatively unintuitive for many developers, so by explicitly setting them, a project makes clear what standard behavior it expects. To account for custom build types, a build type is considered to be a debug configuration if it is listed in the DEBUG_CONFIGURATIONS global property. Build Type This chapter and the next cover two closely related topics. • Prefer to avoid modifying the various CMAKE_..._FLAGS variables and their configuration specific counterparts. CMake tutorials and example code frequently use the above variables to control the compiler and linker flags. The ALL option makes the all target depend on this new custom target (the various generators name the all target slightly differently, but it is generally something like all, ALL or similar). Asking for the myExe target to be built will result in the generator and the generated source file being built before building myExe. This book will always use out-of-source builds and will follow the pattern of the source and build directories being under a common parent. 65 8.5. Overriding Commands . While the directory structure may be driven by how developers think of the project, the way the project is structured also impacts the build system. 12.1. Policy Control CMake's policy functionality is closely tied to the cmake_minimum_required() command, which was introduced back in Chapter 3, A Minimal Project. CMake will not properly handle flag variables if their contents contain semicolons, which is what a list would be turned into if specified by the project. An internal representation of the project is created as the commands, functions, etc. Variables . As is the usual pattern with CMake, target properties control which standard will be used when building that target's sources and when linking the final executable or shared library. If the HEX option is given, the contents will be converted to a hexadecimal representation, which can be useful for files containing binary data rather than text. STRING The variable is treated as an arbitrary string. It can be an excellent tool for expanding the range of compilers a project can support. All target types and generators support this option, making it the preferred build stage whenever there is a choice. 363 26.4.10. Thanks for the awesome tool! The book's cover and some of the supporting material on the website are the result of a better eye and understanding of graphical design than my own. Developers should ideally be able to change the generator in one place and the rest of the script should still function correctly. CMake 3.11 removed the restriction on using target_...() commands to set these properties, so whereas INTERFACE IMPORTED used to be very similar to plain IMPORTED libraries, with CMake 3.11 they are now much closer to plain INTERFACE libraries in terms of their set of restrictions. A good way to get early experience of the benefits of such an arrangement is to set up two or more different builds for the same source directory. These forms are discussed here for the benefit of understanding older CMake projects, but their use is generally discouraged for new projects. In general, policies are not something that developers are exposed to all that often, mostly just when CMake issues a warning about the project relying on an older version's behavior. In the first stage, the CMakeLists.txt file is read and a representation of the project is built up in memory. These remove much of the boilerplate of defining the test code and allow projects to specify a minimal set of information for the check. If it is known that substitution is not needed, specifying COPYONLY is good practice, since it prevents unnecessary processing and any unexpected substitutions. Like for the string() command, list() expects the operation to perform as its first argument. It counts the named arguments plus any additional unnamed arguments that were given. 308 25.2.3. Apple-specific Targets . After this is done, the generation phase creates the project files. There were some wikis and personal websites that had some useful contents, but there were also many that contained out-of-date or questionable advice and examples. 5.5. Debugging Variables And Diagnostics As projects get more complicated or when investigating unexpected behavior, it can be useful to print out diagnostic messages and variable values during a CMake run. When using configure_file() and substituting file names or paths, a common mistake is to mishandle spaces and quoting. A large proportion of custom tasks are related to file system manipulation. The Ninja generator is an excellent choice for the latter, since it has the broadest platform support of all the generators and it also creates very efficient builds. The first argument to string() defines the operation to be performed and subsequent arguments depend on the operation being requested. 13 3.4. Commenting . String length can be trivially obtained and strings can easily be converted to upper or lower case.) The target_compile_definitions() command is quite straightforward, with each item having the form VAR or VAR=VALUE. Absolute paths are typically only needed when adding directories that are outside the main source tree. 77 10. 122 15. In contrast, for the TARGET form, no APPEND keyword is necessary for second and subsequent calls to add_custom_command() for the same target. The usual CMakeLists.txt syntax is supported, but there is no configure or generate step and the CMakeCache.txt file is not updated. The directory-specific getter command has two forms: get_directory_property(resultVar [DIRECTORY dir] property) get_directory_property(resultVar [DIRECTORY dir] DEFINITION varName) The first form is used to get the value of a property from a particular directory or from the current directory if the DIRECTORY argument is not used. file(COPY someDir DESTINATION . In comparison, macro arguments are string replacements, so whatever was used as the argument to the macro call is essentially pasted into wherever that argument appears in the macro body. 171 18.6. Recommended Practices . These sort of expressions tend to find use in more advanced situations such as handling a custom compiler or to work around a problem specific to a particular compiler or toolchain. 14 4. 37 5.9. Recommended Practices . The file() command provides the bulk of the functionality, with the simplest being the forms that write directly to a file: file(WRITE fileName content) file(APPEND fileName content) Both of these commands will write the specified content to the named file, the only difference between the two being that if fileName already exists, APPEND will append to the existing contents whereas WRITE will discard the existing contents before writing. The setter command is defined as follows: set_directory_properties(PROPERTIES prop1 val1 [prop2 val2 ...]) While being a little more concise, this directory-specific setter command lacks any APPEND or APPEND_STRING option. For a debug build, compiler flags are used to enable the recording of information that debuggers can use to associate machine instructions with the source code. For log messages intended to remain as part of the build, aim to always specify a mode with the message() command. In general, it is recommended to only remove specific entries without wildcards unless it is absolutely certain the wildcards used are safe. If still not found at the top level directory, then the GLOBAL scope will be searched. WIN32 When building the executable on a Windows platform, this option instructs CMake to build the executable as a Windows GUI application. The first run of CMake may therefore appear to do what the developer intended (if placed before any project() command), but if the line is ever changed to specify something else for the flags, that change won't be applied to an existing build because the variable will already be in the cache at that point. HTTPHEADER header Includes a HTTP header for the operation and can be repeated multiple times as needed to provide more than one header value. No entity is specified, although like for the get_property() command, in the case of VARIABLE the variable name is specified as propName. While this setup stage is what CMake is best known for, the CMake suite of tools also includes CTest and CPack for managing the later testing and packaging stages respectively. If the source file is modified, the build will consider the destination to be out of date and will re-run cmake automatically. # Simple multi-line content with bracket syntax, # no = needed between the square bracket markers set(multiLine [[First line Second line]]) # Bracket syntax prevents unwanted substitution set(shellScript [=[#!/bin/bash [[-n "${USER}"]] && echo "Have USER"]=]) # Equivalent code without bracket syntax set(shellScript "#!/bin/bash [[-n \"\${USER}\"]] && echo \"Have USER\" ") As the above example shows, bracket syntax is particularly well suited to defining content like Unix shell scripts. Commands are pre-defined by CMake, whereas functions and macros can be defined either by CMake (typically through modules) or by projects themselves. The following all evaluate to 1 if the respective condition is satisfied, or 0 otherwise. Using this command is so important that CMake will issue a warning if the CMakeLists.txt file does not call cmake_minimum_required() before any other command. This offers most of the advantages of an out-of-source build, but it does still carry with it some of the disadvantages of an in-source arrangement. This result is then cached so subsequent CMake runs do not have to redo the test. They affect all targets created in the current directory scope and below after the commands are called, but any existing targets remain unaffected (i.e. similar to the behavior of add_compile_options()). 378 27.2. FetchContent . The same approach as was discussed for add_subdirectory() in Section 7.1.2, "Scope" can be used for functions too. If, however, some other target defined in the same directory scope depends on any of the output files, CMake will automatically create dependency relationships that ensure the output files are generated before the target that needs them. 9 3. It can also affect the order in which header paths are searched for some compilers. The --build option points to the build directory used by the CMake project generation step. Aside from these cases, cmake_policy(VERSION) usually expresses the intent more clearly, but both commands will effectively achieve the same thing from a policy perspective. Primarily, there are two main effects of calling include_directories(): • The listed paths are added to the INCLUDE_DIRECTORIES directory property of the current CMakeLists.txt file. If requiring CMake 3.9 or later, use the GENERATOR_IS_MULTI_CONFIG global property to definitively query the generator type instead of relying on the existence of CMAKE_CONFIGURATION_TYPES to perform a less robust check. An important difference between normal and cache variables is that the set() command will only overwrite a cache variable if the FORCE keyword is present, unlike normal variables where the set() command will always overwrite a pre-existing value. Sometimes, however, a project needs to create one or more files by running a command or series of commands and the generation of that file doesn't really belong to any existing target. This therefore places more of a burden on the project to ensure that all targets have correctly specified language standard details. This highlights the danger of using return() within macros. On Windows, the default library name would be targetName.lib, while on Unix-like platforms, it would typically be libtargetName.a. SHARED Specifies a shared or dynamically linked library. 3. Section 25.2.1, "Interface Properties" covers this aspect of specifying header search paths in more detail. Anything else that links to library A doesn't need to know about B because it is an internal implementation detail of A. If adding a custom build type or removing one of the default build types, do not modify the cache variable but instead change the regular variable of the same name (it will take precedence over the cache variable). Furthermore, the techniques for multi and single configuration generators can and should be combined to give robust behavior across all generator types. CMake also offers very powerful features for managing dependencies between targets and how libraries are linked. Instead, the file(GENERATE...) command can be used: file(GENERATE OUTPUT outFile INPUT inFile | CONTENT content [CONDITION expression]) This works somewhat like file(WRITE...) except that it writes out one file for each build type supported for the current CMake generator. Attribution is not required. The main difficulty with in-source builds, however, is that all the build outputs are intermixed with the source files. Support for specific compiler flags can be checked using the CheckCompilerFlag modules, each of which provide a single macro with a name following a predictable pattern: include(CheckCCompilerFlag) check_c_compiler_flag(flag resultVar) include(CheckCXXCompilerFlag) check_cxx_compiler_flag(flag resultVar) include(CheckFortranCompilerFlag) check_fortran_compiler_flag(flag resultVar) The flag-checking macros update the CMAKE_REQUIRED_DEFINITIONS variable internally to include flag in a call to the appropriate check__source_compiles() macro with a trivial test file. In the case of ccmake, pressing enter on that cache variable will cycle through the values provided. 100 Part II: Builds In Depth . One other drawback to in-source builds is that it can be nontrivial to clear out all build output and start again with a clean source tree. • Prefer to use command mode -E echo rather than the COMMENT keyword with add_custom_command() and add_custom_target(). The file() command provides the inverse operation, converting an absolute path to relative: file(RELATIVE_PATH outVar relativeToDir input) The following example demonstrates its usage: 157 set(basePath /base) set(fooBarPath /base/foo/bar) set(otherPath /other/place) file(RELATIVE_PATH fooBar ${basePath} ${fooBarPath}) file(RELATIVE_PATH other ${basePath} ${otherPath}) # The variables now have the following values: # fooBar = foo/bar # other = ../other/place The third form of the get_filename_component() command is a convenience for extracting parts of a full command line: get_filename_component(progVar input PROGRAM [PROGRAM_ARGS argVar] [CACHE]) With this form, input is assumed to be a command line which may contain arguments. The dependency CMake automatically creates does not enforce this, it creates a weaker dependency which does ensure generator is brought up to date but the custom command is only run if the output file is missing altogether. The following example demonstrates the behavior: 55 CMakeLists.txt add_subdirectory(subdir) message("====") include(subdir/CMakeLists.txt) subdir/CMakeLists.txt message("CMAKE_CURRENT_SOURCE_DIR message("CMAKE_CURRENT_BINARY_DIR message("CMAKE_CURRENT_LIST_DIR message("CMAKE_CURRENT_LIST_FILE message("CMAKE_CURRENT_LIST_LINE = = = = = ${CMAKE_CURRENT_SOURCE_DIR}") ${CMAKE_CURRENT_BINARY_DIR}") ${CMAKE_CURRENT_LIST_DIR}") ${CMAKE_CURRENT_LIST_FILE}") ${CMAKE_CURRENT_LIST_LINE}") This produces output like the following: CMAKE_CURRENT_SOURCE_DIR CMAKE_CURRENT_BINARY_DIR CMAKE_CURRENT_LIST_DIR CMAKE_CURRENT_LIST_FILE CMAKE_CURRENT_LIST_LINE ==== CMAKE_CURRENT_SOURCE_DIR CMAKE_CURRENT_BINARY_DIR CMAKE_CURRENT_LIST_DIR CMAKE_CURRENT_LIST_FILE CMAKE_CURRENT_LIST_LINE = = = = = /somewhere/src/subdir /somewhere/build/subdir /somewhere/src/subdir /somewhere/src/subdir/CMakeLists.txt 5 = = = = = /somewhere/src /somewhere/build /somewhere/src/subdir /somewhere/src/subdir/CMakeLists.txt 5 The above example also highlights another interesting characteristic of the include() command. There was a distinct gap, which meant those new to CMake had a hard time learning good practices, leading to many becoming overwhelmed or frustrated. 50 7.1.1. Source And Binary Directory Variables . SHARED Targets created with add_library(name SHARED ...) or equivalent, such as omitting the SHARED keyword but with the BUILD_SHARED_LIBS variable set to true. set(regex "^ *FOO_VERSION *= *([^]+) *$") file(STRINGS config.txt fooVersion REGEX "${regex}") string(REGEX REPLACE "${regex}" "\\1" fooVersion "${fooVersion}") If config.txt contained a line like this: FOO_VERSION = 2.3.5 Then the value stored in fooVersion would be 2.3.5. 168 18.4. File System Manipulation In addition to reading and writing files, CMake also supports other common file system operations. 15 EXCLUDE_FROM_ALL Sometimes, a project defines a number of targets, but by default only some of them should be built. 3.4. Commenting Before leaving this chapter, it will be useful to demonstrate how to add comments to a CMakeLists.txt file. Developers should become familiar with the concepts of PRIVATE, PUBLIC and INTERFACE relationships. This is like NAME except only the part of the file name up to but not including the first "." is extracted. In the case of functions and variables, the symbol needs to resolve to something that is part of the test executable. Many of these modules are provided as part of the support for find_package() calls (see Section 23.5, "Finding Packages"), while others are intended to be used more directly via include() to bring things into the current scope. • If a cache variable is of type STRING, then CMake GUI will look for a cache variable property named STRINGS. 9 Chapter 3. Specifying generator expressions for each feature to cover the supported and unsupported cases would be tedious and potentially more fragile, but a more convenient mechanism is provided by CMake through its module system. 9.1. General Property Commands CMake provides a number of commands for manipulating properties. For installed targets, relative paths are normally used and they would be interpreted as relative to the base install location rather than the source directory. 8.5. Overriding Commands When function() or macro() is called to define a new command, if a command already exists with that name, the undocumented CMake behavior is to make the old command available using the same name except with an underscore prepended. The MAIN_DEPENDENCY option is intended to identify a source file which should be considered the main dependency of the custom command. Compare this with properties which are typically well defined and documented by CMake and which always apply to a specific entity. The meaning of the value(s) are property specific. While TARGET_PROPERTY is a very flexible expression type, it is not always the best way to obtain information about a target. They can also easily become out of date as code development continues over time. Any variables which were newly added or which changed value from the previous run will be highlighted in red (when a project is first loaded, all variables are shown highlighted). Thinking about and establishing these things early places the project in a better position when it comes time to deliver the first public release. Essentially, CMake supports basic regular expression syntax only. file(STRINGS myStory.txt versionLines REGEX "(PKG|MODULE)_VERSION") It can also be combined with LIMIT_COUNT to obtain just the first match. All such aliases within a project would typically share the same projNamespace. An example of this would be a function defined in library A which has at least one parameter of a type defined and implemented in library B, so code cannot call the function from A without providing a parameter whose type comes from B. The OFFSET option can be used to read only from the offset specified, counted in bytes from the beginning of the file. In Section 7.3, "Ending Processing Early", the return() statement was discussed as a way to end processing early within a file or function. # Wrong, list used instead of a string set(CMAKE_CXX_FLAGS -Wall -Werror) # Correct, but see later sections for why appending would be preferred set(CMAKE_CXX_FLAGS "-Wall -Werror") # Appending to existing flags the correct way (two methods) set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Werror") string(APPEND CMAKE_CXX_FLAGS " -Wall -Werror") Distinguish between cache and non-cache variables All of the variables mentioned above are cache variables. This means that when copying individual files, the copy command does not require an earlier step to ensure the destination directory exists. 43 6.1.5. Existence Tests . This is an important shift in thinking, as it affects how a project can most effectively define the way targets should be built. 133 16.3. Promoting Imported Targets . 269 24.4. Parallel Execution . For example: mkdir build cd build cmake -G "Unix Makefiles" ../source If the -G option is omitted, CMake will choose a default generator type based on the host platform. 339 26.3. Multi Configuration Packages . These are not intended to be set by the project, but rather by the developer as a cache variable temporarily to enable/disable a warning or to check whether the project issues warnings with a particular policy enabled. It would be up to the project what such project-specific properties mean and how they might affect the build. While it primarily has a direct effect on the compiler and linker behavior, it also has an effect on the directory structure used for a project. 39 6.1.2. Logic Operators . This lack of separation causes directories to become cluttered with all sorts of files and subdirectories, making it harder to manage the project sources and running the risk of build outputs overwriting source files. The value of a cache variable is retrieved in exactly the same way as a normal variable (i.e. with the ${myVar} form), but the set() command is different when used to set a cache variable: set(varName value... The result of the test is stored in the resultVar cache variable in the usual way. These can be used to set the language standard behavior for the entire project, ensuring consistent usage across all targets. POST_BUILD The commands will be run after all other rules for the specified target. For example, flags for a custom Profile build type could be defined as follows: set(CMAKE_C_FLAGS_PROFILE set(CMAKE_CXX_FLAGS_PROFILE set(CMAKE_EXE_LINKER_FLAGS_PROFILE set(CMAKE_SHARED_LINKER_FLAGS_PROFILE set(CMAKE_STATIC_LINKER_FLAGS_PROFILE set(CMAKE_MODULE_LINKER_FLAGS_PROFILE "-p "-p "-p "-p "-p "-p -g -g -g -g -g -g -O2" -O2" -O2" -O2" -O2" -O2" CACHE CACHE CACHE CACHE CACHE CACHE STRING STRING STRING STRING STRING STRING "") "") "") "") "") "") The above assumes a GCC-compatible compiler to keep the example simple and turns on profiling as well as enabling debugging symbols and most optimizations. This control over whether old or new behavior should be used is done through CMake's policy mechanisms. For the same reason, the various quote characters in the content are also not interpreted as anything other than part of the content. Later chapters cover these and other uses of cache variables, but first, the ways to manipulate such variables need to be understood. This chapter focuses on the more commonly used and general purpose properties, with later chapters covering a number of the more specific ones. When a target is created, the initial value of the _EXTENSIONS property is taken from the CMAKE__EXTENSIONS variable. The executable will then only be built if it is explicitly requested by the build command or if it is a dependency for another target that is part of the default ALL build. The following example highlights the situation: # WARNING: This macro is misleading macro(dangerous) # Which ARGN? TIMEOUT seconds Abort the operation if more than seconds have elapsed. 45 6.2. Looping . cmake_minimum_required(VERSION 3.2) project(MyApp) add_executable(myExe main.cpp) Each line in the above example executes a built-in CMake command. 89 11.3. Checking Existence And Support . These properties could, however, be set using set_property() or set_target_properties(). If any part of the destination path does not exist, CMake will attempt to create the missing directories as part of the call. Unlike their non-interface counterparts, none of the above INTERFACE_... properties are initialized from directory properties. The result of running the commands will be stored in the named variable as either an integer return code of the last command or a string containing some kind of error message. It is often also called the build directory. 9.8. Recommended Practices Properties are a crucial part of CMake. Typically, the location doesn't need to be configuration-specific, so it is very common for only IMPORTED_LOCATION to be set. CMake will add a compiler flag for each path with an appropriate prefix prepended (typically -I or /I). These are some common examples where the high level control provided by the cmake_minimum_required() command alone is not enough. In addition, CMake can be used as a platform independent scripting language, processing a file as a sequence of CMake commands when invoked with the -P option. Visual Studio compilers again may support the requested standard by default or not, depending on compiler version. A fast configure stage is important because it can be invoked automatically when some files are modified (e.g. any CMakeLists.txt file in the project, any file included by a CMakeLists.txt file or any file listed as a source of a configure_file() command as discussed in the next chapter). This is assumed to be a list of directories and CMake will search each of these in order. CMakeLists.txt set(BAR "Some \"quoted\" value") configure_file(quoting.txt.in quoting.txt) configure_file(quoting.txt.in quoting_escaped.txt ESCAPE_QUOTES) quoting.txt.in A: @ B: "@" quoting.txt A: Some "quoted" value B: "Some "quoted" value" quoting_escaped.txt A: Some \"quoted\" value B: "Some \"quoted\" value" As the above example shows, the ESCAPE_QUOTES option causes escaping of all quotes regardless of their context. Certain categories of checks are so common that CMake provides dedicated modules for them. Ideally, a project would only modify them early in the top level CMakeLists.txt file, if at all, so as to minimize the opportunity for misuse and developer surprise. Developers may use these to temporarily change a specific policy setting's default or to prevent warnings about a particular policy. 6 2.2. Out-of-source Builds The more preferable arrangement is for the source and build directories to be different, which is called an out-of-source build. This is done using the $ generator expression which lists the object files in a form suitable for add_executable() or add_library() to consume. Usually, the output file would be different for each build type. One example is the add_subdirectory() command which pushes a new policy scope onto the stack upon entering the specified subdirectory and pops it again when the command returns. The primary variables directly affecting the compiler flags have the following form: • CMAKE__FLAGS • CMAKE__FLAGS_ In this family of variables, corresponds to the language being compiled, with typical values being C, CXX, Fortran, Swift and so on. Developers may find the CMake documentation for the CMAKE__KNOWN_FEATURES variables to be a particularly useful resource, since it not only lists the features understood by that particular version of CMake, it also contains references to standard documents relating to each feature. This is intended to mimic aspects of the behavior of the Unix rm -f command. The input can also be a list of paths specified in a form compatible with the platform's PATH environment variable. Instead, hashing will only be performed if the developer explicitly requests it as a build target. Furthermore, since no CMake release in the 3.x series has used a tweak number, projects should not need to specify one either. 357 26.4.8. DEB . In reality, however, there are a few different types of dependency relationships that can exist between libraries: PRIVATE Private dependencies specify that library A uses library B in its own internal implementation. Libraries . The following example illustrates the behavior. The main disadvantage of using the oldest workable version is that it may result in more deprecation warnings, since newer CMake versions will warn about older behaviors to encourage projects to update

themselves. Anything defined by a Find module or package config file is generally expected to have local visibility, so they shouldn't generally add globally visible targets. 212   21.6.1. Raspberry Pi . The two are largely interchangeable except for the top of the project where a call to cmake_minimum_required() is mandatory to enforce a minimum CMake version. The most basic way to run CMake is via the cmake command line utility. The build tool doesn't even have to be on the user's PATH for cmake --build to work (although it may need to be for the initial configuration step when cmake is first invoked). It is common to find different file types or individual modules grouped under their own directories, or for files belonging to logical functional groupings to be in their own part of the project's directory hierarchy. 102   13. This offers a number of advantages: • Build logic is localized, meaning that characteristics of the build can be defined in the directory where they have the most relevance. CMake's more mature features can be used to control behavior at a coarse level at the expense of losing the ability to define relationships between targets. In some cases, there can be multiple ways of achieving the same thing, so it is useful to be aware of the different choices and understand how to use them effectively.) The targetName is required as the first argument, but after that, callers can provide any number of PRIVATE, PUBLIC or INTERFACE sections in any order, with each section permitted to contain any number of items. Some of these variables are heavily used by projects, such as those that are only defined when building for a particular platform (WIN32, APPLE, UNIX, etc.). Unless there is a good reason to structure things that way, keeping the build directory completely outside of the source tree instead is recommended. 6.2.3. Interrupting Loops Both while() and foreach() loops support the ability to exit the loop early with break() or to skip to the start of the next iteration with continue(). Two other forms also exist which can be used to define executable targets that reference other things. 6.1.5. Existence Tests The last category of if expressions support testing whether or not various CMake entities exist. • If value is an unquoted constant with value 0, OFF, NO, FALSE, N, IGNORE, NOTFOUND, an empty string or a string that ends in -NOTFOUND, it is treated as false. If a relative path is given, it will be treated as relative to the current source directory (very early versions of CMake had different behavior, see the documentation for policy CMP0015 for details). The fact that omitting the prefix makes the command treat the item as a generic option instead is also counter-intuitive given the command's name. This useful resource is very helpful for looking up the various commands, options, keywords, etc. A popular example of their use is for header-only libraries where there is no physical library that needs to be linked, but header search paths, compiler definitions, etc. Because of this, much of the material online and elsewhere about CMake tends not to use them, which is unfortunate, since generator expressions are typically more robust and provide more generality than older methods. This help string is used as the tooltip in the CMake GUI and as a one-line help tip in the ccmake tool. 106   13.4. Recommended Practices . This is typically when custom commands should be run, but there are some situations where a custom task needs to be performed during the configure stage instead, such as: • Executing external commands to obtain information to be used during configuration. If subString does not appear in inputString, then outVar will be given the value -1. 86   11.1. Useful Development Aids . These typically do a better job of abstracting away things like platform differences, option-dependent tool selection and so on, but more importantly the usage requirements are then robustly handled by CMake. For example, on macOS, the app bundle layout has a very specific directory structure, whereas on iOS, the directory structure is flattened. No variable/string test is performed, the arguments are used exactly as the values are specified. This is where add_custom_command() should be used with the TARGET keyword as follows: 144 add_custom_command(TARGET targetName buildStage   COMMAND command1 [args1...]   [COMMAND command2 [args2...]]   [WORKING_DIRECTORY dir]   [BYPRODUCTS files...]   [COMMENT comment]   [VERBATIM]   [USES_TERMINAL]) Most of the options are very similar to those for add_custom_target(), but instead of defining a new target, the above form attaches the commands to an existing target. 82 One other generator expression relating to targets deserves special mention. 162 The following copies all header (.h) and script (.sh) files from someDir, except headers whose file name ends with _private.h. The directory structure of the source is preserved. Imported targets are encountered frequently once projects start using packages built externally or they refer to tools from the file system that are found through Find modules. There are some common examples where well-intentioned guidance leads to logic which only works for a subset of supported project generators or platforms, but where the use of suitable generator expressions instead would result in no such limitations. One approach is to use generator expressions to conditionally set compiler defines or include directories based on the availability of a particular compiler feature. Variables, properties, flow control, generator expressions, functions, etc. Two main methods are provided by CMake for specifying language requirements. Only the non-INTERFACE_... target properties are modified (i.e. LINK_LIBRARIES, LINK_FLAGS and STATIC_LIBRARY_FLAGS). CMAKE_BINARY_DIR The top-most directory of the build tree. In other cases, the project may need to iterate over a set of values or to keep repeating some set of steps until a certain condition is met. add_definitions(-DSomeSymbol /DFoo=Value ...) remove_definitions(-DSomeSymbol /DFoo=Value ...) The add_definitions() and remove_definitions() commands add and remove entries in the COMPILE_DEFINITIONS directory property. Installing......Page 30625.1.1. Relative Layout......Page 30725.1.2. Base Install Location......Page 30925.2. Installing Targets......Page 31125.2.1. Interface Properties......Page 31625.2.2. RPATH......Page 31725.2.3. Apple-specific Targets......Page 32025.3. Installing Exports......Page 32325.4. Installing Files And Directories......Page 32625.5. Custom Install Logic......Page 32925.6. Installing Dependencies......Page 33025.7. Writing A Config Package File......Page 33125.7.1. Config Files For CMake Projects......Page 33225.7.2. Config Files For Non-CMake Projects......Page 33925.8. Recommended Practices......Page 34026.1. Packaging Basics......Page 34326.2. Components......Page 34826.3. Multi Configuration Packages......Page 35226.4. Package Generators......Page 35326.4.1. Simple Archives......Page 35426.4.2. Qt Installer Framework (IFW)......Page 35626.4.3. WIX......Page 36126.4.4. NSIS......Page 36226.4.5. DragNDrop......Page 36426.4.6. productbuild......Page 36526.4.7. RPM......Page 36626.4.8. DEB......Page 37126.5. Recommended Practices......Page 37227.1. ExternalProject......Page 37527.1.1. Tour Of Main Features......Page 37627.1.2. Step Management......Page 38227.1.3. Miscellaneous Features......Page 38527.1.4. Common Issues......Page 38727.2. FetchContent......Page 38927.2.1. Developer Overrides......Page 39327.2.2. Other Uses For FetchContent......Page 39427.2.3. Restrictions......Page 39527.3. ExternalData......Page 39627.4. Recommended Practices......Page 39728.1. Superbuild Structure......Page 39928.2. Non-superbuild Structure......Page 40128.3. Common Top Level Subdirectories......Page 40528.4. IDE Projects......Page 40628.5. Defining Targets......Page 40928.5.1. Target Sources......Page 41028.5.2. Target Outputs......Page 41328.5.3. Windows Specific Issues......Page 41528.6. Miscellaneous Project Features......Page 41628.7. Recommended Practices......Page 418Index......Page 422 This can be handled by having the project provide a default value if CMAKE_BUILD_TYPE hasn't been set. A common example of this is a project that requires the Xcode generator due to the Apple platform's unique features, such as code signing and iOS/tvOS/watchOS support. 5.3. Cache Variables In addition to normal variables discussed above, CMake also supports cache variables. The object library's usage requirements do, however, propagate transitively exactly like an ordinary library would. 282   24.8.3. CTest Configuration . This is by no means a comprehensive set, but they do give a flavor of the sort of functionality that is available. 112   14.1.3. Target Property Commands . 163 As a convenience, if destination includes a path (relative or absolute), CMake will try to create the destination path as needed when copying only a single source file. The use of CMAKE_MODULE_PATH should be preferred over hard-coding absolute or relative paths across complex directory structures in include() calls, since this will encourage generic CMake logic to be decoupled from the places where that logic may be applied. CMake's command mode, which was introduced in the previous chapter, also provides a variety of file-related features which overlap with much of what file() provides, but it covers a complimentary set of scenarios to file() rather than being an alternative in most cases. INTERFACE Interface dependencies specify that in order to use library A, parts of library B must also be 17 used. A special mention is deserved for my colleague Mike Wake. # Imported library created with local visibility. Most of the time, developers will probably find the explanation in Section 4.3, "Linking Targets" more intuitive, but the above more precise description can help explain the behavior in more complex projects where properties may be manipulated in unusual ways. There is a directory property of the same name, but it is not used to provide an initial value for this target property. Specifically, consider whether CMake's script mode would be a better choice than using a Unix shell script or Windows batch file, or even a script for a language like Python, Perl etc. 49 Chapter 7. Working with very recent feature additions to a language, such as when using proposed features for an upcoming language release, is one scenario where compile features may be a useful approach where CMake supports such features. The platform independent version, however, has no such weakness. The following variation on the hashing example of the preceding section demonstrates the technique: add_executable(myExe main.cpp) # Output file with relative path, generated in the build directory add_custom_command(OUTPUT myExe.md5   COMMAND writeHash $) # Absolute path needed for DEPENDS, otherwise relative to source directory add_custom_target(computeHash   DEPENDS ${CMAKE_CURRENT_BINARY_DIR}/myExe.md5) 146 When defined this way, building the myExe target will not result in running the hashing step, unlike the earlier example which added the hashing command as a POST_BUILD step of the myExe target. All of these modules have a name beginning with Check. 140 Inevitably, at some point it may become desirable to rename or refactor a library, but there may be external projects which expect the existing library targets to be available to link to. These modules allow aspects of some languages or language-related packages to be made available to projects by defining relevant functions, macros, variables and properties. 57 If the project requires CMake 3.10 or later, prefer to use the include_guard() command without arguments instead of an explicit if-endif block in cases where multiple inclusion of a file must be prevented. Both environments 104 keep separate directories for the different build types, so switching between builds doesn't cause constant rebuilds. CMake provides the list() command to facilitate such tasks. A core tool CMake provides for this the string() command, which provides a wide range of useful string handling functionality. Both are included in the official CMake release packages on all platforms. if("23a" EQUAL 23) Version number comparisons are somewhat like an enhanced form of numerical comparisons. While projects are able to define whatever build types they want, the default build types provided by CMake are usually sufficient for most projects: Debug With no optimizations and full debug information, this is commonly used during development and debugging, as it typically gives the fastest build times and the best interactive debugging experience. • Writing or touching files which need to be updated any time CMake is re-run. The focus has shifted from a more build-global view to one where the requirements of each individual target can be controlled, along with how those requirements should or should not be carried across to any other targets that depend on it. The preferred way to handle this is to explicitly set the policy to the desired behavior (OLD or NEW), which stops the warning. This can be particularly true of Find… modules which may not track newer versions of the packages they are finding as closely as one might like. 320   25.6. Installing Dependencies . In practice, projects would more typically set the variables that provide the defaults for the above target properties rather than setting the target properties directly. The remainder of this chapter introduces a number of interesting modules that are included as part of a CMake release. If the function or variable is provided by a library, that library must be linked as part of the test, which can be done using the CMAKE_REQUIRED_LIBRARIES variable. For example, if a DIRECTORY property is requested but is not set for the directory specified, its parent directory scope's property is queried recursively up the directory scope hierarchy until the property is found or the top level of the source tree is reached. It is a good habit to always specify the type of the variable on the command line in general anyway so that it is likely to be shown in GUI applications in the most appropriate form. If another executable target name is listed as the command to execute, CMake will automatically substitute the built location of that other target's executable. Later chapters give further details on setting up the toolchain and demonstrate how to query and modify things like compiler flags, compiler locations, etc. For some compilers/linkers, this setting can change the standard library the target is linked with (see examples below). Where possible, it is recommended that developers work with a recent CMake release. For the most part, developers tend to ignore the generation step and just think of it as the end result of configuration. 6   2.2. Out-of-source Builds . The behavior of return() within a macro, on the other hand, is very different. Any attempt to use such a name for a different target type will result in an error. A simplistic explanation is that object files are only added to a target that links directly to the object library, not transitively beyond that. The VERSION keyword was added to the project() command, mandating a version number of the form major.minor.patch.tweak as the expected format. 74   9.4. Target Properties . Macros should generally only 68 be used where the contents of the macro body really do need to be executed within the scope of the caller. Many of these rely on predefined CMake variables for their logic, especially variables relating to the compiler and target platform. If using CMake 3.8 or later, compile features can be used to specify the desired language standard on a per target basis. 16.4. Recommended Practices Version 3.0 of CMake brought with it a significant change to the recommended way projects should manage dependencies and requirements between targets. Pay special attention to any possible return() statements that exit that section of code and ensure no push is left without a corresponding pop. It also gives developers a good base to work from should they need to experiment with things like compiler settings, since these are easily found and modified within the GUI environment. The value of that argument can be accessed in the function or macro body using the usual variable notation, even though macro arguments are not technically variables. Use of the CMakePushCheckState module is highly recommended, but avoid using the RESET option to cmake_push_check_state() if support for CMake versions before 3.10 is important. 10   3.1. Managing CMake versions . • If copying more than one file or a whole directory structure, the file(COPY) or file(INSTALL) command must be used. Modifying the previous example slightly, the effect of PARENT_SCOPE becomes clear: 53 CMakeLists.txt set(myVar foo) message("Parent (before): myVar = ${myVar}") add_subdirectory(subdir) message("Parent (after): myVar = ${myVar}") subdir/CMakeLists.txt message("Child (before): myVar = ${myVar}") set(myVar bar PARENT_SCOPE) message("Child (after): myVar = ${myVar}") This produces the following output: Parent Child Child Parent (before): (before): (after): (after): myVar myVar myVar myVar = = = = foo foo foo bar ① ② ① The myVar in the child scope is not affected by the set() call because the PARENT_SCOPE keyword tells CMake to modify the parent's myVar, not the local one. Sometimes a quick scan of the available modules can yield an unexpected hidden gem, or a new module may offer a better maintained implementation of something a project has been implementing in an inferior way up to that point. Directories following a PUBLIC keyword are added to both. Multiple executables can also be defined within the one CMakeLists.txt file by calling add_executable() multiple times with different target names. This behavior stems from their main intended use, which is as part of a Find module or package config file. CMake offers even more conditional tests based on things like platform and compiler details, CMake policy settings, etc. 52   7.2. include() . If listing a file in DEPENDS for either add_custom_target() or add_custom_command(), always use an absolute path to avoid non-robust legacy path matching behavior. In such cases, the targetName used with target_link_libraries() must have been defined by an add_executable() or add_library() command in the same directory from which target_link_libraries() is being called. Since the target_include_directories() command is basically just populating the relevant target properties, all the usual features of those properties apply. Because they can't be linked, they therefore don't provide transitive dependencies to the targets they are added to as objects/sources. The expression may use any of the following operators which all have the same meaning as they would in C code: + - * / % | & ^ ~ > * / %. These expressions have names starting with TARGET_LINKER_FILE and TARGET_SONAME_FILE and tend not to be used as frequently as the TARGET_FILE expressions. Common uses of this would be to provide compiler flags only for debug builds or to select different implementations for different build types. In other words, they are used to specify compiler flags which consuming targets should inherit. That directory acts like a catalog of ready-to-use functionality, able to be conveniently accessed from anywhere in the project. The file() command can not only copy or create files, it can also be used to read in a file's contents: file(READ fileName outVar   [OFFSET offset] [LIMIT byteCount] [HEX]) Without any of the optional keywords, this command reads all of the contents of fileName and stores them as a single string in outVar. If the item is not in the list, it is not an error. foreach(outerVar IN ITEMS a b c)   unset(s)   foreach(innerVar IN ITEMS 1 2 3)   # Stop inner loop once string s gets long   list(APPEND s "${outerVar}${innerVar}")   string(LENGTH s length)   if(length GREATER 5)   break() ①   endif()   # Do no more processing if outer var is "b" if(outerVar STREQUAL "b") continue() ② endif() message("Processing ${outerVar}-${innerVar}") endforeach()   message("Accumulated list: ${s}") endforeach() ① Ends the innerVar for loop early. include(CheckSymbolExists) check_symbol_exists(symbol headers resultVar) include(CheckCXXSymbolExists) check_cxx_symbol_exists(symbol headers resultVar) For each of the items specified in headers (a CMake list if more than one header needs to be given), a corresponding #include will be added to the test source code. From CMake 3.8, the value 17 is also supported and from CMake 3.12, the value 20 can be used. hideMe=mysteryValue ... Each option given to add_compile_options() is added to the COMPILE_OPTIONS directory property. A useful pattern is to collect together a project's module files in a single directory and add it to the CMAKE_MODULE_PATH somwhere near the beginning of the top level CMakeLists.txt file. In addition, CMake doesn't require variables to be defined before using them. This chapter covers all of these various target types and discusses their uses. His support in giving the space and encouragement to work through these things and his willingness to wear some short-term (and sometimes not-so-short-term) pain have been an instrumental part of distilling many processes and techniques down to what works in practice. set(myVar foo) set(result ${myVar}) set(myVar bar CACHE STRING "") # Local myVar # result = foo # Cache myVar set(result ${myVar})   # First run: result = bar # Subsequent runs: result = foo set(myVar fred) set(result ${myVar}) # result = fred Loosely speaking, the resultant behavior is that ${myVar} will retrieve the last value that was assigned to myVar, regardless of whether it was a normal variable or a cache variable. The cmake executable does not have to be on the current PATH and if multiple versions of CMake are installed, the correct version is always used, regardless of which one might otherwise have been selected based on the user's PATH. This allows consuming projects to link to the namespaced target name instead of the real target name, a technique which enables consuming projects to switch between building the child project themselves or using a pre-built installed project relatively easily. When it comes to naming targets, don't use target names that are too generic. If this option is not provided, CMake looks for a variable named CMAKE_TLS_VERIFY instead. One of its important responsibilities is to check the compilers for each enabled language and ensure they are able to compile and link successfully. For everyday development, however, the GUI tools provided by CMake often present a better user experience. Some target properties affect how targets are presented in the developer's IDE project, while others affect the tools used when compiling/linking. The following are equivalent, with no error or warning if myVar does not already exist: set(myVar) unset(myVar) In addition to variables defined by the project for its own use, the behavior of many of CMake's commands can be influenced by the value of specific variables at the time the command is called. This becomes important at the link stage where a consistent standard library should be used across all the linked libraries and object files. Both DOWNLOAD and UPLOAD share a number of common options: LOG outVar Save logged output from the operation to the named variable. To support such use cases, CMake allows the output file to be the same for different build types, but only if the generated file contents are also identical for those build types. When performing the copy, configure_file() has the ability to substitute CMake variables. A property can return TRUE for DEFINED and FALSE for SET, or vice versa. If directory property commands must be used, do so as early in the CMakeLists.txt file as possible to avoid some of the less intuitive behavior described in the preceding sections. An INTERFACE IMPORTED library is not required to (and indeed is prohibited from) setting an IMPORTED_LOCATION. Perhaps more likely to be useful, however, is the LIMIT_COUNT option which limits the total number of lines stored rather than the number of bytes. Projects should rarely need to query or modify source file properties, but for those situations that require it, CMake provides dedicated setter and getter commands to make the task easier. The following directory structure shows such an arrangement: The corresponding CMakeLists.txt file then only needs to add the cmake directory to the CMAKE_MODULE_PATH and it can then call include() using just the base file name when loading each of the modules. As an example, if a project defines its own custom build type called StrictChecker and that build type should be considered an unoptimized debug build type, it can (and should) make this clear like so: set_property(GLOBAL PROPERTY APPEND DEBUG_CONFIGURATIONS StrictChecker) New projects should normally prefer to use generator expressions instead of the debug and optimized keywords with the target_link_libraries() command. 28 CMake's two-stage setup process was introduced in Section 2.3, "Generating Project Files". 5.4. Manipulating Cache Variables Using set() and option(), a project can build up a useful set of customization points for its developers. I have personally learned much from him just by observing the way he interacts with developers and users, providing strong leadership and fostering an environment of respect for others. The part of the variable name must be one of the following: EXE Targets created with add_executable(). This may lead to better output buffering behavior in some situations, such as helping IDE environments capture and present the build output in a more timely manner. The first variable will be applied to all build types, including single configuration generators with an empty CMAKE_BUILD_TYPE, while the second variable is only applied to the build type specified by . Sometimes, a particular CMake behavior needs to change or more stringent checks and warnings may be introduced in newer versions. While this works, it does have the disadvantage of creating a build target with no real meaning. The return() command can be used for exactly this purpose, but note that it cannot return a value to the caller. If there is any chance of escaping being relevant, use of the VERBATIM keyword is recommended. 11.5. Recommended Practices CMake's collection of modules provides a wealth of functionality built on top of the core CMake language. Some specific cases are discussed in Chapter 25, Installing where developers may come up against certain limitations of imported targets, but such scenarios are not all that common. 142   17.2. Adding Build Steps To An Existing Target . Processing will continue until the configure stage completes, but generation will not be performed. Note, however, that this dependency relationship has limitations. For example, with a prefix of ARG, the variable corresponding to a keyword named FOO would be ARG_FOO. For GLOB_RECURSE, they can also include path components. In addition to executables, developers also frequently need to build and link libraries. When using a STATUS mode, the message will be printed correctly ordered with other CMake messages and will be preceded by two hyphens, whereas without any mode keyword, no leading hyphens are prepended and it is not unusual for the message to appear out of order relative to other messages which did include a mode keyword. Such uses reduce the amount of ifthen-else logic, resulting in better readability as long as the generator expressions are not too complex. No further escaping is performed by the platform, so the developer can have confidence in how the command is ultimately constructed for execution. It was added to facilitate child processes being able to configure and run sub-builds with the same compilers as the main project. These files might be used by the commands or they could just be some additional files which are loosely associated with the target, such as documentation, etc. GLOBAL means the build itself, so there is no specific entity name required. 146   17.4. Configure Time Tasks . Language Requirements . Projects should be wary about setting CMAKE_INCLUDE_DIRECTORIES_BEFORE, as most developers will likely assume that the default behavior of directories being appended will apply. Version 3.2 is perhaps the oldest any new project should consider, since it provides a reasonably complete feature set for modern CMake techniques. Consider, for example, a project built on Windows but using the MinGW compiler. In the event that a project was relying on the old behavior, it could continue to do so using policy CMP0045 like so: # Allow non-existent target with get_target_property() cmake_policy(SET CMP0045 OLD) # Would halt with an error without the above policy change get_target_property(outVar doesNotExist COMPILE_DEFINITIONS) The need for setting a policy to NEW is less common. These generators ignore the CMAKE_BUILD_TYPE cache variable and instead require the developer to choose the build type within the IDE or with a command line option at build time. set(x 3) set(y 7) math(EXPR z "(${x}+${y}) / 2") message("result = ${z}") Expected output: result = 5 5.9. Recommended Practices Where the development environment allows it, the CMake GUI tool is a useful way to quickly and easily understand the build options for a project and to modify them as needed during development. • Just the library name without a path, usually also without any platform-specific file name prefix (e.g. lib) or suffix (e.g. .a, .so, .dll). All colon separators are replaced with semi-colons, thereby converting a PATH-like input into a CMake list of paths. These methods do not work for all build types, since LOCATION is not known at configure time for multi configuration generators and the build output directory structure is typically different across the various CMake generator types. This is the sort of situation the variable number of = signs in the start and end markers is meant to handle, allowing the markers to be chosen so that they do not match anything in the content they surround. While not required, the prevailing convention is for keywords to be all uppercase, with words separated by underscores if required. The result of the retrieval is stored in a variable whose name is given by resultVar. The OUTPUT and TARGET forms also have slightly different behavior when it comes to appending more dependencies or commands to the same output file or target. A simple example might be a module file which sets a few policies temporarily like so: # Save existing policy state cmake_policy(PUSH) # Set some policies to OLD to preserve a few old behaviors cmake_policy(SET CMP0060 OLD) # Library path linking behavior cmake_policy(SET CMP0021 OLD) # Tolerate relative INCLUDE_DIRECTORIES # Do various processing here... The total number of bytes read can be limited using LIMIT_INPUT, while the total number of bytes stored can be limited using LIMIT_OUTPUT. Multiple -D options can be provided to set more than one variable at a time on the cmake command line. As usual, the book is available exclusively from 8 Likes VDOC.PUB Download Embed This document was uploaded by our user. 233   22.8. Limitations . If the output files are specified with no path or with a relative path, they are relative to the current binary directory. set(input /some/path/foo.bar.txt) get_filename_component(path1 get_filename_component(path2 get_filename_component(fullName get_filename_component(baseName get_filename_component(extension ${input} ${input} ${input} ${input} ${input} DIRECTORY) PATH) NAME) NAME_WE) EXT) # # # # # /some/path /some/path foo.bar.txt foo .bar.txt The second form of get_filename_component() is used to obtain an absolute path: get_filename_component(outVar input component [BASE_DIR dir] [CACHE]) In this form, input can be a relative path or it can be an absolute path. It always selects the right command for the host platform. Note, however, that the TIMEOUT option on its own won't cause CMake to halt and report an error. Most projects will rely on CMake generating such files for them though, so the need to do this should be fairly uncommon. This is repeated until condition evaluates to false or the loop is exited early (see next section). 2.4. Running The Build Tool At this point, with project files now available, the developer can use their selected build tool in the way to which they are accustomed. Only COMMAND and DEPENDS can be used for the second and subsequent calls for the same output file, the other options such as MAIN_DEPENDENCY, WORKING_DIRECTORY and COMMENT are ignored when the APPEND keyword is present. 321   25.7. Writing A Config Package File . These variables should ideally be set just after the first project() command in the top level CMakeLists.txt file. Finding Things . A further restriction of this command compared to the more generic set_property() is that it always applies to the current directory. If a value is the same as the name of a defined variable, it will be treated as a variable. For generators which do not support multiple configurations, the developer has to re-run CMake to switch the build between Debug, Release, etc. This allows optional steps to be defined and invoked when needed instead of always being run, which can be quite useful if the additional steps are time consuming or won't always be relevant. The people, the tools and the processes in place made working on the project truly a pleasure. It is generally the build type used when building software for final production releases. For this reason, prefer to use add_custom_target() or add_custom_command() instead of execute_process() if there is a choice. For such cases, CMake provides the COMPILE_DEFINITIONS, COMPILE_FLAGS and COMPILE_OPTIONS source file properties (the COMPILE_OPTIONS source file property was only added in CMake 3.11). They are controlled by the following family of variables: • CMAKE__LINKER_FLAGS • CMAKE__LINKER_FLAGS_ These are specific to a particular type of target, each of which was introduced back in Chapter 4, Building Simple Targets. For cache variables, consider grouping related variables under a common prefix followed by an underscore to take advantage of how CMake GUI groups variables based on the same prefix automatically. The following shows a number of substitution examples: CMakeLists.txt set(FOO "String with spaces") configure_file(various.txt.in various.txt) 159 various.txt.in CMake version: ${CMAKE_VERSION} Substitution works inside quotes too: "${FOO}" No substitution without the $ and {}: FOO Empty ${} specifier gets removed Escaping has no effect: \${FOO} @-syntax also supported: @ various.txt CMake version: 3.7.0 Substitution works inside quotes too: "String with spaces" No substitution without the $ and {}: FOO Empty specifier gets removed Escaping has no effect: \String with spaces @-syntax also supported: String with spaces The ESCAPE_QUOTES keyword can be used to cause any substituted quotes to be preceded with a backslash. Generator expressions should only be used for the value part of a definition, not for the name part (i.e. only after the "=" in a -DVAR=VALUE item or not at all for a -DVAR item). Again, this harms readability and offers little benefit, so specifying the loopVar with endforeach() is discouraged for new projects. This is the case even if the code being tested is changed, so to force re-evaluation, the variable has to be manually removed from the cache. 363   27. That existing target can be an executable or library target, or it can even be a custom target (with some restrictions). Toolchains And Cross Compiling . The primary method for performing such operations is the get_filename_component() command, which has three different forms. The file name passed to include() typically has the extension .cmake, but it can be anything. 126 Features specified as PUBLIC will be added to both properties and will therefore be applied to both the target itself and to any other target which links to it. I frequently wrote about areas that came up in my interaction with colleagues at work or in my own development activities and I found this to be both rewarding and useful to others. The build directory will contain the necessary project files which can be loaded into an IDE, read by command line tools, etc. This is the variable to use where the directory of the current file being processed is required, no matter how it has been added to the build. endif() Very early versions of CMake required expression1 to be repeated as an argument to the else() and endif() clauses, but this has not been required since CMake 2.8.0. While it is still not unusual to encounter projects and example code using that older form, it is discouraged for new projects since it can be somewhat confusing to read. The CMake community wouldn't be as strong and as vibrant as it is today without the ongoing support of Kitware and its staff, both past and present. I don't understand how you do it, but I do like the end result! In every book's acknowledgments section, the author invariably thanks family members and spouses, and there's good reason for that. PRIVATE features populate the COMPILE_FEATURES property, which is applied to the target itself. It can be easy to omit the DEPENDS section by mistake, since the first build will run the custom command to create the missing output files and the build will appear to be behaving correctly. Most of the relevant properties for any imported target have names beginning with IMPORTED, but for executables, IMPORTED_LOCATION and IMPORTED_LOCATION_ are the most important. 257   23.6. Recommended Practices . The optional keywords can be used to retrieve details about the property other than just its value: DEFINED The result of the retrieval will be a boolean value indicating whether or not the named property has been defined. Supported values include C, CXX, Fortran, ASM, Java and others. 58 Chapter 8. This would also alter the compiler and linker flags for the already defined targets in the current directory scope. The modification to myVar in the child scope is not visible to the parent. The targetName is used within the CMakeLists.txt file to refer to the library, with the name of the built library on the file system being derived from this name by default. The main reason for this is that cache variables are primarily intended as a customization point for developers. • Copying files. For the full list of supported subcommands, including those added in later CMake versions, the reader should consult the CMake documentation. Rather than distributing functions and macros throughout the source tree, a common practice is to nominate a particular directory (usually just below the top level of the project) where various XXX.cmake files can be collected. Projects are strongly encouraged to define namespaced aliases at least for all targets that are intended to be installed/packaged. These needs are well covered by CMake, offering the traditional behavior with some additions to make working with CMake features a little easier. add_library(targetName INTERFACE [IMPORTED [GLOBAL]]) All the various target_...() commands can be used with their INTERFACE keywords to define the usage requirements the interface library will carry. WARNING CMake warning, usually shown highlighted in red where supported (cmake command line console or the cmake-gui log area). Such a function is frequently encountered when writing test cases (a topic covered in Chapter 24, Testing). Where no substitution is needed, another alternative is to use the file() command with either the COPY or INSTALL form, both of which support the same set of options: file(fileOrDir1 [fileOrDir2...]   DESTINATION dir   [NO_SOURCE_PERMISSIONS | USE_SOURCE_PERMISSIONS | [FILE_PERMISSIONS permissions...]   [DIRECTORY_PERMISSIONS permissions...]]   [FILES_MATCHING]   [PATTERN pattern | REGEX regex] [EXCLUDE]   [PERMISSIONS permissions...]   [...]) 161 Multiple files or even entire directory hierarchies can be copied to a chosen directory, even preserving symlinks if present. The following would write out a single header called foo_compiler_detection.h in the build directory and start each macro name with the string foo_: include(WriteCompilerDetectionHeader) write_compiler_detection_header(FILE foo_compiler_detection.h   PREFIX foo   COMPILERS GNU Clang MSVC Intel   FEATURES cxx_override   cxx_final   cxx_nullptr   cxx_rvalue_references) Example C++ code making use of the macros defined by the above might look like this: #include "foo_compiler_detection.h" class MyClass foo_FINAL : public Base { public: #if foo_COMPILER_CXX_RVALUE_REFERENCES   MyClass(MyClass&& c); #endif   void func1() foo_OVERRIDE;   void func2(int* p = foo_NULLPTR); }; The target consuming the above source file would still need to have the appropriate language standard selected, but in this case, since fall back implementations are available, the desired standard can be specified but not required: 129 set(CMAKE_CXX_STANDARD 11) set(CMAKE_CXX_STANDARD_REQUIRED OFF) set(CMAKE_CXX_EXTENSIONS OFF) add_library(foo MyClass.cpp) # The header is written to the build directory # so ensure we add that to the header search path target_include_directories(foo   PUBLIC "${CMAKE_CURRENT_BINARY_DIR}") CMake provides fall back implementations for quite a few features, all of which are described in the WriteCompilerDetectionHeader module documentation. If no mode keyword is provided, then the message is considered to be important information and is logged without any modification. 14.1.2. Linker Flags The target properties associated with linker flags have similarities to those for the compiler flags, but there are fewer properties involved: 112 LINK_LIBRARIES This property holds a list of all libraries the target should link to directly. Despite the size, complexity and technical nature of the material, I have been constantly amazed at how it has made self-publishing not just a viable option, but also an enjoyable experience with surprisingly few practical limitations. This is where the build tool's project files are created using the internal representation built up in the configure step. depending on the situation. variables for this last check cmake_reset_check_state() check_symbol_exists(__TIME__ "" HAVE_PPTIME) # Restore all CMAKE_REQUIRED_... One can also set the relevant INTERFACE_... properties directly with set_property() or set_target_properties(), but the target_...() commands are safer and easier to use. 22 set(foo ab) set(bar ${foo}cd) set(baz ${foo} cd) set(myVar ba) set(big "${${myVar}r}ef") set(${foo} xyz) set(bar ${notSetVar}) # # # # # # # foo bar baz myVar big ab bar = = = = = = = "ab" "abcd" "ab;cd" "ba" "${bar}ef" = "abcdef" "xyz" "" Strings are not restricted to being a single line, they can contain embedded newline characters. Since no two calls to project() can use the same projectName, these project-specific variables won't be overwritten by other calls to the project() command. Only consider modifying the CMAKE_CONFIGURATION_TYPES variable if it is known that a multi configuration generator is being used or if the variable already exists. 92 In situations where multiple checks are being made or where the effects of performing the checks need to be isolated from each other or from the rest of the current scope, it can be cumbersome to manually save and restore the state before and after the checks. The most basic list operations are counting the number of items and retrieving one or more items from the list: list(LENGTH listVar outVar) list(GET listVar index [index...] outVar) # Example set(myList a b c) # Creates the list "a;b;c" list(LENGTH myList len) message("length = ${len}") list(GET myList 2 1 letters) message("letters = ${letters}") The output of the above example would be: length = 3 letters = c;b Appending and inserting items is also a common task: list(APPEND listVar item [item...]) list(INSERT listVar index item [item...]) Unlike the LENGTH and GET cases, APPEND and INSERT act directly on the listVar and modify it in-place, as demonstrated by the following example: 35 set(myList a b c) list(APPEND myList d e f) message("myList (first) = ${myList}") list(INSERT myList 2 X Y Z) message("myList (second) = ${myList}") Which gives the following output: myList (first) = a;b;c;d;e;f myList (second) = a;b;X;Y;Z;c;d;e;f Finding a particular item in the list follows the expected pattern: list(FIND myList value outVar) # Example set(myList a b c d e) list(FIND myList d index) message("index = ${index}") Expected output: index = 3 Three operations are provided for removing items, all of which modify the list directly: list(REMOVE_ITEM myList value [value...]) list(REMOVE_AT myList index [index...]) list(REMOVE_DUPLICATES myList) The REMOVE_ITEM operation can be used to remove one or more items from a list. Furthermore, if using any of the Check... modules, aim to isolate the checking logic from the scope in which it may be invoked. In the case of the custom Profile build type defined above, an example might be: set(CMAKE_PROFILE_POSTFIX _profile) If creating packages that contain multiple build types, setting CMAKE__POSTFIX for each build type is highly recommended. 413   Preface A few years ago, I was surprised at the lack of published material for learning how to use CMake. The initial focus in the first few chapters is on building a basic executable or library, covering just enough to give a new developer a quick introduction to CMake. Developers at the beginning of their career often start out using this approach because of the perceived simplicity. • Consider whether using IMPLICIT_DEPENDS or DEPFILE options with add_custom_command() is worth the generator-specific behavior. Note, however, that only the equivalent of the _STANDARD and _STANDARD_REQUIRED target property behaviors are provided, so the _EXTENSIONS target property or CMAKE__EXTENSIONS variable should still be used to control whether or not compiler extensions are allowed. REALPATH Compute the absolute path of input with symbolic links resolved. 43 if(DEFINED name) if(COMMAND name) if(POLICY name) if(TARGET name) if(TEST name) # Available from CMake 3.4 onward Each of the above will return true if an entity of the specified name exists at the point where the if command is issued. someValue will replace any previous value of the myVar cache variable. The name may also contain the characters ./-+ but these are rarely seen in practice. • Avoid direct manipulation of the target and directory properties that affect compiler and linker behavior. The name is the only mandatory argument for the project() command. The effect of calling return() inside a function is covered in Section 8.4, "Scope", including special attention for a common mistake that can result in returning from the current file unintentionally. Introduction Whether a seasoned developer or just starting out in a software career, one cannot avoid the process of becoming familiar with a range of tools in order to turn a project's source code into something an end user can actually use. Those config files would define imported libraries with the namespaced names rather than the bare original names. Some older projects may use EXPECTED_MD5 as an alternative to EXPECTED_HASH MD5=..., but new projects should prefer the EXPECTED_HASH form. add_library(targetName (STATIC | SHARED | MODULE | OBJECT | UNKNOWN)   IMPORTED [GLOBAL]) The library type must be given immediately after the targetName. Furthermore, if the source is already present at the destination with the same timestamp, the copy for that file is deemed as already having been done and will be skipped. 95 Chapter 12. An example best illustrates the behavior: CMakeLists.txt set(myVar foo) message("Parent (before): myVar = ${myVar}") message("Parent (before): childVar = ${childVar}") add_subdirectory(subdir) message("Parent (after): message("Parent (after): myVar = ${myVar}") childVar = ${childVar}") 52 subdir/CMakeLists.txt message("Child message("Child (before): myVar = ${myVar}") (before): childVar = ${childVar}") set(myVar bar) set(childVar fuzz) message("Child message("Child (after): (after): myVar = ${myVar}") childVar = ${childVar}") This produces the following output: Parent Parent Child Child Child Child Parent Parent (before): (before): (before): (before): (after): (after): (after): (after): myVar childVar myVar childVar myVar childVar myVar childVar = = = = = = = = foo foo bar fuzz foo ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ① myVar is defined at the parent level. This is achievable with the help of the $ generator expressions. This is intended for more complex projects where it may not be clear how a variable ended up with a particular value. Before CMake 3.11, none of the target_...() commands could be used to set INTERFACE_... properties on any kind of IMPORTED library. The above could alternatively be implemented like so: cmake_minimum_required(VERSION 3.0) project(PolicyExample) # Only set the policy if the version of CMake being used # knows about that policy number if(POLICY CMP0055)   cmake_policy(SET CMP0055 NEW) endif() It is also possible to get the current state of a particular policy. 175   19.2. Source Code Access To Version Details . Consider the following example: add_custom_command(OUTPUT transformed.cpp   COMMAND transform ${CMAKE_CURRENT_SOURCE_DIR}/original.cpp   transformed.cpp   MAIN_DEPENDENCY ${CMAKE_CURRENT_SOURCE_DIR}/original.cpp) add_executable(original original.cpp) add_executable(transformed transformed.cpp) The above would lead to a linker error for the original target because original.cpp would not be compiled to an object file, so there would be no object files at all (and therefore no main() function). Irrespective of whether using add_subdirectory(), include() or a combination of both, the CMAKE_CURRENT_LIST_DIR variable is generally going to be a better choice than CMAKE_CURRENT_SOURCE_DIR. These and other details are covered in more detail in Section 22.2, "Application Bundles". The CMake documentation for try_compile() discusses additional variables which may have an effect on the checks, while other aspects of try_compile() behavior relating to toolchain selection are covered in Section 21.5, "Compiler Checks". When multiple input strings are given, they are joined together without any separator between each string before searching for substitutions. It is then safe to append flags to the current value because the cache variable is left untouched, so every CMake run starts with the same set of flags from the cache variable, regardless of how often CMake is invoked. The CACHE keyword is optional. It is not possible for CMake to know the properties of a directory scope it has not yet encountered. The command will set variables with names of the form CMAKE_MATCH_ where is the group to match. 17.2. Adding Build Steps To An Existing Target Custom commands sometimes do not require a new target to be defined, they may instead specify additional steps to be performed when building an existing target. It is the most suitable approach when the project's developers know which language version provides the features used by the project's code. This makes it trivial to turn them on and off in the CMake GUI and other tools which understand how to parse CMakeLists.txt files (a growing number of IDE environments are acquiring this capability). 12.3. Recommended Practices Where possible, projects should prefer to work with policies at the CMake version level rather than manipulating specific policies. Rather, the directory and target properties are combined in the final compiler command line. POLICY Tests whether a particular policy is known to CMake. • Try to avoid using PRE_BUILD with the TARGET form of add_custom_command(). 4.5. Old-style CMake The target_link_libraries() command also has a few other forms, some of which have been part of CMake from well before version 2.8.11. The first project() command encountered will create cache variables for these if they don't already exist (this is a bit of a simplification, a more complete explanation is given in Chapter 21, Toolchains And Cross Compiling).   # Make Unix shell scripts executable by everyone   PATTERN *.sh PERMISSIONS   OWNER_READ OWNER_WRITE OWNER_EXECUTE   GROUP_READ GROUP_EXECUTE   WORLD_READ WORLD_EXECUTE   # Ensure only owner can read/write private key files   REGEX _dsa\$|_rsa\$ PERMISSIONS   OWNER_READ OWNER_WRITE) CMake offers a third option for copying files and directories. If choosing to do this, it would be wise for projects to use some project-specific prefix on the property name to avoid potential name clashes with properties defined by CMake or other third party packages. The value of an environment variable is obtained using the special form $ENV{varName} and this can be used anywhere a regular ${varName} form can be used. 23 Chapter 18, Working With Files includes further examples which highlight situations where bracket syntax can be a better alternative. String comparisons are one of the most common situations where such unexpected substitutions occur. Furthermore, no further transformation of the bracketed content is performed (i.e. no variable substitution or escaping). Those expressions mentioned above are only a subset of what CMake supports, but they form a good foundation for covering the majority of situations most developers are likely to face. These features were added progressively from CMake 2.8.11 through to 3.2 and lead to considerably simpler and more robust CMakeLists.txt files. Rather than referring to CMAKE_BUILD_TYPE in the CMakeLists.txt file, projects should instead use other more robust alternative techniques, such as generator expressions based on $. The paths provided to include_directories() can be relative or absolute. 1   Part I: Fundamentals .... GLOBAL ensures the command ends processing of the file if it has been processed before anywhere else in the project (i.e. variable scope is ignored). Therefore, features like input/output redirection and environment variables are not supported. Other keywords such as nullptr and static_assert have alternative implementations which can be used if the keyword is not supported by the compiler. This is particularly useful for the COMPILE_OPTIONS property, since it enables only adding a particular flag if some condition is met, such as only for a particular compiler. STATIC Targets created with add_library(name STATIC ...) or equivalent, such as omitting the STATIC keyword but with the BUILD_SHARED_LIBS variable set to false or not defined. In all of the following examples, the expression evaluates to true: if(1.2 if(1.2 if(1.2.3 if(2.0.1 if(1.8.2 VERSION_EQUAL VERSION_LESS VERSION_GREATER VERSION_GREATER VERSION_LESS 1.2.0) 1.2.3) 1.2) 1.9.7) 2) The version number comparisons have the same robustness caveats as numeric comparisons. 24   5.3. Cache Variables . Projects should take advantage of these features to let CMake provide locations of targets rather than hard-coding paths manually, as this allows the project to robustly support all platforms and generator types with minimal effort. The second form of the include() command serves an entirely different purpose. GUI tools typically use the docstring as a tooltip for the cache variable or as a short one line description when the variable is selected. 13   3.5. Recommended Practices . Understanding these areas well can be the difference between a fragile, complex project and a robust, easy to maintain one. Success or failure of the set of commands can be captured using the RESULT_VARIABLE option. From CMake 3.12, object libraries can be used with target_link_libraries(), either as the target being added to (i.e. the first argument to the command) or as one of the libraries being added. 289   24.10. if("Hi from ${who}" MATCHES "Hi from (Fred|Barney).*")   message("${CMAKE_MATCH_1} says hello") endif() 6.1.4. File System Tests CMake also includes a set of tests which can be used to query the file system. The first command does not need to have the COMMAND keyword preceding it, but for clarity it is recommended to always include the COMMAND keyword even for the first command. 7.3. Ending Processing Early There can be occasions where a project may want to stop processing the remainder of the current file and return control back to the caller. The later parts of the book all rely heavily on the material covered in this first part. add_executable(generator generator.cpp) add_custom_command(OUTPUT onTheFly.cpp   COMMAND generator) add_executable(myExe ${CMAKE_CURRENT_BINARY_DIR}/onTheFly.cpp) CMake automatically recognizes that myExe needs the source file generated by the custom command, which in turn requires the generator executable. This is especially relevant when the destination file is a C/C++ header and the current source and binary directories are both on the header search path. From the first CMake version supporting this feature, valid values for C_STANDARD are 90, 99 or 11, while for CXX_STANDARD the valid values are 98, 11 or 14. These commands behave just like their similarly named C language counterparts and both operate only on the inner-most enclosing loop. Developers can still set CMAKE_BUILD_TYPE to any value at the cmake command line or edit the CMakeCache.txt file manually. 124   15.1. Setting The Language Standard Directly . They have the same values as described above even for a CMakeLists.txt file pulled in via add_subdirectory(), in which case CMAKE_CURRENT_LIST_DIR would have the same value as CMAKE_CURRENT_SOURCE_DIR. _EXTENSIONS Many compilers support their own extensions to the language standard and a compiler and/or linker flag is usually provided to enable or disable those extensions. It changes for every call to add_subdirectory() and is restored again when add_subdirectory() returns. An empty build type is its own unique, nameless build type. • Computing a checksum or hash of a file. If present, the result is stored as a cache variable rather than a regular variable. The algo and engine libraries, on the other hand, are linked to collector as PRIVATE, so myApp will not be directly linked to them. Specifying a command with variable_watch() would be very uncommon though. Typical commands involve running a script or a system-provided executable, but they can also run executable targets created as part of the build. Also, since no particular shell behavior is guaranteed, escaping within the executable names or their arguments may be handled differently on different platforms. 239   23. When choosing how to specify the CMake version to conform to, the choice between cmake_minimum_required(VERSION) and cmake_policy(VERSION) would usually fall to the latter. 366   27.1.1. Tour Of Main Features . A build type is considered to be a debug build if it is listed in the DEBUG_CONFIGURATIONS global property, otherwise it is considered to be optimized. 13.3. Custom Build Types Sometimes a project may want to limit the set of build types to a subset of the defaults, or it may want to add other custom build types with a special set of compiler and linker flags. The source directory is where the CMakeLists.txt file is located and the project's source files and all other files needed for the build are organized under that location. Each time the configure step is initiated, the cache variables shown in the middle of the UI are updated. For custom build types, they should have their name added to this global property if they should be treated as a debug build in this scenario. This can seem somewhat unintuitive, but consider the situation if the variable was named as the entity along with the VARIABLE keyword, just like for the other entity type keywords. TARGET Returns true if a CMake target of the specified name has been defined by one of the commands add_executable(), add_library() or add_custom_target(). Consider a function that defines an executable target, links that target to some library and defines a test case for it. The exception to this is that when setting a cache variable's value, if that cache variable did not exist before the call to set() or if the FORCE option was used, then any normal variable in the current scope is effectively removed. 128   15.3. Recommended Practices . 8.1. The Basics Functions and macros in CMake have very similar characteristics to their same-named counterparts in C/C++. A normal variable can be defined in a CMakeLists.txt file as follows: set(varName value... CMake will then interpret the commands as a recipe for generating the named output files. As far as variables are concerned, the function is essentially its own self-contained sandbox, unlike macros which share the same variable scope as their caller. This is the approach used by cmake_parse_arguments(), with its prefix argument determining the prefix of all the variable names it sets in the caller's scope. Object libraries have their uses, but they are also less flexible than static libraries. Note how the entire value given with the -D option should be quoted if setting a cache variable with a value containing spaces. The OUTPUT_FILE and ERROR_FILE options can be used to specify the names of files to send the streams to and just like the variable-focused options, specifying the same file name for both results in a merged stream. An internal set of failure regular expressions is also passed as the FAIL_REGEX option, testing whether the flag results in a diagnostic message being issued or not. While it is possible to use command mode in conjunction with execute_process() to copy files at configure time, there is little reason to do so, since configure_file() and file() are both more direct and have the added benefit that they stop on any error automatically. In such a case, the value of the property was returned as -NOTFOUND rather than issuing an error, but in all likelihood, the project probably contained incorrect logic. target_link_libraries(main PRIVATE bart) # Typo in name being linked to, CMake flags an error # at generation time because a namespaced name must # be a CMake target. They can be particularly useful in larger, more complex projects where some parts might or might not be present or be enabled. Instead of using COMMENT, projects can use the -E echo command to intersperse comments anywhere in a sequence of custom commands: set(discardDir "${CMAKE_CURRENT_BINARY_DIR}/private") add_custom_target(myCleanup   COMMAND ${CMAKE_COMMAND} -E echo "Removing ${discardDir}"   COMMAND ${CMAKE_COMMAND} -E remove_directory "${discardDir}"   COMMAND ${CMAKE_COMMAND} -E echo "Recreating ${discardDir}"   COMMAND ${CMAKE_COMMAND} -E make_directory "${discardDir}") CMake's command mode is a very useful way of carrying out a range of common tasks in a platform independent way. When the custom target is built, the specified command(s) will be executed in the order given, with each command able to have any number of arguments. Internal cache variables are sometimes used to persistently record internal information by the project, such as caching the result of an intensive query or computation. endif() 45 The above examples are both cases of testing the platform instead of the entity the constraint actually relates to. One might reasonably expect the following code to behave as noted in the inline comments: 121 # Save the original set of flags so we can restore them later set(oldCxxFlags "${CMAKE_CXX_FLAGS}") # This library has stringent build requirements, so enforce them just for it alone # WARNING: This doesn't do what it may appear to do! set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Werror") add_library(strictReq STATIC ...) # Less strict requirements from here, so restore the original set of compiler flags set(CMAKE_CXX_FLAGS "${oldCxxFlags}") add_library(relaxedReq STATIC ...) It may be surprising to learn that with the arrangement above, the strictReq library will not be built with -Wall -Werror flags. While CMake's automatic dependencies may seem convenient, in practice the project will still typically need to list out all the required targets and files in a DEPENDS section to ensure that the full dependency relationships are adequately specified. In practice, just about any logic based on CMAKE_BUILD_TYPE within a project is questionable unless it is protected by a check to confirm a single configuration generator is being used. Projects employ modules in one of two ways, either directly or as part of finding an external package. Each library listed can be one of the following: • A path to a library, usually specified as an absolute path. Such details should come from outside the project, such as through environment variables (still somewhat insecure), files found on the user's file system with 173 appropriate permissions limiting access or a keychain of some kind. Arguments are separated from each other by spaces and may be split across multiple lines: add_executable(myExe   main.cpp   src1.cpp   src2.cpp) Command names are also case insensitive, so the following are all equivalent: add_executable(myExe main.cpp) ADD_EXECUTABLE(myExe main.cpp) Add_Executable(myExe main.cpp) Typical style varies, but the more common convention these days is to use all lowercase for command names (this is also the convention followed by the CMake documentation for built-in commands). • Obtaining a list of files from a directory. This older property has been deprecated since CMake 2.8.12, but policy CMP0022 can be used to give the old property precedence if needed. Each of the keyword arguments is a list of keyword names supported by that function or macro, so they should each be surrounded by quotes to ensure they are parsed correctly. Also consider specifying a TIMEOUT, INACTIVITY_TIMEOUT or both to prevent a configure run from blocking indefinitely if network connectivity is poor or unreliable. Later chapters provide specific scenarios where these commands are used with great effect. TARGET_FILE_DIR Same as TARGET_FILE but without the file name. The target_compile_features() command makes this easy and clearly specifies whether such requirements are PRIVATE, PUBLIC or INTERFACE. The CheckFortranFunctionExists module may be useful for those projects working with Fortran, but note that there is no CheckFortranVariableExists module. The part is an uppercase string corresponding to one of the build types, such as DEBUG, RELEASE, RELWITHDEBINFO or MINSIZEREL. To address situations like these, CMake provides an additional set of variables: CMAKE_CURRENT_LIST_DIR Analogous to CMAKE_CURRENT_SOURCE_DIR except it will be updated when processing the included file. 75   9.6. Cache Variable Properties . 21 Chapter 5. The behavior of relative paths was left unspecified and usually ended up being relative to the working directory of when cmake was invoked. The official reference documentation was a useful resource for those willing to go exploring, but as a way of learning CMake in a progressive, structured manner it was not ideal. Since the value of CMAKE_CURRENT_SOURCE_DIR does not change when include() is called, it may seem difficult for the included file to work out the directory in which it resides. 2.3. Generating Project Files Once the choice of directory structure has been made, the developer runs CMake, which reads in the CMakeLists.txt file and creates project files in the build directory. The names of these modules have the form CheckSourceCompiles and each provides an associated macro to perform the test: include(CheckCSourceCompiles) check_c_source_compiles(code resultVar [FAIL_REGEX regex]) include(CheckCXXSourceCompiles) check_cxx_source_compiles(code resultVar [FAIL_REGEX regex]) include(CheckFortranSourceCompiles) check_fortran_source_compiles(code resultVar [FAIL_REGEX regex] [SRC_EXT extension]) For each of the macros, the code argument is expected to be a string containing source code that should produce an executable for the selected language. Projects do not often use this particular type of message. Furthermore, generator expressions are supported by the underlying directory and target properties, so the add_compile_options() command also supports them. --config Release --target MyApp If the developer wishes to experiment with different generators, all that needs to be done is change the argument given to the -G CMake option and the correct build tool will be automatically invoked. Each argument in the function definition serves as a case-sensitive label for the argument it represents. which may want to enable or disable certain parts of the build. The basic form introduced back in Chapter 4, Building Simple Targets can be used to define the usual types of libraries most developers are familiar with, or it can also be used to define object libraries which are just a collection of object files that are not combined into a single archive or shared library. 294   25. If no variable exists with that name, an empty string is substituted. add_subdirectory(sourceDir [binaryDir] [EXCLUDE_FROM_ALL]) The sourceDir does not have to be a subdirectory within the source tree, although it usually is. It is not permitted to specify a file as the source and an existing directory for the destination. COMPILE_OPTIONS Any compiler flags that are neither header search paths nor symbol definitions are provided in 111 this property. The developer therefore needs some assistance from CMake to determine the directories of interest. Execution never reaches the first implementation. Also prefer to add and remove individual items rather than completely replacing the list. CMAKE_CURRENT_SOURCE_DIR will contain the location of the file from where include() was called, not the directory containing the included file. For example, suppose a directory myDir contains a file someFile.txt and the following command was issued: cmake -E copy_directory myDir targetDir The result of this command would be that targetDir would contain the file someFile.txt, not myDir/someFile.txt. For example, to produce the string OLD_COMPILER if the C++ compiler version is less than 4.2.0, the following expression could be used: $ Such expressions tend to be used only in situations where the type of compiler is known and a specific behavior of the compiler needs to be handled in some special way by the project. The details of what include_directories() actually does is more complex than the simplistic explanation above. Prefer to provide cache variables for controlling whether to enable optional parts of the build instead of encoding the logic in build scripts outside of CMake. The set_target_properties() command does not support appending to existing property values and if a list value needs to be provided for a given property, the set_target_properties() command requires that value to be specified in string form, e.g. "this;is;a;list". Apple Features . Once CMake has finished reading and processing the CMakeLists.txt file, it then performs the generation step. 2.1. In-source Builds It is possible, though discouraged, for the source and build directories to be the same. 255   23.5.2. FindPkgConfig . These variables will be overwritten by the next command that does any sort of regular expression operation. 9.6. Cache Variable Properties Properties on cache variables are a little different in purpose to other property types. The APPEND and APPEND_STRING keywords can be used to control how the named property is updated if it already has a value. Unfortunately, it is common to see such expressions based on the wrong variables. As such, directory properties mostly focus on setting defaults for target properties and overriding global properties or defaults for the current directory. One could be forgiven for thinking that the number of different interface and imported library combinations is overly complicated and confusing. In practice, it is usually easier and more robust to use the variables to set language requirements project-wide rather than use per target properties. Some IDE environments understand this convention and will still provide appropriate syntax highlighting on the source file based on the file's extension without the .in suffix. This fairly advanced topic is covered in CMake's documentation for the try_run() command, which is what the macros use internally to perform the checks. Unfortunately, for some CMake versions and project generators, it doesn't always act as expected and can even result in broken builds. Fortran projects may want to use CheckFortranSourceCompiles for consistency instead. The cmake_reset_check_state() macro is a convenience for clearing all the CMAKE_REQUIRED_... variables and the RESET option to cmake_push_check_state() is also just a convenience for clearing the variables as part of the push. In the following examples, only the if(...) part of the command is shown for illustration purposes, the corresponding body and endif() is omitted: # Examples of unquoted constants if(YES) if(0) if(TRUE) # These are also treated as unquoted constants because the # variable evaluation occurs before if() sees the values set(A YES) set(B 0) if(${A}) # Evaluates

to true if(${B}) # Evaluates to false # Does not match any of the true or false constants, so proceed # to testing as a variable name in the fall through case below if(someLetters) # Quoted value, so bypass the true/false constant matching # and fall through to testing as a variable name or string if("someLetters") The CMake documentation refers to the fall through case as the following form: if() What this means in practice is the if-expression is either: • An unquoted name of a (possibly undefined) variable.) This allows projects to precisely define how one library depends on others. When the location of the imported executable is needed, CMake will first look at the configuration-specific property and only if that is not set will it look at the more generic IMPORTED_LOCATION property. The PROJECT_... variables, on the other hand, are updated every time project() is called, so they can be used to provide the version details of the most recent call to project() in the current scope or above. The following general guide may be useful: • Where possible, prefer to use the target_...() commands to describe relationships between targets and to modify compiler and linker behavior. The following expressions are supported: if(EXISTS pathToFileOrDir) if(IS_DIRECTORY pathToDir) if(IS_SYMLINK fileName) if(IS_ABSOLUTE path) if(file1 IS_NEWER_THAN file2) The above should largely be self-explanatory, but there are some points to be aware of. Additional details about the checks can be found in subdirectories within the build area, but developers would typically only need to look there if working with a new or unusual compiler or when setting up toolchain files for cross-compiling. Do not prefix the library names with any -l option or similar, provide just the library name or the name of a CMake imported target (discussed in Chapter 16, Target Types). LINK_FLAGS This holds a list of flags to be passed to the linker for targets that are executables, shared libraries or module libraries. It should also be noted that prior to CMake 3.11, adding custom build types to CMAKE_CONFIGURATION_TYPES was not technically safe. There are a few different forms of foreach(), the most basic of which is: foreach(loopVar arg1 arg2 ...) # ... A few read-only directory properties also provide a degree of introspection, holding information about how the build reached the directory, what things have been defined at that point, etc. In fact, it is quite common for a custom target to be defined simply to provide a way for the developer to trigger the custom command. else() # WARNING: Intended to call the original command, but it is not safe _someFunc() endif() endfunction() If the command is only ever overridden like this once, it appears to work, but if it is overridden again, then the original command is no longer accessible. The WriteCompilerDetectionHeader module defines a function called write_compiler_detection_header() which automates such handling. In such cases, a workable compromise may be to set the variable values in the top level CMakeLists.txt file as early as possible, ideally at the very top just after the cmake_minimum_required() command (or even better, in the toolchain file if one is being used - see Chapter 21, Toolchains And Cross Compiling for further details). 79 10.1. Simple Boolean Logic A generator expression is specified using the syntax $ where the content between the angle brackets can take a few different forms. 3.1. Managing CMake versions CMake is continually updated and extended to add support for new tools, platforms and features. add_executable(myExe main.cpp) add_custom_command(TARGET myExe POST_BUILD COMMAND script1 $) # Additional command which will run after the above from a different directory add_custom_command(TARGET myExe POST_BUILD COMMAND writeHash $ BYPRODUCTS ${CMAKE_BINARY_DIR}/verify/myExe.md5 WORKING_DIRECTORY ${CMAKE_BINARY_DIR}/verify) 145 17.3. Commands That Generate Files Defining commands as additional build steps for a target covers many common use cases. It is also possible to silently discard the content of a stream altogether with the OUTPUT_QUIET and ERROR_QUIET options. Much of CMake's file-related functionality is provided by the file() command, with a few other commands offering alternatives better suited to certain situations or providing related helper capabilities. Two special library types IMPORTED and INTERFACE are discussed later in Chapter 16, Target Types. In that situation, there would be nothing to specify for the property name. In very early versions of CMake, include() and find_package() did not automatically push and pop an entry on the policy stack. It does not create a new build target with the alias name. 6.1.2. Logic Operators CMake supports the usual AND, OR and NOT logical operators, as well as parentheses to control order of precedence. ② childVar is not defined at the parent level, so it evaluates to an empty string. 139 17. This has the added advantage that, unlike specifying the standard directly, compile feature requirements can be part of a target's interface and therefore can be enforced on other targets linking to it. This is the representation used by CMake for paths on all platforms. • The caller can simply omit those arguments which don't need to be provided. The maximum number of bytes to read can also be limited with the LIMIT option. The cmake_minimum_required(VERSION...) and cmake_policy(VERSION...) commands reset the state of all policies. This means it can only be used to set or replace a property, it cannot be used to add to an existing property directly. Booleans are shown as a checkbox, files and paths have a browse filesystem button and strings are usually presented as a text line edit. 307 25.2.2. RPATH . CMakeLists.txt is just an ordinary text file and is usually edited directly, just like any other source file in the project. 132 16.2. Libraries . All of the options to include() work exactly as before. If BUILD_SHARED_LIBS has been set to true, the library target will be a shared library, otherwise it will be static. Two fundamental CMake commands in any multi-directory project are add_subdirectory() and include(). # Unix example set(customPath /usr/local/bin:/usr/bin:/bin) file(TO_CMAKE_PATH ${customPath} outVar) # outVar = /usr/local/bin;/usr/bin;/bin 158 18.2. Copying Files The need to copy a file during the configure stage or during the build itself is a relatively common one. endfunction() # Later in the project... When downloading or uploading files with the file(DOWNLOAD...) or file(UPLOAD...) commands, security and efficiency aspects should be carefully considered. • Avoid listing a source file as a MAIN_DEPENDENCY in add_custom_command() unless the intention is to replace the default build rule for that source file. If an item is preceded by the optimized keyword, it will only be added if the build is not a debug build. Having to support multiple platforms adds more complications that can affect everything from the set of available tools through to the different capabilities available and restrictions enforced. In practice, this second form should rarely be needed and its use should be avoided for scenarios other than debugging the build or similar temporary tasks. One of the most common techniques used by those new to CMake is to add a compiler define at the top level of the project: cmake_minimum_required(VERSION 3.0) project(FooBar VERSION 2.4.7) add_definitions(-DFOOBAR_VERSION=\"${FooBar_VERSION}\") This makes the version available as a raw string able to be used like so: void printVersion() { std::cout E-Book Information Year: 2,018 Edition: 1 Pages: 429 Pages In File: 429 Language: English Topic: 92 Org File Size: 4,262,182 Extension: pdf Toc: Professional CMake......Page 1Table of Contents......Page 3Preface......Page 10Part I: Fundamentals......Page 12Chapter 1. 188 20.5. Symbol Visibility . While some of these features may work on some platforms, they are not universally supported. Recommended Practices......Page 303Chapter 25. Another developer on the same project may prefer to use a different generator because it integrates better with their IDE tool, or a future version of CMake may add support for a new generator type which might bring other benefits. 229 22.6. Code Signing . Understand what the properties do and how the different commands manipulate them, but prefer to use the more specialized target and directory specific commands where possible. Generator expressions like $ should be used instead, as they robustly provide the required path for all generators, whether they be single or multi configuration. The string() command has a CONFIGURE form which provides the same functionality: string(CONFIGURE input outVar [@ONLY] [ESCAPE_QUOTES]) The options have the same meaning as they do for configure_file(). Some of the returned names may correspond to undocumented or internal entities not intended for projects to use directly. The following example shows how to set the C standard details, this time only for a specific target: # Build target foo with C99, no compiler extensions set_target_properties(foo PROPERTIES C_STANDARD 99 C_STANDARD_REQUIRED ON C_EXTENSIONS OFF) It should be noted that _STANDARD technically specifies a minimum standard, not necessarily an exact requirement. Examples include things like downloading bootstrap files at configure time or bringing a file into the build which cannot or should not be stored as part of the project sources (e.g. sensitive files that should 172 only be accessible for certain developers, very large files, etc.). The most basic way of defining a variable is with the set() command. Table of Contents Preface. If neither the option nor the variable are defined, the default behavior is to not verify the server certificate. Another common error is to not create a dependency on a file that is needed by the custom command, but which isn't listed as part of the command line to be executed. This is the most robust way to obtain the directory in which the final executable or library is built. It is also straightforward to strip whitespace from the start and end of a string. Prefer file(COPY) instead for other situations. Target Types . Chapter 25, Installing covers much of the motivation and mechanics of the other combinations. This tends to limit the usefulness of compile features in general, with projects frequently choosing to set the language standard through the target properties described in the previous section instead. USES_TERMINAL is another console-related option which instructs CMake to give the command direct access to the terminal, if possible. All three keywords only affect the single item following it, not all items up to the next keyword.]]) In the first case above, any generator expressions in the content of the input.txt.in file will be evaluated when writing the output file. The variable can be modified by interactive GUI tools or by scripts without having to change anything in the project itself. To that end, CMake provides a number of variables which keep track of the source and binary directories for the CMakeLists.txt file currently being processed. GUI tools do not show INTERNAL variables. have no associated feature. It may also change other behaviors, such as altering which source files get compiled or what libraries to link to. 7.1. add_subdirectory() The add_subdirectory() command allows a project to bring another directory into the build. It is therefore recommended for developers to occasionally make a quick scan through the CMake documentation page listing the pre-defined variables to help become familiar with what is available. Other than that, the ccmake tool is a useful alternative when the full cmake-gui application is not practical or not available, such as over a terminal connection that cannot support UI forwarding. SET The result of the retrieval will be a boolean value indicating whether or not the named property has been set. The CMakeLists.txt files are the canonical description of the project and the generated project files should be considered part of the build output. This restriction is being actively discussed by the CMake developers and may be removed in a future release. These have a close relationship with the CTest and CPack tools distributed as part of the CMake suite and are covered in depth in Chapter 24, Testing and Chapter 26, Packaging. This is especially useful for scripted builds driven by continuous integration systems, etc. The set of build types known to multi configuration generators is controlled by the CMAKE_CONFIGURATION_TYPES cache variable, or more accurately, by the value of this variable at the end of processing the top level CMakeLists.txt file. Full paths should also be given when using IS_NEWER_THAN, since the behavior for relative paths is not well defined. C++ keywords such as final and override are common examples of this. 387 27.4. Recommended Practices . While script mode doesn't support command line options like ordinary shells or command interpreters, it does support passing in variables with -D options, just like ordinary cmake invocations. It always holds an absolute path to the file, not just the file name. This works just like the behavior already discussed back in Section 7.2, "include()" except that only the base name of the module needs to be given to the include() command, not the full path or file extension. The source file may need to surround a substituted variable with quotes if it needs to be treated as a single path or file name. 49 7. 362 26.4.9. FreeBSD . It is updated each time a new file is processed as a result of an add_subdirectory() call and is restored back again when processing of that directory is complete. It acts as a read-only way to refer to another library but does not create a new build target. For example, the following obtains a list with all lines in myStory.txt that contain either PKG_VERSION or MODULE_VERSION. Both of these measures will help avoid interfering with changes made to the cache variable by the developer. The entire matched string is stored in group 0. Such a directory structure might look something like this: If frequently switching between build types, this arrangement avoids having to constantly recompile the same sources just because compiler flags change. They are a critical part of the target_...() command set and they become even more important for the install and packaging stages of a project. In such cases, each compile feature would have to be listed out individually, which can be impractical and would likely be incomplete. # Platform differences are already handled by myObjLib. 39 6.1.1. Basic Expressions . When a developer commences work on a project, they must decide where they want their build directory to be in relation to their source directory. Custom commands should be defined as though they were being executed in isolation and without any shell features such as redirection, variable substitution, etc., with only command order being enforced. Similarly, if a TARGET, SOURCE or TEST property is requested but is not set for the 72 specified entity, the DIRECTORY scope will be searched (including recursively up the directory hierarchy and ultimately to the GLOBAL scope if necessary). The following shows an example of code which leads to such a warning: cmake_minimum_required(VERSION 2.8.12) set(FooBar_VERSION 2.4.7) project(FooBar) In the above, the FooBar_VERSION variable is explicitly set, but this variable name conflicts with the variable that the project() command would automatically define. Projects can specify relative directories too, not just absolute directories. 20 Unless there are strong reasons to do so, try to avoid specifying the STATIC or SHARED keyword for a library until it is known to be needed. These various concepts and features are covered in the next few chapters, but for now, the goal is just to get a simple build working as a starting point. 74 9.4. Target Properties Few things in CMake have such a strong and direct influence on how targets are built as target properties. Such namespaced aliases can even be used within the project itself, not just other projects consuming it as a pre-built package or child project. For static library targets, the commands will run before the library archiver tool. The projectName_... variables are the most robust, since they are always unambiguous in which project's details they will provide. A good strategy would be to ensure the project builds with the default generator type on each platform of interest, plus one other type. It is particularly convenient when logging information for multiple entities and/or properties. This work is sold with the understanding that the author is not held responsible for the results accrued from the advice in this book. 43 6.1.6. Common Examples . This in turn makes it easier to relocate CMake modules to different directories as a project evolves, or to re-use the logic across different projects. The following examples demonstrate the features discussed so far: 80 Expression $ Result Notes foo $ $ Error, not a 1 or 0 $ foo $ foo $ $ foo Error, NOT requires a 1 or 0 $ $ foo $ $ Result will be yes or no depending on ${foo} Just like for the if() command, CMake also provides support for testing strings, numbers and versions in generator expressions, although the syntax is slightly different. Some very basic use of variables and properties are also given to provide a flavor for how these CMake features relate to libraries and targets in general. Struct members can be tested with CheckStructHasMember, specific C or C++ function prototypes can be tested with CheckPrototypeDefinition and the size of non-user types can be tested with CheckTypeSize. For example: A variation on this used by some developers is to make the build directory a subdirectory of the source directory. Both check whether a particular symbol exists as either a pre-processor symbol (i.e. something that can be tested via an #ifdef statement), a function or a variable. For GLOB, both files and directories matching the expression are stored in outVar. This can make them less convenient than the other library types, since header search paths, compiler defines, etc. CMake will extract the full path to the executable which would be invoked by the specified command line, resolving the executable's location using the PATH environment variable if necessary and store the result in progVar. It has been updated for the CMake 3.22 release. CMake will convert this to a path to the built library when generating the linker command, including supplying any prefix or suffix to the file name as appropriate for the platform. 177 19.3. Source Control Commits . The behavior of CMake 3.1 and later uses the quoted value without substitution, which is what developers intuitively expect. A behavior unique to single configuration generators is that if CMAKE_BUILD_TYPE is not set, the build type is empty. The noValueKeywords define standalone keyword arguments which act like boolean switches. The TIMEOUT option can be used to handle commands which may run longer than expected or which might possibly never complete. set(myVar set(myVar set(myVar set(myVar a b c) a;b;c) "a b c") a;b;c) a "b c") # # # # # myVar myVar myVar myVar myVar = = = = = "a;b;c" "a;b;c" "a b c" "a;b;c" "a;b c" The value of a variable is obtained with ${myVar}, which can be used anywhere a string or variable is expected. include(module [OPTIONAL] [RESULT_VARIABLE myVar] [NO_POLICY_SCOPE]) When given a module name, the include() command will look in a well-defined set of locations for a file whose name is the name of the module (case-sensitive) with .cmake appended. Variable values can be used and will be evaluated before printing the result. 124 15.2. Setting The Language Standard By Feature Requirements . Loosely speaking, an item can be made to apply to just the Debug build type by preceding it with the keyword debug, or to all build types except Debug by preceding it with the keyword optimized. 89 include(CheckCSourceCompiles) check_c_source_compiles(" int main(int argc, char* argv[]) { int myVar; return 0; }" noWarnUnused FAIL_REGEX "[Ww]arn") if(noWarnUnused) message("Unused variables do not generate warnings by default") endif() In the case of Fortran, the file extension can affect how compilers treat source files, so the file extension can be explicitly specified with the SRC_EXT option to obtain the expected behavior. Code like the following tends to make projects harder to work with and can lead to developers feeling like they are fighting the project when they want to change flags for their own build through the CMake GUI application or similar: # Case 1: Only has an effect if the variable isn't already in the cache set(CMAKE_CXX_FLAGS "-Wall -Werror" CACHE STRING "C++ flags") # Case 2: Using FORCE to always update the cache variable, but this overwrites # any changes a developer might make to the cache set(CMAKE_CXX_FLAGS "-Wall -Werror" CACHE STRING "C++ flags" FORCE) # Case 3: FORCE + append = recipe for disaster (see discussion below) set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Werror" CACHE STRING "C++ flags" FORCE) The first case above highlights a common oversight made by developers new to CMake. The changes with 3.12 blur that distinction, but the remaining differences still leave room for unexpected results. There are essentially two approaches: in-source and out-of-source builds. Variable values can be obtained directly with the ${} syntax, which is both clearer and simpler than using get_property(). If the CMakeLists.txt file is processed with a CMake version older than the one specified, it will halt immediately with an error. In the above example, a clearer set of commands would be: subdir/CMakeLists.txt set(localVar bar) set(myVar ${localVar} PARENT_SCOPE) Obviously the above is a trivial example, but for real world projects, there may be many commands which contribute to building up the value of localVar before finally setting the parent's myVar variable. This allows libraries from multiple build types to be put in the same directory without overwriting each other. For single configuration generators, consider setting CMAKE_BUILD_TYPE to a better default value if it is empty. Some features may also have a macro of the form prefix_UPPERCASEFEATURE which provides the most appropriate implementation for that feature for each of the named compilers, including different versions of a compiler, where relevant. Section 28.5.1, "Target Sources" discusses some aspects of this topic. This is very similar to the situation for C/C++ headers and it is fairly common to see a similar form of include guard used: 56 if(DEFINED cool_stuff_include_guard) return() endif() set(cool_stuff_include_guard 1) # ... 54 7.3. Ending Processing Early . The property argument can be the name of any global property or one of the following pseudo properties: VARIABLES Return a list of all regular (i.e. non-cache) variables. The top section allows the project's source and build directories to be defined, the middle section is where the cache variables can be viewed and edited, while at the bottom are the Configure and Generate buttons and their associated log area. Once all cache variables are shown without red highlighting, the generate stage can be run. The function or macro's body is then executed at the point of the call. The include() command does a similar thing, pushing a new policy scope before starting to process the specified file and popping it again when processing of that file is completed. CMake brings some order to this, starting with a human-readable file called CMakeLists.txt that defines what should be built and how, what tests to run and what package(s) to create. The following incorrect logic is sometimes used: if(APPLE) # Some Xcode-specific settings here... 41 6.1.4. File System Tests . With neither keyword specified, the value(s) given replace any previous value. For the OUTPUT form, the APPEND keyword must be specified and the first OUTPUT file listed must be the same for the first and subsequent calls to add_custom_command(). target_link_libraries(targetName item1 [item2 ...] [item3 [item4 ...]] ... The build directory is where CMake will generate all build output (recommended directory layouts were discussed in Chapter 2, Setting Up A Project). For new projects, both must be set, but for existing projects, setting the build directory will also update the source directory, since the source location is stored in the build directory's cache. # Windows-specific example of imported library add_library(myWindowsLib SHARED IMPORTED) set_target_properties(myWindowsLib PROPERTIES IMPORTED_LOCATION /some/path/bin/foo.dll IMPORTED_IMPLIB /some/path/lib/foo.lib) 134 # Assume FOO_LIB holds the location of the library but its type is unknown add_library(mysteryLib UNKNOWN IMPORTED) set_target_properties(mysteryLib PROPERTIES IMPORTED_LOCATION ${FOO_LIB}) # Imported object library, Windows example shown add_library(myObjLib OBJECT IMPORTED) set_target_properties(myObjLib PROPERTIES IMPORTED_OBJECTS /some/path/obj1.obj # These .obj files would be .o /some/path/obj2.obj # on most other platforms) # Regular executable target using imported object library. Link flag As a special case, items starting with a hyphen other than -l or -framework will be treated as flags to be added to the linker command. elseif(expression2) # commands ... 83 $, $ In some situations, it may be useful to only add content if the compiler version is older or newer than some particular version. The situation becomes clear when pasting the contents of the macro body directly into the function where it is called (which is effectively what CMake will do with it): function(func) # Now it is clear, ARGN here will use the arguments from func foreach(arg IN LISTS ARGN) message("Argument: ${arg}") endforeach() endfunction() In such cases, consider making the macro a function instead, or if it must remain a macro then avoid treating arguments as variables. When a target is created, the initial value of this target property will be empty. The CMake GUI application will then present that variable as a combo box containing the valid values instead of as a text edit field. The destination cannot simply be a directory to copy the file into, it must be a file name, optionally with a path which can be absolute or relative. These more specialized modules check compiler flags, pre-processor symbols, functions, variables, header files and more. Sometimes a project may need to write a file whose contents depend on the build type. The full set of supported forms are: add_executable(targetName [WIN32] [MACOSX_BUNDLE] [EXCLUDE_FROM_ALL] source1 [source2 ...]) add_executable(targetName IMPORTED [GLOBAL]) add_executable(aliasName ALIAS targetName) The IMPORTED form can be used to create a CMake target for an existing executable rather than one built by the project. 388 28. The reasons for this and some of 132 the associated implications of reduced target visibility are covered in Section 16.3, "Promoting Imported Targets" further below. In later chapters, many of these properties are discussed and their relationship to other CMake commands, variables and features are explored. Other developers reading the call site usually won't need to look at the function implementation or its documentation to understand what each of the arguments mean. Note how the behavior to ignore the first newline immediately after the opening bracket makes the command more readable. Such files need to appear in a DEPENDS section for the build to be considered robust. These can be a little verbose, but they offer a great deal of flexibility and support very precise handling of feature-based functionality. Chapter 14, Compiler And Linker Essentials discusses a range of topics which highlight the strong preference for the target_...() commands. 231 22.7. Creating And Exporting Archives . add_compile_definitions() still affects all targets created in the same directory scope regardless of whether those targets are created before or after add_compile_definitions() is called, as this is a characteristic of the underlying COMPILE_DEFINITIONS directory property the command manipulates, not of the command itself. The configure_file() command allows a single file to be copied from one location to another, optionally performing CMake variable substitution along the way. Neither approach is more correct, it's more a matter of preference. The LENGTH_MAXIMUM and LENGTH_MINIMUM options can be used to exclude strings longer or shorter than a certain number of bytes respectively. A number of older CMake modules used to provide only variables to refer to imported entities. The UPLOAD form performs the complementary operation, uploading the named file to the specified url. The rest of the project, however, wants to move to the new behavior. A point that is often not well understood is that normal and cache variables are two separate things. In other words, what matters is the value the variable holds at the end of the CMakeLists.txt file for that directory. For example, the directory may be responsible for collecting a set of source file names and passing it back up to the parent as a list of files. One consequence of this is that if a project sets the _EXTENSIONS property, it should also set the _STANDARD property or else _EXTENSIONS may effectively be ignored. A number of variables of the form CMAKE_REQUIRED_... can be set before calling any of the compilation test macros to influence how they compile the code: CMAKE_REQUIRED_FLAGS Additional flags to pass to the compiler command line after the contents of the relevant CMAKE__FLAGS and CMAKE__FLAGS_ variables (see Section 14.3, "Compiler And Linker Variables"). This is especially important if a project makes use of things like git submodules or embeds third party source trees. The headers in myHeaderOnlyToolkit can then use this symbol to determine what things they declare and define rather than relying on the __cplusplus symbol provided by the C++ standard, the value of which is often unreliable for a range of compilers. The commands will be executed as part of building targetName, with the buildStage argument required to be one of the following: PRE_BUILD The commands should be run before any other rules for the specified target. 288 24.9. GoogleTest . Any targets created before the call are not affected. When the user attempts to modify trafficLight in cmake-gui, they will be given a combobox containing the three values Red, Orange and Green instead of a simple line edit widget which would otherwise have allowed them to enter any arbitrary text. These are all discussed in detail in Chapter 16, Target Types. If, however, more than one source file is listed, destination must refer to an existing directory. For such cases, source_generic.cpp may be the more appropriate source file. Before delving into the details, it is important to note that as CMake has evolved, the available methods for controlling the compiler and linker behavior have also improved. 206 21.2. Defining The Target System . The FORCE keyword is discussed further below. When the destination is an existing directory, the source file will be copied into it. There is no equivalent option for the C or C++ cases. 73 9.3. Directory Properties . These requirements are also preserved when targets are exported and installed (see Chapter 25, Installing). CMAKE_REQUIRED_DEFINITIONS A CMake list of compiler definitions, each one specified in the form -DFOO or -DFOO=bar. The macro takes only one argument, that being the name of a variable in which to store the boolean result (true means the system is big endian): include(TestBigEndian) test_big_endian(isBigEndian) message("Is target system big endian: ${isBigEndian}") 11.3. Checking Existence And Support One of the more comprehensive areas covered by CMake's modules is checking for the existence of or support for various things. A common situation where it can be useful to exclude a target from ALL is where the executable is a developer tool that is only needed occasionally. It only affects https:// urls. The result is val1 if expr evaluates to 1 and val0 if expr evaluates to 0. It is used to load the named module, a topic covered in depth in Chapter 11, Modules. From there, I became more deeply involved and now fulfill the role of volunteer co-maintainer. When writing out multiple lines to a file and when no substitution should be performed, bracket syntax is often the most convenient way to specify the content to be written. 26 5.4.1. Setting Cache Values On The Command Line CMake allows cache variables to be manipulated directly via command line options passed to cmake. 278 24.8. CDash Integration . The end result would look something like this: 107 cmake_minimum_required(3.11) project(Foo) if(CMAKE_CONFIGURATION_TYPES) if(NOT "Profile" IN_LIST CMAKE_CONFIGURATION_TYPES) list(APPEND CMAKE_CONFIGURATION_TYPES Profile) endif() else() set(allowableBuildTypes Debug Release Profile) set_property(CACHE CMAKE_BUILD_TYPE PROPERTY STRINGS "${allowableBuildTypes}") if(NOT CMAKE_BUILD_TYPE) set(CMAKE_BUILD_TYPE Debug CACHE STRING "" FORCE) elseif(NOT CMAKE_BUILD_TYPE IN_LIST allowableBuildTypes) message(FATAL_ERROR "Invalid build type: ${CMAKE_BUILD_TYPE}") endif() endif() # Set relevant Profile-specific flag variables if not already set... Every target subsequently created in the current directory scope and below will then inherit those options in their own COMPILE_OPTIONS target property. The primary way a project specifies details about its expected CMake version behavior is with the cmake_minimum_required() command. When a target is created, the initial value of this property is taken from the CMAKE__STANDARD variable. If the project still wanted to support being built with earlier CMake versions, then the additional checks would have to be explicitly enabled when run with later CMake versions. 26 5.4.1. Setting Cache Values On The Command Line .) The PRIVATE, PUBLIC and INTERFACE keywords have their usual meanings, controlling how the listed features should be applied. 183 20. If sourceDir is a path outside the source tree, CMake requires the binaryDir to be specified since a corresponding relative path can no longer be constructed automatically. The type can also be omitted, in which case the variable is given a special type that is similar to INTERNAL. The build type (also known as the build configuration or build scheme in some IDE tools) is a high level control which selects different sets of compiler and linker behavior. After the test has been performed once, subsequent CMake runs will use the cached result rather than performing the test again. If none of those match the value, the result of the expression is true. When the macro is called from inside another function, the macro ends up using the ARGN variable from that enclosing function, not the ARGN from the macro itself. One of the most useful commands for copying files at configure time is, unfortunately, one of the less intuitively named. They can even be used to provide different content based on whether a build or an install is being performed. MACROS Return a list of just the defined macros. 79 10.1. Simple Boolean Logic . The INCLUDE_DIRECTORIES and COMPILE_DEFINITIONS properties are really just conveniences, taking care of the compiler specific flags for the most common things projects often want to set. 4.6. Recommended Practices Target names need not be related to the project name. Consider the following simple example command: add_executable(myApp main.cpp) By default, the name of the executable would be myApp.exe on Windows and myApp on Unix-based platforms like macOS, Linux, etc. In particular, since the generated files won't exist until the end of the configure phase, they cannot be copied or read with configure_file(), file(COPY...), etc. variables to their original values # from the top of this example cmake_pop_check_state() 93 11.4. Other Modules CMake has excellent built-in support for some languages, especially C and C++. CMake disallows multiple file(GENERATE...) commands trying to generate the same output file. A number of methods have evolved in CMake for manipulating target properties. The CMake documentation provides the full list of policies, along with an explanation of the OLD and NEW behavior of each one. Library aliases cannot be installed and they cannot be defined as an alias of another alias. Rather than holding a standalone value like a variable does, a property provides information specific to the entity it is attached to. 390 28.2. Non-superbuild Structure . It may also add a linker flag like -stdlib=libc++ depending on the platform. For a complex project with many subdirectories, this can be quite tedious. Variables The preceding chapters showed how to define basic targets and produce build outputs. Rather than relying on developers being aware of this different behavior, projects should instead prefer to always specify the INPUT and OUTPUT files with an absolute path to avoid errors or files being generated in unexpected locations. PRIVATE items populate the COMPILE_DEFINITIONS target property, while INTERFACE items populate the INTERFACE_COMPILE_DEFINITIONS target property. Along the way, there was a classic pivotal moment which I now look back on with some degree of amusement. 298 25.1.1. Relative Layout . This search order allows projects to add their own modules seamlessly by adding directories to CMAKE_MODULE_PATH. In addition to the generic set_property() and get_property() commands, CMake also provides get_cmake_property() for querying global entities. For the toolchain options presented, each one requires progressively more information from the developer. Messages will normally be preceded by two hyphens. AUTHOR_WARNING Like WARNING, but only shown if developer warnings are enabled (requires the -Wdev option on the cmake command line). Comments are used extensively throughout this book and developers are encouraged to also get into the habit of commenting their projects just as they would for ordinary source code. There are some important differences in the way file(GENERATE...) behaves compared to most other CMake commands. For these situations, the file() command offers two more forms which help transform paths between platform native and CMake formats: file(TO_NATIVE_PATH input outVar) file(TO_CMAKE_PATH input outVar) The TO_NATIVE_PATH form converts input into a native path for the host platform. If different subdirectories of a large, complex project both want to make use of CMake code in some file in a common area of the project, they may both include() that file independently. This is undesirable, especially for multi configuration generators which don't even use the CMAKE_BUILD_TYPE variable's value. Compare the following two ways of setting policy behavior and note how the check and the enforcement use a consistent approach: # Version-level policy enforcement if(NOT CMAKE_VERSION VERSION_LESS 3.4) cmake_policy(VERSION 3.4) endif() # Individual policy-level enforcement if(POLICY CMP0055) cmake_policy(SET CMP0055 NEW) endif() If a project needs to manipulate multiple individual policies locally, surround that section with calls to cmake_policy(PUSH) and cmake_policy(POP) to ensure that the rest of the scope is isolated from the 100 changes. This distinction is critical, as it means the build type is not always known when CMake is processing a project's CMakeLists.txt file. Another supported but deprecated form is the following: target_link_libraries(targetName LINK_INTERFACE_LIBRARIES item [item...]) This is a pre-cursor to the INTERFACE keyword of the newer form covered above, but its use is discouraged by the CMake documentation. Another common example, this time used appropriately, is the conditional inclusion of a target based on whether or not a particular CMake option has been set. 47 6.2.3. Interrupting Loops . As a last resort, the CMAKE_POLICY_DEFAULT_CMPxxxx and CMAKE_POLICY_WARNING_CMPxxxx variables may allow a developer or project to work around some specific policy-related situations. This can also be used to check if a specific environment variable is defined: if(DEFINED SOMEVAR) if(DEFINED ENV{SOMEVAR}) # Checks for a CMake variable # Checks for an environment variable COMMAND Tests whether a CMake command, function or macro with the specified name exists. SEND_ERROR Indicates an error message which will be shown highlighted in red, where supported. Note that CMake does not enforce that the cache variable must be one of the values from the STRINGS property, it is only a convenience for the CMake GUI and ccmake tools. An example best illustrates how the three different keyword types are defined and handled: 63 function(func) # Define the supported set of keywords set(prefix ARG) set(noValues ENABLE_NET COOL_STUFF) set(singleValues TARGET) set(multiValues SOURCES IMAGES) # Process the arguments passed in include(CMakeParseArguments) cmake_parse_arguments(${prefix} "${noValues}" "${singleValues}" "${multiValues}" ${ARGN}) # Log details for each supported keyword message("Option summary:") foreach(arg IN LISTS noValues) if(${${prefix}_${arg}}) message(" ${arg} enabled") else() message(" ${arg} disabled") endif() endforeach() foreach(arg IN LISTS singleValues multiValues) # Single argument values will print as a simple string # Multiple argument values will print as a list message(" ${arg} = ${${prefix}_${arg}}") endforeach() endfunction() # Examples of calling with different combinations # of keyword arguments func(SOURCES foo.cpp bar.cpp TARGET myApp ENABLE_NET) func(COOL_STUFF TARGET dummy IMAGES here.png there.png gone.png) The corresponding output would look like this: Option summary: ENABLE_NET enabled COOL_STUFF disabled TARGET = myApp SOURCES = foo.cpp;bar.cpp IMAGES = Option summary: ENABLE_NET disabled COOL_STUFF enabled TARGET = dummy SOURCES = IMAGES = here.png;there.png;gone.png Compared to basic argument handling using named arguments and/or the ARG... variables, the 64 advantages of cmake_parse_arguments() are numerous. Indeed, it is not unusual for an organization to build up its own collection of modules, perhaps even storing them in their own separate repository. Dependent projects can always require a more recent version if they so wish, but they cannot require an older one. Cache variables can also have a property marking them as advanced or not. The binary directory is where everything produced by the build is created. 28 5.5. Debugging Variables And Diagnostics . The source directory must be set to the directory containing the CMakeLists.txt file at the top of the project's source tree. It should be noted, however, that logging with a STATUS mode is not the same as logging a message with no mode keyword at all. This is discussed in detail in Section 23.5, "Finding Packages", but for the moment, a simplified form of that command without any of the optional keywords demonstrates its basic usage: find_package(PackageName) When used in this way, the behavior is very similar to include() except CMake will search for a file called FindPackageName.cmake rather than PackageName.cmake. For all the other types of entities, any number of items of that type can be listed. Chapters cover the toolchain and build configuration, different types of targets, carrying out custom tasks and handling platformspecific features. The COMPILE_FLAGS property is treated as a single string that is included directly on the compiler command line. CACHE_VARIABLES Return a list of all cache variables. The RELATIVE option can be used to change this behavior such that the reported paths are relative to a specific directory. The test is case-insensitive. They can be used to collect together transitive properties and dependencies without actually producing their own binaries, or they can even be a kind of library that is simply a collection of object files rather than a traditional static or shared library. The reasoning and details of the pre-3.3 behavior are non-trivial and are largely historical, but for the interested reader, the full set of information is available in the CMake documentation under the CMP0060 policy. They often lead to a number of very common mistakes, with some of the more prevalent ones presented below. Because copying a file is generally a familiar task to most users, it is natural for new CMake developers to implement file copying in terms of the same methods they already know. are executed. Because a macro does not introduce a new scope, the behavior of the return() statement is dependent on where the macro is called. The entitySpecific part is similar to that for set_property() and must be one of the following: GLOBAL DIRECTORY TARGET SOURCE INSTALL TEST CACHE VARIABLE [dir] target source file test var As before, GLOBAL refers to the build as a whole and therefore requires no specific entity to be named. For this reason, it is likely to be clearer to avoid using this second RANGE form and explicitly specify both the start and stop values instead. The CONFIGURE_DEPENDS option added in CMake 3.12 tries to address this deficiency, but it comes with performance penalties and only works for some project generators. It is common to see tutorials and examples use a variable for the project name and reuse that variable for the name of an executable target like so: # Poor practice, but very common set(projectName MyExample) project(${projectName}) add_executable(${projectName} ...) This only works for the most basic of projects and encourages a number of bad habits. This must be a single string with multiple flags being separated by spaces, unlike all the other variables below which are CMake lists. 14 Chapter 4. The use of CMake scripts instead of a platform specific shell or a separately installed script engine can reduce the complexity of the project and reduce the additional dependencies it requires in order to build. 42 CMake also supports testing a string against a regular expression: if(value MATCHES regex) The value again follows the variable-or-string rules defined above and is compared against the regex regular expression. It may help to think of VARIABLE as specifying the current scope, then the property of interest is the variable named by propName. These are typically just wrappers around the macros provided by one of the CheckSourceCompiles modules, so the same set of variables used for customizing how the test code is built still apply. No command by that name previously existed, so no further action is required. 103 13.1.1. Single Configuration Generators . Consider a project processed for a multi configuration CMake generator like Xcode or Visual Studio. Note that not all functionality provided by a particular language version can be explicitly specified using compile features. The above example shows a nested condition, but any part of a generator expression can be nested. It will always hold an absolute path. This can lead to CMP0048 policy warnings which highlight the conflict. One way is to use a quoted value spread across multiple lines: file(WRITE multi.txt "First line Second line ") If using CMake 3.0 or later, the lua-inspired bracket syntax introduced back in Section 5.1, "Variable Basics" can sometimes be more convenient, since it prevents any variable substitution of the content. This property is intended for general linker flags, not those flags which specify other libraries to link to. add_executable(myExe $) Another form of the add_library() command allows interface libraries to be defined. Modules......Page 9511.1. Useful Development Aids......Page 9611.3. Checking Existence And Support......Page 9811.5. Recommended Practices......Page 10312.1. Policy Control......Page 10512.2. Policy Scope......Page 10812.3. Recommended Practices......Page 109Part II: Builds In Depth......Page 11113.1. Build Type Basics......Page 11213.1.2. Multiple Configuration Generators......Page 11313.2. Common Errors......Page 11413.3. Custom Build Types......Page 11513.4. Recommended Practices......Page 11914.1.1. Compiler Flags......Page 12014.1.2. Linker Flags......Page 12114.1.3. Target Property Commands......Page 12214.2. Directory Properties And Commands......Page 12514.3. Compiler And Linker Variables......Page 12714.4. Recommended Practices......Page 13115.1. Setting The Language Standard Directly......Page 13315.2. Setting The Language Standard By Feature Requirements......Page 13515.2.1. Detection And Use Of Optional Language Features......Page 13715.3. Recommended Practices......Page 13916.1. Executables......Page 14116.2. Libraries......Page 14216.4. Recommended Practices......Page 14817.1. Custom Targets......Page 15117.2. Adding Build Steps To An Existing Target......Page 15317.3. Commands That Generate Files......Page 15517.4. Configure Time Tasks......Page 15817.5. Platform Independent Commands......Page 16017.6. Combining The Different Approaches......Page 16217.7. Recommended Practices......Page 16418.1. Manipulating Paths......Page 16518.2. Copying Files......Page 16818.3. Reading And Writing Files Directly......Page 17418.4. File System Manipulation......Page 17818.5. Downloading And Uploading......Page 18018.6. Recommended Practices......Page 18219.1. Project Version......Page 18419.2. Source Code Access To Version Details......Page 18619.3. Source Control Commits......Page 18919.4. Recommended Practices......Page 19220.1. Build Basics......Page 19420.3. Shared Library Versioning......Page 19520.4. Interface Compatibility......Page 19720.5. Symbol Visibility......Page 20220.5.1. Specifying Default Visibility......Page 20320.5.2. Specifying Individual Symbol Visibilities......Page 20420.6. Mixing Static And Shared Libraries......Page 20920.7. Recommended Practices......Page 212Chapter 21. The CMake users mailing list is also a great source of advice and a recommended forum for asking CMake-related questions where the documentation doesn't provide sufficient guidance. Therefore, prefer to start with a dependency as PRIVATE and only make it PUBLIC when it is clear that the dependency is needed by those linking to the target. For DIRECTORY, if no dir is named, the current source directory is used. 16 4.3. Linking Targets . Choosing the most appropriate command for copying files is not always clear. When looking for a module's file, CMake first consults the variable CMAKE_MODULE_PATH. One notable exception to this is projects intended for a wide range of compilers where language standard support can vary. This can be done as long as it comes after the project() command, since that command populates the default compiler and linker flag variables. CMake provides two lists of known features: CMAKE__KNOWN_FEATURES which contains all known features for the language and CMAKE__COMPILE_FEATURES which contains only those features supported by the compiler. An alternative is to use CMake itself as a scripting engine, providing a platform independent language in which to express arbitrary logic. 139 16.4. Recommended Practices . 406 28.6. Miscellaneous Project Features . set(longStr abcdefabcdef) set(shortBit def) string(FIND ${longStr} ${shortBit} fwdIndex) string(FIND ${longStr} ${shortBit} revIndex REVERSE) message("fwdIndex = ${fwdIndex}, revIndex = ${revIndex}") This results in the following output: fwdIndex = 3, revIndex = 9 Replacing a simple substring follows a similar pattern: string(REPLACE matchString replaceWith outVar input [input...]) 33 The REPLACE operation will replace every occurrence of matchString in the input strings with replaceWith and store the result in outVar. If the same target name is used in more than one add_executable() command, CMake will fail and highlight the error. As covered in later chapters, it also allows developer options to be saved between runs. This essentially just discards the directory part of input. In some situations, however, it may be desirable for the output file to always be the same, such as where the file contents do not depend on the build type but rather on some other generator expressions. For example, consider a project which has two C++ source files, one for building with Visual Studio compilers or those compatible with them (e.g. Intel) and another for building with all other compilers. That said, some of these expressions do have valid uses. In various contexts, variables may be interpreted as a different type, but ultimately, they are just strings. Note that in practice, the use of VARIABLE scope with get_property() is relatively uncommon. The first is to set the language standard directly and the second is to allow projects to specify the language features they need and let CMake select the appropriate language standard. CMake has a large number of pre-defined properties of each type. In the days where C++11 support was relatively new, CMake had no direct support for choosing which standard to use, so projects were left to work out the required flags on their own. Aim to avoid modifying policy settings inside a function. Selecting the Grouped option can make viewing advanced variables easier by grouping variables together based on the start of the variable name up to the first underscore. In addition to the above, each individual argument can be referenced with a name of the form ARG# where # is the number of the argument (e.g. ARG1, ARG2, etc.). These are simpler and often have good support in IDE environments not so closely associated with a particular compiler (Qt Creator, KDevelop, etc.). 195 20.6. Mixing Static And Shared Libraries . A common naming convention is for the file name part of the source and destination to be the same, except the source has an extra .in appended to it. Any source files or directories specified without an absolute path are treated as being relative to the current source directory. Think of it as the CMake project file, defining everything about the build from sources and targets through to testing, packaging and other custom tasks. Report DMCA The handbook for every CMake user, providing structured learning, the latest best practices and real-world advice from one of the CMake co-maintainers. For example, CMake also provides other expressions which give details about the directory and file name of a target's built binary. A toolchain file can provide the same information but has the advantage that it can be re-used as needed. 200 20.7. Recommended Practices . Flow Control A common need for most CMake projects is to apply some steps only in certain circumstances. Projects should generally try to avoid these two options due to the severely limited number of project generators that support them. CMake even provides some convenience macros through the WriteCompilerDetectionHeader module which make this task easier. Aliases can only point to real targets (i.e. an alias of an alias is not supported) and they cannot be installed or exported (both covered in Chapter 25, Installing). In particular, some CMake generators may rebuild more than should be necessary when compile options for only a few source files change. 284 24.8.4. Test Measurements And Results . add_executable(main main.cpp) add_library(bar STATIC ...) add_library(foo::bar ALIAS bar) # Typo in name being linked to, CMake will assume a # library called "bart" will be provided by the # system at link time and won't issue an error. As soon as the CMake run is finished, the change to the environment variable is lost. The standard output and standard error streams can be captured and stored in variables instead of being sent to the default pipes. One of the main ways developers get caught out by this behavior is to treat the compiler and linker variables as though they apply immediately to any targets that are created. If a project uses FORCE to unilaterally set cache variables in this manner, any change made by the developer to that cache variable will be lost. The other mechanism CMake provides for helping debug usage of variables is the variable_watch() command. In particular, the family of target_...() commands are a critical part of CMake and all but the most trivial of CMake projects would typically use them. This is a prime example of the functionality provided by generator expressions. It extracts just the extension part of the file name from the first "." onward. 96 12.2. Policy Scope. These meta-features take the form _std_ and when listed as a required compile feature, CMake will ensure compiler flags are used which enable that language standard. Any changes to variables in the child are performed on the child's copy, leaving the caller's variables unchanged. 41 6.1.3. Comparison Tests . • All but the special global pseudo properties can be fully manipulated using the generic set_property() command, making it predictable for developers and offering flexible APPEND functionality where needed. In the following example, an executable built by the project is used to generate a source file which is then compiled as part of another executable. Where possible, the use of platform independent commands should be preferred. target_compile_definitions(targetName item1 [item2 ...] [item3 [item4 ...]] ... A fundamental part of CMake is the concept of a project having both a source directory and a binary directory. This command enables projects to perform find and replace operations, regular expression matching, upper/lower case transformations, strip whitespace and other common tasks. The output from the above example would be: Processing a-1 Processing a-2 Processing a-3 Accumulated list: a1;a2;a3 Accumulated list: b1;b2;b3 Processing c-1 Processing c-2 Processing c-3 Accumulated list: c1;c2;c3 48 6.3. Recommended Practices Minimize opportunities for strings to be unintentionally interpreted as variables in if(), foreach() and while() commands. 71 If none of the optional keywords are given, the value of the named property is retrieved. Similarly, if the destination directory is not absolute, it will be interpreted as being relative to the current binary directory. These allow the developer to choose between different build configurations without having to re-run CMake, which is more suitable for generators creating projects for use in IDE environments like Xcode and Visual Studio. The above behavior of scoping for variables highlights one of the important characteristics of add_subdirectory(). To address this, CMake 3.9 added a new GENERATOR_IS_MULTI_CONFIG global property which is set to true when a multi configuration generator is being used, providing a definitive way to obtain that information instead of relying on inferring it from CMAKE_CONFIGURATION_TYPES. When a target is created, the initial value of this target property is taken from the directory property of the same name. When calling set_property() with APPEND or APPEND_STRING options, only the immediate value of the property is considered (i.e. no inheriting occurs when working out the value to append to). The set() command acts more like set-if-notset when used to define cache variables, as does the option() command (which has no FORCE capability). TLS_CAINFO fileName A custom Certificate Authority file can be specified with this option. Some, like Makefiles and Ninja, support only a single build type per build directory. else() # Things for other platforms here... This will be a subset of what the COMMANDS pseudo property would return, but note that the upper/lower case of the names can be different to what the COMMANDS pseudo property reports. 4 2. CMake finds an existing command by that name, so it defines the name _printme to point to the old command and sets printme to point to the new definition. Even experienced users should find at least the Recommended Practices section at the end of each chapter to be a us E-Book Content A PRACTICAL GUIDE CRAIG SCOTT Professional CMake: A Practical Guide Version: 1.0.3 © 2018 by Craig Scott This book or any portion thereof may not be reproduced in any manner or form without the express written permission of the author, with the following specific exceptions: • The original purchaser may make personal copies exclusively for their own use on their electronic devices, provided that all reasonable steps are taken to ensure that only the original purchaser has access to such copies. Back in Section 4.3, "Linking Targets", the target_link_libraries() command was presented, along with an explanation of how inter-target dependencies are expressed using PRIVATE, PUBLIC and INTERFACE specifications. Set the project name directly rather than via a variable, choose a target name according to what the target does rather than the project it is part of and assume the project will eventually need to define more than one target. The general form of the command is: target_link_libraries(targetName item1 [item2 ...] [item3 [item4 ...]] ... Rather than the somewhat verbose set() command, developers can use option() instead: option(optVar helpString [initialValue]) If initialValue is omitted, the default value OFF will be used. 18 4.4. Linking Non-targets In the preceding section, all the items being linked to were existing CMake targets, but the target_link_libraries() command is more flexible than that. Custom Tasks No build tool can ever hope to implement every feature that will ever be needed by any given project. 37 6. When listing build targets as the command or arguments in a call to add_custom_command() or add_custom_target(), prefer to explicitly list them as DEPENDS items rather than relying on CMake's automatic target dependency handling. The variable can still be given other values via set() commands in the CMakeLists.txt file or by other means such as manually editing the CMakeCache.txt file. There are usually better comparison expressions that handle strings more robustly, which are covered in Section 6.1.3, "Comparison Tests" further below. While an empty build type is technically valid, it is also often misunderstood by developers to mean a Debug build rather than its own distinct build type. Both of the above can be a surprise to developers, but at least the CMake 3.1 behavior is always predictable. Typically, argsToParse is given as ${ARGN}, which is the set of unnamed arguments passed to the enclosing function or macro. Note, however, that keywords should not be too long or they can be cumbersome to use. 9.7. Other Property Types CMake also supports properties on individual tests and it provides the usual test-specific versions of the property setter and getter commands: set_tests_properties(test1 [test2...] PROPERTIES propertyName1 value1 [propertyName2 value2] ... 76 9.7. Other Property Types . The VARIABLE type is a bit different, with the variable name being specified as the propName rather than being attached to the VARIABLE keyword. Try to establish a variable naming convention early. I'd also like to thank my past colleagues, Matt Bolger and Lachlan Hetherton, who both provided constructive feedback and reminded me of the importance of a fresh set of eyes. The source directory is frequently under version control with a tool like git, subversion, or similar. BRIEF_DOCS Retrieves the brief documentation string for the named property. If PROGRAM_ARGS is given, the set of command line arguments are also stored as a list in the variable named by argVar. They are not intended for permanent use, but are more aimed at helping developers quickly and easily log information temporarily to help investigate problems in the project. The simple example given previously required defining a target name for the executable and listing the source files to be compiled: add_executable(myApp main.cpp) This assumes the developer wants a basic console executable to be built, but CMake also allows the developer to define other types of executables, such as app bundles on Apple platforms and Windows GUI applications. The commands are passed to the operating system for execution as is with no 149 intermediate shell environment. 244 23.1.2. Cross-compilation Controls . It is also impossible to distinguish between different overloads of the same function, so these cannot be checked either. This undocumented behavior may even be removed in a future version of CMake, so it should not be used. Projects may use cache variable properties to provide a pre-defined set of values for GUI tools to present as a combobox or similar instead (see Section 9.6, "Cache Variable Properties"). # Require C++11 and disable extensions for all targets set(CMAKE_CXX_STANDARD 11) set(CMAKE_CXX_STANDARD_REQUIRED ON) set(CMAKE_CXX_EXTENSIONS OFF) When using GCC or Clang, the above would typically add the -std=c++11 flag. set(foo "My variable") unset(bar) include(CMakePrintHelpers) cmake_print_variables(foo bar CMAKE_VERSION) The output for the above would be something similar to the following: 88 foo="My variable" ; bar="" ; CMAKE_VERSION="3.8.2" 11.2. Endianness When working with embedded platforms or projects intended for a wide variety of architectures, it can be desirable for the project to be aware of the endianness of the target system. CMake 3.12 introduces features that make object libraries behave more like other types of libraries, but with some caveats. The cache is a primary means for developers to change variables locally without having to edit project files. 186 20.4. Interface Compatibility . In general, however, where there is a choice, static libraries will typically be the more convenient choice in CMake projects. option(BUILD_MYLIB "Enable building the myLib target") if(BUILD_MYLIB) add_library(myLib src1.cpp src2.cpp) endif() More complex projects often use the above pattern to conditionally include subdirectories or perform a variety of other tasks based on a CMake option or cache variable. The weaker automatic dependencies may not enforce all the relationships that developers may intuitively expect. When the configure stage is run for the first time on a new project, the developer is presented with a dialog similar to that shown in the next screenshot: This dialog is where the CMake generator and toolchain are specified. Consider whether the benefits outweigh the costs in terms of time to implement and maintain the checks and the complexity of the project. Sometimes, however, more complex logic is required and such custom tasks are often implemented using platform specific shell scripts. For improved readability, arguments can be split across multiple lines. These situations should generally be relatively rare. The two main exceptions to this are at the start of the project's top level CMakeLists.txt file and at the top of a module file that could be re-used across multiple projects. In particular, do not rely on the undocumented behavior whereby the old implementation of a command is made available by such a name when a function or macro redefines an existing command. Building Simple Targets As shown in the previous chapter, it is relatively straightforward to define a simple executable in CMake. Very early versions of CMake required the name to be repeated as an argument to endfunction() or endmacro(), but new projects should avoid this as it only adds unnecessary clutter. Introduction . get_cmake_property(resultVar property) Just like for get_property(), resultVar is the name of a variable in which the value of the requested property will be stored when the command returns. Since spaces are not permitted, the project name does not have to be surrounded by quotes. Nonetheless, a good habit to establish is to define the project's version here so that other parts of the project can refer to it. Rather than requiring a project to manually save the existing value of any policies it wants to change temporarily, CMake provides a policy stack which can be used to simplify this process: cmake_policy(PUSH) cmake_policy(POP) The existing state of all policies can be saved with a PUSH operation and the current state discarded with a corresponding POP. 240 23.1.1. Apple-specific Behavior . The simplest way to invoke it is to change to the build directory and pass options to cmake for the generator type and location of the source tree. A partial exception to this is in toolchain files where initial defaults may be defined (see Chapter 21, Toolchains And Cross Compiling for a detailed discussion of this area). In addition to the generic and target-specific property getters and setters, CMake also has a number of other commands which modify target properties. At some point, developers will need to carry out a task that falls outside the directly supported functionality. endwhile() 47 The condition is tested and if it evaluates to true (following the same rules as the expression in if() statements), then the loop body is executed. In general, prefer to use functions rather than macros, since the use of a new variable scope within the function better isolates that function's effects on the calling scope. Instead of copying individual files, command mode can also copy entire directories: cmake -E copy_directory dir1 [dir2...] destination Unlike the file-related copy commands, the destination directory is created if required, including any intermediate path. As an example, before version 3.0, CMake allowed a project to call get_target_property() with the name of a target that didn't exist. Developers should be aware that the file(GLOB...) commands are not as fast as, say, the Unix find shell command. For profiling, the RelWithDebInfo default build type is a good one to choose as the base configuration since it enables both debugging and most optimizations: 108 set(CMAKE_C_FLAGS_PROFILE "${CMAKE_C_FLAGS_RELWITHDEBINFO} -p" CACHE STRING "") set(CMAKE_CXX_FLAGS_PROFILE "${CMAKE_CXX_FLAGS_RELWITHDEBINFO} -p" CACHE STRING "") set(CMAKE_EXE_LINKER_FLAGS_PROFILE "${CMAKE_EXE_LINKER_FLAGS_RELWITHDEBINFO} -p" CACHE STRING "") set(CMAKE_SHARED_LINKER_FLAGS_PROFILE "${CMAKE_SHARED_LINKER_FLAGS_RELWITHDEBINFO} -p" CACHE STRING "") set(CMAKE_STATIC_LINKER_FLAGS_PROFILE "${CMAKE_STATIC_LINKER_FLAGS_RELWITHDEBINFO} -p" CACHE STRING "") set(CMAKE_MODULE_LINKER_FLAGS_PROFILE "${CMAKE_MODULE_LINKER_FLAGS_RELWITHDEBINFO} -p" CACHE STRING "") Each custom configuration should have the associated compiler and linker flag variables defined. 302 25.2.1. Interface Properties . 123 Chapter 15. ③ myVar is still visible in the child scope. For all of the target_...() commands except target_link_libraries(), the specified target can be defined anywhere in the project, with the only requirement being that the target has been created at some point before the target_...() command is called. In general though, projects should prefer to use the variables or properties to set the language requirements at a higher level for better maintainability and robustness. 159 18.3. Reading And Writing Files Directly . Unfortunately, this often results in the use of platform-specific shell commands with add_custom_target() and add_custom_command(), sometimes also with dependency problems that require developers to run CMake multiple times and/or manually build targets in a particular sequence. For functions, that label acts like a variable, whereas for macros it acts like a string substitution. Imported libraries also support a number of other target properties, most of which can typically be left alone or are automatically set by CMake. The type must be one of the following: BOOL The cache variable is a boolean on/off value. Just like for the compiler flags, the CMAKE__LINKER_FLAGS are used when linking any build configuration, whereas the CMAKE__LINKER_FLAGS_ flags are only added for the corresponding CONFIG. 363 26.5. Recommended Practices . A typical scenario where projects may be tempted to go down this path is to pass version details to just one or two sources as compiler definitions, but as discussed in Section 19.2, "Source Code Access To Version Details", there are better alternatives to source properties which do not suffer from the same side effects. 330 25.8. Recommended Practices . End users cannot manipulate properties directly, so the project is in full control of how the properties are set. 149 17.5. Platform Independent Commands . Modules The preceding chapters have focused mostly on the core aspects of CMake. In CMake, a variable has a particular scope, much like how variables in other languages have scope limited to a particular function, file, etc. A variable is not attached to any particular entity and it is very common for projects to define and use their own variables. As a result, it may require manual escaping, whereas COMPILE_OPTIONS is a list and CMake performs any required escaping or quoting automatically. The techniques presented are aimed at real world use, with the intention of establishing good habits and teaching sound methods which scale to very large projects and can handle more complex scenarios. • Because directories can be self-contained, it becomes relatively trivial to turn parts of the build on

or off simply by choosing whether or not to add in that directory. cmake -D myVar:type=someValue ... That said, since many online tutorials and examples still use them, developers should at least be aware of the directory level properties and commands. This means all targets created in the current directory and below will have the directories added to their INCLUDE_DIRECTORIES target property. The SYSTEM keyword has the same effect as for the target_include_directories() command. Any line beginning with a # character is treated as a comment. 85 Chapter 11. Used judiciously, generator expressions can result in more succinct CMakeLists.txt files. RelWithDebInfo This is somewhat of a compromise of the previous two. If BASE_DIR is given, relative paths are interpreted as being relative to dir instead of the current source directory (i.e. CMAKE_CURRENT_SOURCE_DIR). 400 28.5.1. Target Sources . Both COPY and INSTALL also support applying certain logic to files that match or do not match a particular wildcard pattern or regular expression. Modules are collected together and provided in a single directory as part of a CMake release. The DOWNLOAD form also supports a few more options: EXPECTED_HASH ALGO=value Specifies the checksum of the file being downloaded so that CMake can verify the contents. If the commands produce no output on the console, it can sometimes be useful to specify a short message with the COMMENT option. The following reduced form shows the more commonly useful options: file(STRINGS fileName outVar [LENGTH_MAXIMUM maxBytesPerLine] [LENGTH_MINIMUM minBytesPerLine] [LIMIT_INPUT maxReadBytes] [LIMIT_OUTPUT maxStoredBytes] [LIMIT_COUNT maxStoredLines] [REGEX regex]) Options not shown above relate to encoding, conversion of special file types or treatment of newline characters and would not be needed in most situations. Do not rely on this behavior. Generator Expressions . set(msg "Hello world") file(WRITE hello.txt ${msg}) file(APPEND hello.txt " from CMake") The above example would result in the file hello.txt containing a single line of text Hello world from CMake. Thus, when a project starts with cmake_minimum_required(VERSION 3.2), it says that at least CMake 3.2 is needed and also that the project expects CMake to behave like the 3.2 release. endfunction() macro(name [arg1 [arg2 [...]]]) # Macro body (i.e. commands) ... 322 25.7.1. Config Files For CMake Projects . A naive approach would be to assume the CMAKE_BUILD_TYPE variable could be used as a substitution, but this does not work for multi configuration generators like Xcode or Visual Studio. Later chapters of this book also highlight a number of useful variables and the way they affect or give information about the build. If the library type is not known, the UNKNOWN type should be given, in which case CMake will simply use the full path to the library without further interpretation in places like linker command lines. CMake will use the appropriate linker flags to link each item listed in the LINK_LIBRARIES property. In the CMake reference documentation, if a particular command or property supports generator expressions, the documentation will mention it. Setting a boolean cache variable is such a common need that CMake provides a separate command expressly for that purpose. The DEPENDS keyword is especially useful when the file being listed is generated by some other custom command (see Section 17.3, “Commands That Generate Files” further below), where CMake will set up the necessary dependencies to ensure the other custom commands execute before this custom target’s commands. component determines how symbolic links should be handled when computing the path to be stored in outVar: ABSOLUTE Compute the absolute path of input without resolving symbolic links. 39 • If neither of the above two cases apply, it will be treated as a variable name (or possibly as a string) and evaluated further as described below. Both must be provided when setting a cache variable, although the docstring can be empty. Furthermore, since functions introduce their own variable scope, it may seem that there is no easy way to pass information back to the caller, but this is not the case. For this reason, they are often referred to as usage requirements, in contrast to the non-INTERFACE properties which are sometimes called build requirements. Typically, it is not desirable to store the result in the cache, so the CACHE keyword is not often required. When naming targets for libraries, resist the temptation to start or end the name with lib. Note that it is not unusual to see projects include CMAKE_CURRENT_SOURCE_DIR or CMAKE_CURRENT_BINARY_DIR as part of the path with the source and destination respectively, but this just adds unnecessary clutter and should be avoided. set_property(CACHE CMAKE_BUILD_TYPE PROPERTY STRINGS Debug Release Profile) Properties can only be changed from within the project’s CMakeLists.txt files, so they can safely set the STRINGS property without having to worry about preserving any developer changes. The importance of what a version number communicates to users is often underestimated, resulting in users with unmet expectations or confusion about changes between releases. This is the method by which details about an external package are often brought into the build, including things like imported targets, variables defining locations of relevant files, libraries or programs, information about optional components, version details and so on. This was unreliable and inconsistent, so in CMake 3.10 the behavior was changed to make INPUT act as relative to the current source directory and OUTPUT relative to the current binary directory, just like most other CMake commands that handle paths. This keeps the sources and the build outputs completely separate from each other, thus avoiding the intermixing problems experienced with in-source builds. and developers will likely want to bookmark it for quick reference. A few minutes consulting the CMake reference documentation can save many hours of unnecessary work constructing complex generator expressions which were not really needed. The interested reader should consult the CMake module documentation for full details. It has mostly the same effect as DEPENDS for the listed file, but some generators may apply additional logic such as where to place the custom command in an IDE project. 132 16.1. Executables . In practice, this means it will be created with a WinMain() entry point instead of just main() and it will be linked with the /SUBSYSTEM:WINDOWS option. 111 14.1. Target Properties . 78 Chapter 10. For historical reasons, the items passed to the target_link_libraries() command can be prefixed with the debug or optimized keywords to indicate that the named item should only be linked in for debug or non-debug builds respectively. CMake, the chosen build tool (e.g. make, Visual Studio, etc.), CTest and CPack will all create various files within the build directory and subdirectories below it. Do not define or call a function or macro with a name that starts with a single underscore. It relies on those who were kind enough to (sometimes unknowingly!) get involved in these activities as much as it does on the author and I cannot thank these people enough for their kindness and wisdom. The property specific setters may be more convenient in some situations, such as allowing multiple properties to be set at once, but their lack of APPEND functionality may steer some projects towards just using set_property(). Information can be obtained about the compiler being used, the platform for which the target is being built, the name of the build configuration and more. The use of these keywords is strongly discouraged, since generator expressions provide much better control over when an item should be added. It’s value is different for different build configurations when using a multi configuration generator like Xcode or Visual Studio. Of course, add_custom_command() can also be used to generate files consumed by existing targets, such as generating source files. --config Release With the above example, a developer could simply change the generator name given to the -G parameter and the rest of the script would work unchanged. 86 CMakeLists.txt: cmake_minimum_required(VERSION 3.0) project(Example) list(APPEND CMAKE_MODULE_PATH "${CMAKE_SOURCE_DIR}/cmake") # Inject code from project-provided modules include(CoolThings) include(MyModule) There is one exception to the search order used by CMake to find a module. This can be useful, for example, to have some commands run from one working directory and other commands run from somewhere else. Out-ofsource builds also have the advantage that the developer can create multiple build directories for the same source directory, which allows builds to be set up with different sets of options, such as debug and release versions, etc. The 11th Edition of the book Professional CMake: A Practical Guide is now available. Avoid unary expressions with quotes, prefer to use a string comparison operation instead. REMOVE_DUPLICATES will ensure the list contains only unique items. The value of the variable is irrelevant, only its existence is tested. When calling execute_process(), most of the time the success of the command should be tested by capturing the result using RESULT_VARIABLE and testing it with the if() command. If more control is required, developers can instead override the native compilers, with the paths to the compilers being given in a follow-up dialog. The following shows a few comment examples and brings together the concepts introduced in this chapter: 13 cmake_minimum_required(VERSION 3.2) # We don't use the C++ compiler, so don't let project() # test for it in case the platform doesn't have one project(MyApp VERSION 4.7.2 LANGUAGES C) # Primary tool for this project add_executable(mainTool main.c debug.c # Optimized away for release builds) # Helpful diagnostic tool for development and testing add_executable(testTool testTool.c) 3.5. Recommended Practices Ensure every CMake project has a cmake_minimum_required() command as the first line of its top level CMakeLists.txt file. This allows CMake to fix bugs internally and introduce new features, but still maintain the expected behavior of any particular past release. If another target needs to be used as an argument to one of the commands, CMake will not automatically perform the same substitution, but it is trivial to obtain an equivalent substitution with the TARGET_FILE generator expression. Not only does it cover the process from start to end, it also supports a wide range of platforms, tools and languages. For single configuration generators, there is only one build type and it is specified by the CMAKE_BUILD_TYPE cache variable, which is a string. Even so, checking CMAKE_CONFIGURATION_TYPES is still such a prevalent pattern that projects should continue to only modify it if it exists and never create it themselves. The example also shows how to invoke the cmake command correctly. Version 2.8.12 has a reduced feature coverage, lacking a number of useful features but it may be workable for older projects. If the setting of a source property occurs in a different directory scope, the target will not see that property change and therefore the compilation, etc. The advice and strategies contained within this work may not be suitable for every situation. This can be changed with the WORKING_DIRECTORY option, which can be an absolute path or a relative path, the latter being relative to the current binary directory. If the project’s CMakeLists.txt file then tries to set the same cache variable and specifies a type of FILEPATH or PATH, then if the value of that cache variable is a relative path, CMake will treat it as being relative to the directory from which cmake was invoked and automatically convert it to an absolute path. Note that this is a one-way transition, it is not possible to demote a global target back to local visibility. POST_BUILD tasks are relatively common, but PRE_LINK and PRE_BUILD are rarely needed since they can usually be avoided by using the OUTPUT form of add_custom_command() instead (see next section). Other characters are also possible via indirect means, but again, these are not typically seen in normal use. Some modules began their life quite early on in CMake’s existence and these can sometimes become less useful if not kept up to date with changes to CMake or to the areas those modules relate to. This is a stronger test than DEFINED in that it tests whether the named property has actually been given a value (the value itself is irrelevant). They instead have to be added to targets in the same way that sources are added. In some special situations, projects may want to indicate that no languages are used, which can be done using LANGUAGES NONE. Language Requirements With the ongoing evolution of the C and C++ languages, developers are increasingly required to understand the compiler and linker flags that enable support for the C and/or C++ version their code uses. 91 Two other notable modules are CheckSymbolExists and CheckCXXSymbolExists. For example: # Regular library targets add_library(algo_fast ...) add_library(algo_accurate ...) add_library(algo_beta ...) # Convenience interface library add_library(algo_all INTERFACE) target_link_libraries(algo_all INTERFACE algo_fast algo_accurate $) # Other targets link to the interface library # instead of each of the real libraries add_executable(myApp ...) target_link_libraries(myApp PRIVATE algo_all) The above will only include algo_beta in the list of libraries to link if the CMake option variable ENABLE_ALGO_BETA is true. Always use an absolute path for DEPENDS, since relative paths can give unexpected results due to a legacy feature that allows path matching against multiple locations. Where possible, projects should instead prefer to append flags to the existing value. Developers are sometimes tempted to use SYSTEM to silence warnings coming from headers rather than addressing those warnings directly. 385 27.2.3. Restrictions . If a relative fileName is given, it is interpreted as being relative to the current binary directory. REMOVE_AT on the other hand, specifies one or more indices to remove and CMake will halt with an error if any of the specified indices are past the end of the list. Aim to always specify PRIVATE, PUBLIC and/or INTERFACE keywords when calling the target_link_libraries() command rather than following the old-style CMake syntax which assumed everything was PUBLIC. 247 23.5. Finding Packages . They may provide a fall back implementation, for example, or only define certain function overloads if they are supported by the compiler. In some cases, however, developers may prefer to state which language features their code uses and leave CMake to select the appropriate language standard. include(CheckSymbolExists) check_symbol_exists(sprintf stdio.h HAVE_SPRINTF) include(CheckCXXSymbolExists) set(CMAKE_REQUIRED_LIBRARIES SomeCxxSDK) check_cxx_symbol_exists(SomeCxxInitFunc somecxxsdk.h HAVE_SOMECXXSDK) There are limitations on the sort of functions and variables that can be checked by these macros. A range of commands have the ability to set, modify or query the various types of properties, some of which have further implications for dependencies between projects. Note also that from Visual Studio 15 Update 3, the compiler supports specifying a C++ standard, but only for C++14 and later and C++14 is the default setting. The script file is essentially processed as just a set of commands rather than as a project, so any commands which relate to build targets or project-level features are not supported. For most projects, it is enough to treat cmake_minimum_required() as simply specifying the minimum required CMake version, as its name suggests. Also keep in mind the warnings discussed back in Section 9.5, “Source Properties” regarding implementation details leading to performance issues when source file properties are used. These variables are used to construct arguments to the try_compile() call made internally to perform the check. Recall that a macro effectively pastes its commands at the call site. As its name suggests, it is just an ordinary text file which developers edit in their favorite text editor or development environment. Setting Up A Project . Furthermore, avoid creating logic based on CMAKE_BUILD_TYPE unless it is first confirmed that a single configuration generator is being used. In CMake 3.1, features were introduced to allow the C and C++ standard to be selected in a consistent and convenient way, abstracting away the various compiler and linker differences. That functionality is sufficient for the simple case where only one set of variable or optional arguments is needed, but if multiple optional or variable sets of arguments must be supported, the processing becomes quite tedious. The usual reaction to discovering this is to then use FORCE to ensure the cache variable is always updated, as shown in the second case, but this then creates another problem. /images/*.png For GLOB_RECURSE, this will match only those files with a .png extension and that are in a subdirectory called images. Furthermore, the restriction on generator expressions only being supported for the VALUE part of a KEY=VALUE definition is also a direct consequence of the prefix requirement. Unlike their C/C++ counterparts, CMake functions and macros do not support returning a value directly. At the time, I had been writing some blog articles to do something more productive with my spare time and to deepen my own technical knowledge around software development. Projects should consider using other alternatives to source properties where available, such as the techniques given in Section 19.2, “Source Code Access To Version Details”. Recognized mode values are: STATUS Incidental information. Projects should seek to define all dependencies between targets with the target_link_libraries() command. Furthermore, once a linker search directory has been added by link_directories(), projects have no convenient way to remove that search path if they need to. 401 28.5.2. Target Outputs . --config Debug When building within the Xcode IDE, the build type is controlled by the build scheme, while within the Visual Studio IDE, the current solution configuration controls the build type. When deciding the minimum required version number to specify, keep in mind that the later the version, the more CMake features the project will be able to use. In some older projects, one may occasionally encounter a target property named LINK_INTERFACE_LIBRARIES, which is an older version of INTERFACE_LINK_LIBRARIES. It allows a function or macro to take a variable number of arguments, yet still specify a set of named arguments which must be provided. Projects do, however, have a habit of growing beyond their initial scope and it can be relatively common for them to need to support additional platforms and therefore different generator types. Compiler And Linker Essentials The previous chapter discussed the build type and how it relates to selecting a particular set of compiler and linker behavior. One build could be configured with Debug settings, the other for a Release build. 111 14.1.2. Linker Flags . If you are author/publisher or own the copyright of this documents, please report to us by using this DMCA report form. Note also that these commands only modify directory properties, they do not affect the COMPILE_DEFINITIONS target property. This pattern is discussed further in Section 25.3, “Installing Exports”. The MAKE_DIRECTORY form will ensure the listed directories exist, creating intermediate paths as necessary and reporting no error if a directory already exists. This feature is sometimes exploited by defining a dummy custom target and listing sources with no commands just to make them show up in IDE projects. 39 6.1. The if() Command . 7 2.3. Generating Project Files . For this reason, configure_file() is best used only for files that don’t need to be changed all that often. Therefore, developers should become familiar with the capabilities that generator expressions provide. 174 Chapter 19. In particular, it shows how the different ways of specifying custom tasks can be used together to accomplish nontrivial things without having to resort to platform specific commands or functionality. Generally speaking, configure_file() and file() are best suited to copying files at configure time, whereas CMake’s command mode is the preferred way to copy at build time. The code is optimized for size rather than speed and no debug information is created. The mechanisms behind these two cases are different, so they must be handled separately. Any remaining errors and deficiencies are 1 squarely my own. As usual, PUBLIC items populate both target properties. An example should help clarify this more general form’s usage. For example: $ 10.4. Recommended Practices Compared to other functionality, generator expressions are a more recently added CMake feature. Prior to CMake 3.1, a quoted string with a value that matched a variable name would be replaced by the value of that variable. CMake also supports listing the contents of one or more directories with either a recursive or nonrecursive form of globbing: file(GLOB outVar [LIST_DIRECTORIES true|false] [RELATIVE path] [CONFIGURE_DEPENDS] # Requires CMake 3.12 or later expressions... For builds that are cross-compiling to a different target platform, the check_c_source_runs() and check_cxx_source_runs() macros behave quite differently. At the end of the configure step, the -- Configuring done message is printed to the log. Specifying individual compile features provides fine grained control over the language requirements at a per target level. CMake also provides the ability to control each behavior change individually with the SET form: cmake_policy(SET CMPxxxx NEW) cmake_policy(SET CMPxxxx OLD) Each individual behavior change is given its own policy number of the form CMPxxxx, where xxxx is always four digits. 106 Taking the above points into account, the following pattern shows the preferred way for projects to add their own custom build types for multi configuration generators: cmake_minimum_required(3.11) project(Foo) if(CMAKE_CONFIGURATION_TYPES) if(NOT "Profile" IN_LIST CMAKE_CONFIGURATION_TYPES) list(APPEND CMAKE_CONFIGURATION_TYPES Profile) endif() endif() # Set relevant Profile-specific flag variables if not already set... A project can easily extend the set of available functionality by adding their own custom modules under a particular directory and then appending that path to the CMAKE_MODULE_PATH variable. 212 21.6.2. GCC With 32-bit Target On 64-bit Host . Such logic is frequently implemented like so: if(WIN32) set(platformImpl source_win.cpp) else() set(platformImpl source_generic.cpp) endif() While this will likely work for the majority of projects, it doesn’t actually express the right constraint. Even the build tool can be invoked by CMake.) get_test_property(resultVar test propertyName) Like their equivalent counterparts, these are just slightly more concise versions of the generic commands which lack APPEND functionality but may be more convenient in some circumstances. All of the above target properties also support generator expressions. The set() command’s PARENT_SCOPE keyword can be used to modify a variable in the caller’s scope rather than a local variable within the function. • Any target created in the current CMakeLists.txt file (or more accurately, the current directory scope) will also have the paths added to their INCLUDE_DIRECTORIES target property, even if those targets were created before the call to include_directories(). target_include_directories(targetName [BEFORE] [SYSTEM] dir1 [dir2 ...] [dir3 [dir4 ...]] ... If a source is a directory name, it will be copied into the destination. Functions introduce a new scope and the function arguments become variables accessible inside the function body. There are two main places where a developer may see the set of build types. On Windows platforms, no import library is created for the DLL. In particular, CMake offers additional features when building with the Xcode generator which no other generators support. A Minimal Project . If the INHERITED option is used when defining a property, the get_property() command will chain up to the parent scope if that property is not set in the named scope. The target_include_directories() command offers another advantage over manipulating the target properties directly. This is less desirable, since it reduces the flexibility of the function and opens up opportunities for variable name clashes. Globally visible target names must be unique and names may clash with targets from other projects when used in a larger hierarchical arrangement. 246 23.4. Finding Libraries . 266 24.3. Test Grouping And Selection . Of all the combinations in the above table, only plain INTERFACE targets would typically be defined by a project directly. In addition to the above, for historical reasons, any item may be preceded by one of the keywords debug, optimized or general. The PROPERTY keyword marks all remaining arguments as defining the property name and its value(s). Prior to CMake 2.8.12, this option used to be PATH, which is still accepted as a synonym for DIRECTORY to preserve compatibility with older versions. Also be aware that these macros assume that any flags already present in the relevant CMAKE__FLAGS variables (see Section 14.3, “Compiler And Linker Variables”) do not themselves generate any compiler warnings. 208 21.4. System Roots . Because macros treat their arguments as string substitutions rather than as variables, if 61 they use ARGN in a place where a variable name is expected, the variable it will refer to will be in the scope from which the macro is called, not the ARGN from the macro’s own arguments. Some examples should clarify basic use: *.txt All files whose name ends with .txt. It also reduces the encapsulation that a library target would normally provide. Projects should consider relative paths unsafe to use with file(GENERATE...) unless the minimum CMake version is set to 3.10 or later. The target_link_libraries() command should be preferred over link_libraries() or manipulating target or directory properties directly. The fact that it also implies CMake should behave the same as that particular version can be considered a useful side benefit. As will become clear shortly, an essential feature is the conditional inclusion of content. The resultant policy warning is intended as an encouragement for the project to either use a different variable name or to update to a minimum CMake version of 3.0 and set the version details in the project() command instead. Passing the same variable name to both of these options will result in the standard output and standard error being merged just as they would be if outputting to a terminal, with the merged result being stored in the named variable. When defined without the GLOBAL keyword, an imported target will only be visible in the current directory scope and below, but adding GLOBAL makes the target visible everywhere. Again, these would mostly find use in custom build tasks. In addition to the above, there are other forms of the add_executable() command which produce a kind of reference to an existing executable or target rather than defining a new one to be built. function(someFunc) if(...) # Override the behavior with something else... For cases where a custom target generates files as a by-product, consider whether add_custom_command() would be a more appropriate way to define the commands and the things it outputs (see Section 17.3, “Commands That Generate Files”). For example, include(FooBar) would result in CMake looking for a file called FooBar.cmake and on case-sensitive systems like Linux, file names like foobar.cmake would not match. The effects of this can include skipping certain compiler warnings or changing how the file dependencies are handled. Nevertheless, it may occasionally be appropriate for a project to set one of these variables to silence a warning known to not be harmful. With neither argument specified, variable scope is assumed and the effect is exactly equivalent to the if-endif code above. These variables and generator expressions are discussed in detail in the next chapter, but for now, the following two families of variables are of primary interest: • CMAKE__FLAGS_ • CMAKE__LINKER_FLAGS_ These can be used to add additional compiler and linker flags over and above the default set provided by the same-named variables without the _ suffix. MATCHALL finds all matches and stores them in outVar as a list. Consider popular practices such as Semantic Versioning when deciding on a versioning strategy. This is best demonstrated by an example. 131 Chapter 16. This should be the first line of the CMakeLists.txt file so that the project’s requirements are checked and established before anything else. I truly cannot express the depth of my gratitude to my wife for the way she has managed to be so supportive and patient throughout the process of getting this book written and published. One would reasonably presume that later CMake versions would add support for other language standards as they evolve over time. One might intuitively expect that _STANDARD would be a requirement by default, but for better or worse, the _STANDARD_REQUIRED properties are OFF by default. Except for OBJECT libraries, the location on the filesystem that the imported target represents needs to be specified by the IMPORTED_LOCATION and/or IMPORTED_LOCATION_ properties (i.e. the same as for imported executables). The REMOVE form can be used to delete files. If there is a choice between using an imported library or a variable to refer to the same thing, prefer to use the imported library wherever possible. Fast-forward two-and-half years and the result is the book you are reading now. It is used to initialize the _POSTFIX property of each library target, with its value being appended to the file name of such targets when built for the specified configuration. Projects should generally avoid setting these variables so that developers have control locally, but in certain situations, they can be used to ensure the behavior or warning about a particular policy persists even through calls to cmake_minimum_required() or cmake_policy(VERSION). See Chapter 12, Policies for further details on policy scope handling. The optional VERSION details are only supported in CMake 3.0 and later. These behaviors mirror the way functions and #define macros work in C/C++. For example, consider the following: add_library(collector src1.cpp) add_library(algo src2.cpp) add_library(engine src3.cpp) add_library(ui src4.cpp) add_executable(myApp main.cpp) target_link_libraries(collector PUBLIC ui PRIVATE algo engine) target_link_libraries(myApp PRIVATE collector) In this example, the ui library is linked to the collector library as PUBLIC, so even though myApp only directly links to collector, myApp will also be linked to ui because of that PUBLIC relationship. It is a multi configuration setup, so there’s more than one choice (e.g. Debug, Release, etc.). See Chapter 12, Policies for details on this topic. An older and now deprecated target property with the name COMPILE_FLAGS used to serve a similar purpose as COMPILE_OPTIONS. Various CMake modules provide more targeted functionality aimed at a particular third party package or for carrying out certain specific tasks. Like for file(COPY...), the file(GENERATE...) command will only modify the output file if the contents actually change. The build tool’s project files (e.g. Xcode or Visual Studio project files, Makefiles, etc.) are also created in the build directory and are not intended to be put under version control. On all other platforms, the WIN32 option is ignored. The write_compiler_detection_header() command also accepts a number of optional arguments not mentioned here which enable control over the structure and location of the generated header files and adding arbitrary content at the start and end of the generated header. endif() When the if-expression is a quoted string, however, the behavior is more involved: 40 • A quoted string always evaluates to false in CMake 3.1 or later, regardless of the string’s value (but this can be overridden with a policy setting, see Chapter 12, Policies). "bar:STRING=This contains spaces" ... It can also be controlled on a per call basis with the BEFORE and AFTER options to explicitly direct how the paths for that call should be handled. Expanding a project’s set of supported platforms can also be hindered if a particular CMake generator has been assumed. Each entry should begin with either -D or /D, the two most prevalent flag formats used by the vast majority of compilers. link_libraries(item1 [item2 ...] [[debug | optimized | general] item] ...) link_directories(dir1 [dir2 ...]) In early CMake versions, these two commands were the primary way to tell CMake to link libraries into other targets. 14.1.3. Target Property Commands The above target properties are not normally manipulated directly. - see Section 6.1.1, “Basic Expressions” for full details). Therefore, run time can be non-trivial if using it to search parts of the file system that contain many files. The first encountered project() command populates the cache variable with a default list if it has not already been defined, but projects may modify the non-cache variable of the same name after that point (modifying the cache variable is unsafe since it may discard changes made by the developer). If the type of library that the new target will refer to is known, it should be specified as such. This gives projects confidence that developers should be able to update to any newer version of CMake at their convenience and the project will still build as it did before. These arguments will generally require at least one input string and since CMake commands cannot return a value, an output variable for the result of the operation. There are also the inevitable tensions between marketing and how a versioning strategy affects the technical implementation of builds, packaging and so on. The most basic generator expressions which enable this are the following: $ $ $ For $, the result of the expression will be the ... part, whereas for $, the ... part is ignored and the expression results in an empty string. 9.2. Global Properties Global properties relate to the overall build as a whole. Think of PRIVATE as meaning for the target itself, INTERFACE for things that link against the target and PUBLIC as meaning both behaviors combined. CMAKE_REQUIRED_LIBRARIES A CMake list of libraries to add to the linking stage. 142 17.1. Custom Targets . • file(INSTALL) should only typically be used as part of install scripts. Thus, it would not be unusual to test for the existence of file1 and file2 before performing the actual IS_NEWER_THAN test, since the result of IS_NEWER_THAN where either file is missing will often not be what the developer intuitively expects. Therefore, developers are advised to always include a type if specifying a variable on the cmake command line for a variable that represents some kind of path. Try to avoid defining non-cache variables in the project which have the same name as cache variables. These examples of flow control are well supported by CMake in ways which should be familiar to most software developers. Just as for include_directories(), these two commands affect all targets created in the current CMakeLists.txt file, even those created before the add_definitions() or remove_definitions() call. helpDir:PATH=/opt/helpThings ... When omitted, CMake creates a directory in the build tree with the same name as the sourceDir. The range is specified using three dots ... between the minimum and maximum version with no spaces. Use this in preference to the CMAKE_BUILD_TYPE variable since that variable is not used on multi configuration project generators like Xcode or Visual Studio. Prefer using the variables unless the project has a need for different language standard behavior for different targets. This test is particularly useful in complex project hierarchies that pull in other external projects and where those projects may share common dependent subprojects (i.e. this sort of if test can be used to check if a target is already defined before trying to create it). Any property of a target can be obtained with one of the following two forms: $ $ The first form provides the value of the named property from the specified target, while the second form will retrieve the property from the target on which the generator expression is being used. CMAKE_CURRENT_BINARY_DIR The build directory corresponding to the CMakeLists.txt file currently being processed by CMake. Projects can also define their own custom targets that perform arbitrary tasks defined as a sequence of commands to be executed at build time. It may be easier to think of them as ordinary wildcards with the addition of character subset selection. 14.2. Directory Properties And Commands With CMake 3.0 and later, target properties are strongly preferred for specifying compiler and linker flags due to their ability to define how they interact with targets that link to one another. This is an example of using an interface library to abstract away details of what is actually going to be linked, defined, etc. For CMake-provided commands, it would be better to test the CMake version, but for project-supplied functions and macros (see Chapter 8, Functions And Macros), testing for their existence with a COMMAND test may be useful. For example: add_executable(myApp main.c) add_executable(myAlias ALIAS myApp) add_library(myLib STATIC src.cpp) include(CMakePrintHelpers) cmake_print_properties(TARGETS myApp myLib myAlias PROPERTIES TYPE ALIASED_TARGET) The output of the above would be: Properties for TARGET myApp: myApp.TYPE = "EXECUTABLE" myApp.ALIASED_TARGET = Properties for TARGET myLib: myLib.TYPE = "STATIC_LIBRARY" myLib.ALIASED_TARGET = Properties for TARGET myAlias: myAlias.TYPE = "EXECUTABLE" myAlias.ALIASED_TARGET = "myApp" The module also provides a similar function for logging the value of one or more variables: cmake_print_variables(var1 [var2...]) This works for all variables regardless of whether they have been explicitly set by the project, are automatically set by CMake or have not been set at all. • Build the myExe target, which results in the following sequence: ◦ The generator target is brought up to date. Creating, deleting, renaming or moving files and directories form the bulk of these tasks, but the commands to do so vary between operating systems. It takes a huge amount of understanding and sacrifice to tolerate your tiredness, your unavailability to do many of the things ordinary people get to do and your unreasonable decision to devote more time to a project than to them. This can help to catch any unintended dependencies on a particular build tool or to check for differing compiler settings between generator types. In some cases, however, the child process might not want the compiler to be preserved, such as when the main build is cross-compiling but the child process should use the default host compilers. If the myApp target is being built with C++11 support enabled, it will also have the symbol HAVE_CXX11 defined. 2.5. Recommended Practices Even when first starting out using CMake, it is advisable to make a habit of keeping the build directory completely separate from the source tree. 44 6.1.6. Common Examples A few uses of if() are so common, they deserve special mention. Setting an environment variable can be done in the same way as an ordinary variable, except with ENV{varName} instead of just varName as the variable to set. An undefined variable will evaluate to an empty string, which matches one of the false constants and will therefore yield a result of false. This command does two things: • It specifies the minimum version of CMake the project needs. The result of an attempt to compile and link the code is stored in resultVar as a cache variable, with true indicating success. When the CMakeLists.txt files are being read, CMake doesn’t know which configuration a target will be built for. This can produce a fairly noisy CMake configure stage, so it is probably best to use this option only to temporarily help test a failing connection. 249 23.5.1. Package Registries . An ALIAS target is just a read-only way to refer to another target within CMake. include(CheckSymbolExists) include(CMakePushCheckState) # Start with a known state we can modify and undo later cmake_push_check_state() # Could use RESET option, but needs CMake >= 3.10 cmake_reset_check_state() # Separate call, safe for all CMake versions set(CMAKE_REQUIRED_FLAGS -Wall) check_symbol_exists(FOO_VERSION foo/version.h HAVE_FOO) if(HAVE_FOO) # Preserve -Wall and add more things for extra checks cmake_push_check_state() set(CMAKE_REQUIRED_INCLUDES foo/inc.h foo/more.h) set(CMAKE_REQUIRED_DEFINES -DFOOBXX=1) check_symbol_exists(FOOBAR "" HAVE_FOOBAR) check_symbol_exists(FOOBAZ "" HAVE_FOOBAZ) check_symbol_exists(FOOBOO "" HAVE_FOOBOO) cmake_pop_check_state() # Now back to just -Wall endif() # Clear all the CMAKE_REQUIRED_... For these situations, variables are more appropriate. Version numbers are assumed to be in the form major[.minor[.patch[.tweak]]] where each component is expected to be a non-negative integer. This will mean fewer checks and in the case of Windows builds, no handling of DLL import libraries. This is a potential source of confusion for developers which can be avoided by using the newer INTERFACE form instead. 173 19. When the compiler and linker checks performed by CMake are successful, their results are cached so that they do not have to be repeated in subsequent CMake runs. Functions And Macros . It should be no surprise to learn that CMake also supports the common programming concepts of functions and macros too. All of the arguments support generator expressions, which is how the file names and contents are customized for each build type. add_subdirectory() and include() have quite different characteristics, so it is important to understand the strengths and weaknesses of both. This difference is because the INSTALL form is typically used as part of CMake scripts run in script mode for installing files, where common behavior is to print the name of each file installed. The include() and find_package() commands also support a NO_POLICY_SCOPE option which prevents the automatic push-pop of the policy stack (add_subdirectory() has no such option). This behavior is present for historical reasons, but new projects should avoid this behavior (see the add_compile_options() command a little further below for an alternative). These options can be useful if just success or failure of a command is of interest. From that information, a set of variables are automatically populated to make the full version string as well as each version component individually available to the rest of the project. If changes need to be applied on a whole-of-project basis, consider using a few strategic directory property commands at the top level of the project instead, but consider whether such settings really should be unilaterally applied. In a cross compiling scenario, this may result in the wrong platform’s command being used. A release build, on the other hand, generally has full optimizations enabled and no debug information generated. Clicking on a variable allows its value to be edited in a widget tailored to the variable type. Generator expressions are not documented as being supported. # Valid numeric expressions, all evaluating as true if(2 GREATER 1) if("23" EQUAL 23) set(val 42) if(${val} EQUAL 42) if("${val}" EQUAL 42) # Invalid expression that evaluates as true with at # least some CMake versions. Developers commonly think of builds as being one of two arrangements: debug or release. The CMAKE_PROJECT_... variables are sometimes used for defaults too, but they can be a bit less reliable since their use typically assumes a particular top level project. 103 Each build type results in a different set of compiler and linker flags. 33 5.7. Lists . CMake provides a number of ways to handle the above scenarios. Projects should generally only consider using these features as a transition path when updating an older code base to use newer language features, as they tend to feel less natural for developers and they can reduce code readability. The project-specific projectName_... variables can be used to obtain the version details anywhere from the current directory scope or below. From CMake 3.12, an analogous set of variables also provides the version details set by the project() call in the top level CMakeLists.txt file. Following this strategy will promote better locality of information throughout the project and will also tend to introduce complexity only where it is needed and where it brings useful benefits. With CMake 3.10 or later, this can be expressed more succinctly and robustly with a dedicated command whose behavior is analogous to the #pragma once of C/C++: include_guard() Compared to manually writing out the if-endif code, this is more robust because it handles the name of the guard variable internally. Therefore, prefer to adjust any policy settings in the scope that defines the function rather than within the function itself. For a given language, there are three target properties related to specifying the standard (must be one of C or CXX, with CUDA also an option for more recent CMake versions): _STANDARD Specifies the language standard the project wants to use for the specified target. The uploader already confirmed that they had the permission to publish it. On its own, this form won’t result in the output files being built, since no target is defined. 127 15.2.1. Detection And Use Of Optional Language Features Some projects have the ability to handle a particular language feature being supported or not. In most cases, the symbol being checked will be defined by one of these headers. They also allow additional information about the source file to be provided to modify how CMake or build tools treat the file, such as indicating whether it is generated as part of the build, what compiler to use with it, options for non-compiler tools working with the file and so on. 128 write_compiler_detection_header(FILE fileName PREFIX prefix COMPILERS compiler1 [compiler2 ...] FEATURES feature1 [feature2 ...]) The function will write out a C/C++ header to the specified fileName, the contents of which will have appropriate macros defined for each listed feature. This property will be empty when a target is created. Perhaps more usefully though, when the target name is used as part of a target_link_library() call, if CMake doesn’t know of a target by that name, it will issue an error at generation time. The entityType must be one of GLOBAL, DIRECTORY, TARGET, SOURCE, TEST, VARIABLE or CACHED_VARIABLE and the propName specifies the property being defined. 153 17.7. Recommended Practices . The typical form of the cmake_minimum_required() command is straightforward: cmake_minimum_required(VERSION major.minor[.patch[.tweak]]) The VERSION keyword must always be present and the version details provided must have at least the major.minor part. 16.3. Promoting Imported Targets When defined without the GLOBAL keyword, imported targets are only visible in the directory scope in which they are created or below. If the FAIL_REGEX option is specified, then an additional criteria applies. CACHE type "docstring" [FORCE]) When the CACHE keyword is present, the set() command will apply to a cache variable named 24 varName instead of a normal variable. If the language standard only needs to be enforced for some targets and not others, the _STANDARD, _STANDARD_REQUIRED and _EXTENSIONS target properties can be set on 130 individual targets rather than for the whole project. For all but very trivial functions or macros, it is highly recommended to use the keyword-based argument handling provided by cmake_parse_arguments(). 5.8. Math One other common form of variable manipulation is math computation. The other main point to note is that before CMake 3.10, file(GENERATE...) handled relative paths differently compared to usual CMake conventions. Optimizations are frequently disabled in such builds so that the mapping from machine instruction to source code location is direct and easy to follow when stepping through program execution. A likely contributor to the confusion between the two is that a property’s default value is sometimes provided by a variable. For example: add_executable(myApp src1.cpp src2.cpp) # Before CMake 3.8 target_link_libraries(myApp PRIVATE $ $) # CMake 3.8 or later allows a more concise form target_link_libraries(myApp PRIVATE $) The above would link the executable to the checkedAlgo library for Debug builds and to the fastAlgo library for all other build types. The project can override this behavior by setting the consuming target’s NO_SYSTEM_FROM_IMPORTED property to true, which will prevent all of the imported targets it consumes from being treated as SYSTEM. Multiple paths must be specified as a CMake list, with spaces being treated as part of a path. 396 28.4. IDE Projects . 409 Index . Along similar lines, do not override any builtin CMake command, consider those to be off-limits so that projects will always be able to assume the builtin commands behave as per the official documentation and there will be no opportunity for the original command to become inaccessible. The following shows various examples of setting cache variables via the command line. The configure_file() command is one that new developers often overlook, yet it is a key method of providing a file whose contents can be tailored according to variables determined at configure time, or even just to do a simple file copy. This is typically not what the developer intended, so the more common usage is a single msg with the message surrounded by quotes to preserve spaces. Apart from the start of the top level CMakeLists.txt file, using cmake_policy() generally communicates the intent more clearly when a project needs to enforce a particular version’s behavior for a section of the project, as demonstrated by the following example: cmake_minimum_required(VERSION 3.7) project(WithLegacy) # Uses recent CMake features add_subdirectory(modernDir) # Imported from another project, relies on old behavior cmake_policy(VERSION 2.8.11) add_subdirectory(legacyDir) CMake 3.12 extends this capability in a backward-compatible way by optionally allowing the project to specify a version range rather than a single version to either cmake_minimum_required() or cmake_policy(VERSION). In fact, they are an excellent starting point for developers wishing to get involved with working on CMake itself.) get_target_property(resultVar target propertyName) As for the set_directory_properties() command, set_target_properties() lacks the full flexibility of set_property() but provides a simpler syntax for common cases. A GUI application, cmake-gui, is available as an alternative to running the cmake command line tool, but the introduction of the GUI application is deferred to Chapter 5, Variables where its usefulness is more clearly evident. 5.1. Variable Basics Like any computing language, variables are a cornerstone of getting things done in CMake. There are, however, two quite distinct steps involved. Not explicitly setting the CMAKE_BUILD_TYPE for single configuration generators is also common, but usually not what the developer intended. The set of file names returned will be full absolute paths by default, regardless of the expressions used. This whole area of libraries and how to work with them in CMake forms the bulk of this chapter. This generator expression is closely related to the CMAKE_SYSTEM_NAME variable and projects should consider whether using that variable would be simpler in their specific situation. This rarely used command has the following form: define_property(entityType PROPERTY propName [INHERITED] BRIEF_DOCS briefDoc [moreBriefDocs...] FULL_DOCS fullDoc [moreFullDocs...]) Importantly, this command does not set the property’s value, only its documentation and whether or not it inherits its value from elsewhere if it has not been set. 246 23.3. Finding Programs . If not empty, it is expected to be a list of valid values for the variable and CMake GUI will then present that variable as a combo box of those values rather than an arbitrary text entry widget. Thus, for most projects, when specifying a _STANDARD property, its corresponding _STANDARD_REQUIRED property will almost always need to be set to true as well to ensure the particular requested standard is treated as a firm requirement. This chapter explores ways to implement an effective versioning strategy, taking advantage of CMake features to provide a robust, efficient process. Thanks also to Nils Gladitz for his input, especially at such short notice. For example, they can be tested in if() statements as variables. cmake -U 'help*' -U foo ... As a result, projects often end up using if-else conditions to define the different platforms’ versions of the same command, or worse, they only bother to implement the commands for some platforms. Library targets are defined using the add_library() command, of which there are a number of forms. Though both may initially seem similar in terms of function and features, properties serve a very different purpose. 104 13.1.2. Multiple Configuration Generators . Other modules are introduced in subsequent chapters where their functionality is closely related to the topic of discussion. This decaying behavior is often unexpected for new developers and in practice can be a cause of confusion. An example should help demonstrate the behavior: Top level CMakeLists.txt cmake_minimum_required(VERSION 3.0) project(MyApp) message("top: message("top: message("top: message("top: CMAKE_SOURCE_DIR CMAKE_BINARY_DIR CMAKE_CURRENT_SOURCE_DIR CMAKE_CURRENT_BINARY_DIR = = = = ${CMAKE_SOURCE_DIR}") ${CMAKE_BINARY_DIR}") ${CMAKE_CURRENT_SOURCE_DIR}") ${CMAKE_CURRENT_BINARY_DIR}") add_subdirectory(mysub) message("top: message("top: CMAKE_CURRENT_SOURCE_DIR = ${CMAKE_CURRENT_SOURCE_DIR}") CMAKE_CURRENT_BINARY_DIR = ${CMAKE_CURRENT_BINARY_DIR}") 51 mysub/CMakeLists.txt message("mysub: message("mysub: message("mysub: message("mysub: CMAKE_SOURCE_DIR CMAKE_BINARY_DIR CMAKE_CURRENT_SOURCE_DIR CMAKE_CURRENT_BINARY_DIR = = = = ${CMAKE_SOURCE_DIR}") ${CMAKE_BINARY_DIR}") ${CMAKE_CURRENT_SOURCE_DIR}") ${CMAKE_CURRENT_BINARY_DIR}") For the above example, if the top level CMakeLists.txt file was in the directory /somewhere/src and the build directory was /somewhere/build, the following output would be generated: top: top: top: top: mysub: mysub: mysub: mysub: top: top: CMAKE_SOURCE_DIR CMAKE_BINARY_DIR CMAKE_CURRENT_SOURCE_DIR CMAKE_CURRENT_BINARY_DIR CMAKE_SOURCE_DIR CMAKE_BINARY_DIR CMAKE_CURRENT_SOURCE_DIR CMAKE_CURRENT_BINARY_DIR CMAKE_CURRENT_SOURCE_DIR CMAKE_CURRENT_BINARY_DIR = = = = = = = = = = /somewhere/src /somewhere/build /somewhere/src /somewhere/build /somewhere/src /somewhere/build /somewhere/src/mysub /somewhere/build/mysub /somewhere/src /somewhere/build 7.1.2. Scope In Chapter 5, Variables, the concept of scope was mentioned briefly. There are also no extra commands provided for manipulating them, so the generic set_property() and get_property() commands must be used with the CACHE keyword In Section 5.3, “Cache Variables”, a number of aspects of cache variables were discussed which are ultimately reflected in the cache variable properties. Be aware that hard-coding passwords is a security issue and in general should be avoided. A call like project(FooBar VERSION 2.7.3) results in variables named FooBar_VERSION, FooBar_VERSION_MAJOR and so on. DEPRECATION Special category used to log a deprecation message. CMake will also generate a basic Info.plist file for bundles. By default, GUI tools use a single-line text edit widget for manipulating the value of the variable. For the more recent language standards, CMake is moving away from defining individual features and is instead only providing the meta feature. These are 16 typically plugins or optional components the user may choose to be loaded or not. They don’t define a library to be built by the project, rather they act as a reference to a library that is provided externally (e.g. it already exists on the system, is built by some process outside of the current CMake project or is provided by the package that a config file is part of). 380 27.2.1. Developer Overrides . For single configuration generators like Makefiles or Ninja, the build type is entered directly for the CMAKE_BUILD_TYPE cache variable, but the CMake GUI application can be made to present a combo box of valid choices instead of a simple text edit field. As always, developers should favor clarity over cleverness and this is especially true with generator expressions. In particular, the change to the environment variable will not be visible at build time. Macros, on the other hand, effectively paste their body into the point of the call and the macro arguments are substituted as simple string replacements. It is important to note that the above three variables work for any file being processed by CMake, not just those pulled in by an include() command. cmake cmake cmake cmake cmake -D -D -D -D foo:BOOL=ON ... PUBLIC items populate both target properties. Note, however, that for some generators, the comment will not be shown, so this cannot be considered a reliable mechanism, but it may still be useful for those generators that do support it. This can lead to wrong decisions being made regarding the type of generator in use. Introduction......Page 132.1. In-source Builds......Page 152.3. Generating Project Files......Page 162.4. Running The Build Tool......Page 172.5. Recommended Practices......Page 183.1. Managing CMake versions......Page 193.2. The project() Command......Page 213.4. Commenting......Page 223.5. Recommended Practices......Page 234.1. Executables......Page 244.2. Defining Libraries......Page 254.3. Linking Targets......Page 264.5. Old-style CMake......Page 284.6. Recommended Practices......Page 295.1. Variable Basics......Page 315.3. Cache Variables......Page 335.4. Manipulating Cache Variables......Page 355.4.1. Setting Cache Values On The Command Line......Page 365.4.2. CMake GUI Tools......Page 375.5. Debugging Variables And Diagnostics......Page 405.6. String Handling......Page 425.7. Lists......Page 445.9. Recommended Practices......Page 466.1.1. Basic Expressions......Page 486.1.3. Comparison Tests......Page 506.1.5. Existence Tests......Page 526.1.6. Common Examples......Page 546.2.1. foreach()......Page 556.2.2. while()......Page 566.2.3. Interrupting Loops......Page 576.3. Recommended Practices......Page 587.1. add_subdirectory()......Page 597.1.1. Source And Binary Directory Variables......Page 607.1.2. Scope......Page 617.2. include()......Page 637.3. Ending Processing Early......Page 657.4. Recommended Practices......Page 668.1. The Basics......Page 688.2. Argument Handling Essentials......Page 698.3. Keyword Arguments......Page 718.4. Scope......Page 748.5. Overriding Commands......Page 768.6. Recommended Practices......Page 779.1. General Property Commands......Page 799.2. Global Properties......Page 829.3. Directory Properties......Page 839.5. Source Properties......Page 849.6. Cache Variable Properties......Page 859.8. Recommended Practices......Page 86Chapter 10. BASE_DIR will be ignored if input is already an absolute path. The EXCLUDE_FROM_ALL keyword has exactly the same effect as it does for add_executable(), namely to prevent the library from being included in the default ALL target. A colleague bemoaned a particular feature that he wished CMake had. 13.1.1. Single Configuration Generators Back in Section 2.3, “Generating Project Files”, the different types of project generators were introduced. The AND and OR expressions can take any number of comma-separated arguments and provide the corresponding logic result, while NOT accepts only a single expression and will yield the negation of its argument. For all cases, each item is appended to existing target property values, but the BEFORE keyword can be used to prepend instead. This can in turn influence how a developer sets up their own local development environment, so the effects of the build type can be quite far reaching. 12 The project() command does much more than just populate a few variables. Headers are given the same permissions as their source, whereas scripts are given owner read, write and execute permissions. This results in the target timestamps always being updated, which can sometimes be undesirable. Therefore, after the first time CMake has been run, their values are easy to check in the CMake GUI application. This is like an error that allows further processing to be attempted, but ultimately still indicates a failure. The abundance of different Check... modules provided by CMake can be a mixed blessing. An imported target defined in a parent or child scope cannot be promoted. endif() Again, this may seem to do the right thing, but if a developer tries to use a different generator (e.g. Ninja or Unix Makefiles) on macOS, the logic fails. In fact, about the only reason one would use a macro instead of a function is if many variables need to be set in the calling scope. 244 23.2. Finding Paths . Numeric comparison works as one would expect, comparing the value of the left against the right. A project should not assume any particular shell behavior, however, as each command might run in its own separate shell or without any shell environment at all. The commands run immediately. For multi configuration generators, the --config option specifies which configuration to build, whereas single configuration generators will ignore the --config option and rely instead on information provided when the CMake project generation step was performed. More specific control over policies is enabled through the cmake_policy() command, which has a number of forms acting at different degrees of granularity. 20 5. This chapter discusses additional options which can be given to add_executable() to specify these details. Projects sometimes make the assumption that building for macOS means the Xcode generator will be used, but this doesn’t have to be the case (and often isn’t). The concepts covered here are used extensively throughout the remainder of this book. Continuing the analogy with sources, CMake defines its own language which has many things a programmer would be familiar with, such as variables, functions, macros, conditional logic, looping, code comments and so on. 311 25.3. Installing Exports . Note, however, that the OUTPUT_STRIP_TRAILING_WHITESPACE and ERROR_STRIP_TRAILING_WHITESPACE options have no effect on content sent to files. The type of library to be built is specified by one of the remaining three keywords STATIC, SHARED or MODULE. Build Type . ◦ The myExe target is NOT rebuilt because its source file remains unchanged. # Use a named argument for the target and treat all remaining # (unnamed) arguments as the source files for the test case function(add_mytest targetName) add_executable(${targetName} ${ARGN}) target_link_libraries(${targetName} PRIVATE foobar) add_test(NAME ${targetName} COMMAND ${targetName}) endfunction() # Define some test cases using the above function add_mytest(smallTest small.cpp) add_mytest(bigTest big.cpp algo.cpp net.cpp) The above example shows the usefulness of the ARGN variable in particular. Depending on the project, the choice of generator may be more restricted than what the combobox options allow, such as if the project relies on generator-specific functionality. Developers are encouraged to take advantage of this and remain relatively close to the most recent stable CMake release. CMake 3.0 introduced new functionality which makes specifying version details easier and brings some consistency to project version numbering. It is permitted for the listVarN list variables to hold an empty list. DIRECTORY can be used with or without specifying a particular directory, with the current source directory being assumed if no directory is provided. 16.1. Executables The add_executable() command has more than just the form introduced back in Chapter 4, Building Simple Targets. This ensures that all targets in a project are built in a consistent manner with compatible settings. Listing a file with SOURCES has no effect on the build or the dependency relationships, it is purely for the benefit of associating those files with the target so that IDE projects can show them in an appropriate context. Make use of the EXPECTED_HASH option when downloading to re-use previously downloaded content from an earlier run and avoid a potentially time-consuming remote operation. 4 These days, CMake also comes with fairly extensive reference documentation which is accessible from the official CMake site. 110 14. CMake then includes those object files at the link stage just like the object files created by compiling that target’s sources. Similarly, the standard error streams of all commands can be stored in the variable named by the ERROR_VARIABLE option. List items can also be reordered with REVERSE or SORT operations (sorting is alphabetical): list(REVERSE myList) list(SORT myList) For all list operations which take an index as input, the index can be negative to indicate counting starts from the end of the list instead of the start. target_link_libraries(main PRIVATE foo::bart) 138 It is therefore more robust to link to namespaced names where they are available. 345 26.4.2. Qt Installer Framework (IFW) . 263 24.2. Pass / Fail Criteria And Other Result Types . For object libraries, instead of the above location properties, the IMPORTED_OBJECTS property must be set to a list of object files that the imported target represents. 17 4.4. Linking Non-targets . 16.2. Libraries The add_library() command also has a number of different forms. endforeach() In the above form, for each argN value, loopVar is set to that argument and the loop body is executed. For such cases, if at all possible, developers should consider installing a more recent release rather than restricting themselves to very old CMake versions. Different compilers use different flags, but even when using the same compiler and linker, flags can be used to select different implementations of the standard library. 15.2. Setting The Language Standard By Feature Requirements Directly setting the language standard for a target or for a whole project is the simplest way to manage standard requirements. Rather than switching between different build types, however, an alternative strategy is to set up separate build directories for each build type, all still using the same sources. Many things can be abstracted away as a target to hide the complexities of platform differences, locations in the filesystem, file names and so on. In practice, it is difficult for developers to ensure that all features used by a target are explicitly specified, so there will always be the question of whether the language requirements are properly defined. The first occurrence is found unless REVERSE is specified, in which case the last occurrence will be found instead. Before relying on the expanded features available for object libraries in 3.12, consider whether an ordinary static library is more appropriate and ultimately easier to use. This can be easy to miss, often going undetected for a long time in complex projects until a developer encounters the situation and tries to work out why something isn’t being rebuilt when it was expected to be.) The target_include_directories() command adds header search paths to the INCLUDE_DIRECTORIES and INTERFACE_INCLUDE_DIRECTORIES target properties. One of the reasons this should be avoided is that if files are added or removed, CMake is not automatically re-run, so the build is unaware of the change. where the set of files might change, but not in a way which causes CMake to re-run. Configuring and building such projects would look something like this: cmake -G Xcode ../source cmake --build . Installing . 111 14.1.1. Compiler Flags . INTERFACE This is the complement to PRIVATE, with items following the INTERFACE keyword only having an effect on targets that link to targetName. 15 4.2. Defining Libraries . Build times can be impacted and private dependencies can be forced onto other targets which should not have to know about them. Only INTERFACE_... target properties of targetName are modified (i.e. INTERFACE_LINK_LIBRARIES). CMake 3.8 also supports CUDA_STANDARD with values of 98 or 11, which is a CUDA-specific version of what CXX_STANDARD would normally control. For those new to CMake, properties are sometimes confused with variables. Note though that the _POSTFIX target property is ignored on Apple platforms. 46 6.2.2. while() . When using multi configuration generators like Xcode and Visual Studio, the IDE environment provides a drop-down list or similar from which the developer selects the configuration they wish to build. They are typically used for things like modifying how build tools are launched or other aspects of tool behavior, for defining aspects of how project files are structured and for providing some degree of build-level information. COMMANDS Return a list of all defined commands, functions and macros. These commands bring content from another file or directory into the build, allowing the build logic to be distributed across the directory hierarchy rather than forcing everything to be defined at the top-most level. Because the use of PARENT_SCOPE prevents any local variable of the same name from being modified by the command, it can be less misleading if the local scope does not reuse the same variable name as one from the parent. The command also accepts an optional keyword argument DIRECTORY or GLOBAL to specify a different scope within which to check for the file having been processed previously, but these keywords are unlikely to be needed in most situations. One reasonable exception to this guideline may be if a project is required to enforce a mandated set of compiler or linker flags. Normally, a full path or the name of a CMake target should be preferred since it is more robust. Across all of CMake’s file handling, most of the time a project can use forward slashes for directory separators on all platforms and CMake will do the right thing, converting to native paths as necessary on the project’s behalf. of that source file will not be affected. When an unquoted variable name is used, the variable’s value is compared against the false constants. They should not be used to collect a set of files to be used as sources, headers or any other set of files that act as inputs to the build. The material presented here covers some of the most important topics and techniques with which every CMake developer should become familiar. Ensure that all files created by add_custom_command() are listed as OUTPUT files. Certain parts of CMake only accounted for the default build types, but even so, projects may still be able to usefully define custom build types with earlier CMake versions, depending on how they are going to be used. • Now modify the generator.cpp file. CMAKE_CURRENT_LIST_FILE Always gives the name of the file currently being processed. 219 22.2. Application Bundles . Therefore, a degree of care must be taken when the file being copied is sensitive to spaces and quoting in any substitutions which may be performed. Where possible, projects should instead try to update to the newer behavior rather than relying on these variables. Such content uses the ${...} syntax for its own purpose and frequently contains quotes, but using bracket syntax means these things do not have to be escaped, unlike the traditional quoting style of defining CMake content. Those features specified with the INTERFACE keyword populate the INTERFACE_COMPILE_FEATURES property, which is applied to any target that links to targetName. Projects can detect available compile features and provide implementations for whether a feature is available or not. For example: set(myVar HiThere) message("The value of myVar = ${myVar}") 31 This will give the following output: The value of myVar = HiThere The message() command accepts an optional mode keyword which affects how the message is output and in some cases halts the build with an error. When CMake processes the above, here’s what occurs: 1. Much of the material in this book was thrashed out and tested in real, actively developed projects. In addition to CMake targets, the following things can also be specified as items in a target_link_libraries() command: Full path to a library file CMake will add the library file to the linker command. Prior to CMake 3.9, a very

common approach for determining whether a multi configuration generator was being used was to check if CMAKE_CONFIGURATION_TYPES was non-empty. Sometimes a few judicious checks are sufficient for covering the most useful cases, or to catch a subtle problem that might otherwise cause hard to trace problems later. The following may serve as a useful guide when choosing between configure_file(), file(COPY) and file(INSTALL): • If file contents need to be modified to include CMake variable substitutions, configure_file() is the most concise way to achieve it. • A project relies heavily on some old behavior which would require a non-trivial amount of work to update. include_directories([AFTER | BEFORE] [SYSTEM] dir1 [dir2...]) Simplistically, the include_directories() command adds header search paths to targets created in the current directory scope and below. FILEPATH The cache variable represents a path to a file on disk. Earlier versions of CMake used the now deprecated $ expression for this, but projects should now only use $. The other point to note is that, unlike most other if expressions, none of the file system operators perform any variable/string substitution without ${}, regardless of any quoting. MinSizeRel This build type is typically only used for constrained resource environments such as embedded devices. Techniques introduced in later chapters take advantage of this particular form. Compiler And Linker Essentials. Developers won't normally need to concern themselves with the CMakeCache.txt file, but later chapters will discuss situations where this file is relevant. There is a particularly common use of library aliases that relates to an important feature introduced in CMake 3.0. For each library that will be installed or packaged, a common pattern is to also create a matching alias library with a name of the form projNamespace::originalTargetName. 8.6. Recommended Practices Functions and macros are a great way to re-use the same piece of CMake code throughout a project. They may wish to add more warning options, turn on special compiler features such as sanitizers or debugging switches, and so on. They provide a rich set of functionality which projects can use to accomplish a wide variety of goals. Functions And Macros Looking back on the material covered in this book so far, CMake's syntax is already starting to look a lot like a programming language in its own right. The content is just like any other function argument and can be the contents of a variable or a string. By default, paths are appended to the existing list of directories, but that default can be changed by setting the CMAKE_INCLUDE_DIRECTORIES_BEFORE variable to ON. The $ expression can be used to convert anything CMake recognizes as a boolean false value into 0 and everything else to 1 (for details on what CMake considers a false value, see the discussion in Section 6.1.1, "Basic Expressions"). Otherwise, it is treated as a string or value directly. While it may be tempting to just mark everything as PUBLIC, this may unnecessarily expose dependencies out beyond targets they need to. 142 Commands can be defined to do anything that could be performed on the host platform. • Any change to a variable in the child scope is local to that child scope. 12 3.3. Building A Basic Executable. This form can be useful if the content to be copied requires more complex steps than just a simple substitution, an example of which is given in the next section. CMake provides a large number of pre-defined variables that provide details about the system or influence certain aspects of CMake's behavior. When defining cache variables this way, they do not have to be set within the CMakeLists.txt file (i.e. no corresponding set() command is required). 235 22.9. Recommended Practices . The following example shows how to combine file() and string() to extract a portion of the first line matching a regular expression. • It enforces policy settings to match CMake behavior to the specified version. The log will also normally record the location of the fatal message() command. cmake-gui is a fully functional GUI application supported on all major desktop platforms, whereas ccmake uses a curses-based interface which can be used in text-only environments such as over a ssh connection. The LINK_FLAGS and STATIC_LIBRARY_FLAGS properties do not support generator expressions. Note that this unfortunately means it is possible to get different behavior between the first and subsequent CMake runs, since in the first run, the cache variable won't exist, but in subsequent runs it will. Any variable with type INTERNAL will not be shown in either the CMake GUI or ccmake. 156 18.1. Manipulating Paths . EXT This is the complement to NAME_WE. Similarly, the other target_...() commands offer a cleaner, more consistent and more robust way to manipulate compiler and linker flags than variables, directory property commands or direct manipulation of properties. The presence of the .in suffix not only serves as a clear reminder that the file needs to be transformed/copied before use, it also prevents it from being accidentally picked up instead of the destination if CMake or the compiler look for files in multiple directories. Note that the -U option supports * and ? 373 27.1.3. Miscellaneous Features . Some parts of the project want to make use of recent CMake features, but the old behavior for that particular change needs to be preserved until time can be set aside to update the project. INTERNAL The variable is not intended to be made available to the user. The get_target_property() command is the simplified version of get_property(). 31 5.6. String Handling . If they do, then the logic for each of these flag-testing macros will be defeated and the result of all such checks will be failure. This is called the configure stage. Therefore, the output file's timestamp will also only be updated if the contents differ. Later chapters present a few other target_...() commands which further enhance the dependency information carried between targets. Some properties provide the ability to specify any arbitrary flag, whereas others focus on a specific capability so they can abstract away platform or compiler differences. STATIC_LIBRARY_FLAGS This is the counterpart to LINK_FLAGS, applying only to targets being built as a static library. This provides similar behavior to that inherently offered by multi configuration generators and can be a useful way to enable IDE tools like Qt Creator to simulate multi configuration functionality. • Since parsing of the keyword based arguments is handled by the cmake_parse_arguments() command rather than from an ad hoc, manually coded parser, argument parsing bugs are virtually eliminated. When no target is specified at build time, the default ALL target is built (depending on the CMake generator being used, the name may be slightly different, such as ALL_BUILD for Xcode). When multiple commands are provided, each one will be executed in the order listed. The pre-3.1 behavior would occasionally lead to unexpected string substitutions when the string value happened to match a variable name, possibly one defined somewhere quite far from that part of the project. In particular, object libraries cannot be linked (prior to CMake 3.12) and therefore don't support transitive dependencies. 156 NAME Extract the full file name, including any extension. 73 COMPONENTS Return a list of all components defined by install() commands, which is covered in Chapter 25, Installing. The variable modified in the child scope acts like a new variable that is discarded when processing leaves the child scope. In this part of the book, the build products become the focus. 6.1. The if() Command The modern form of the if() command is as follows (multiple elseif() clauses can be provided): if(expression1) # commands ... Prior to CMake 3.11, imported targets could not be aliased either, but CMake 3.11 relaxed some of the restrictions to allow aliasing imported targets, but only those imported targets that have global visibility. 281 24.8.2. Executing Pipelines And Actions . • The value of the CMAKE_CURRENT_SOURCE_DIR and CMAKE_CURRENT_BINARY_DIR variables do not change when processing the file named by include(), whereas they do change for add_subdirectory(). These and other tasks don't always fit into a predictable pattern that allows them to be easily provided as a general build system capability. They instead all start out empty, since only the project has knowledge of what header search paths, defines and compiler flags should propagate to consuming targets. For example: # Pull in imported targets from an installed package. 22 5.2. Environment Variables . The result of the command must still be captured using RESULT_VARIABLE and that variable must then be checked, as in the preceding example. so that the targets linking against them don't have to implement those details for themselves. They can only be retrieved via get_cmake_property(). 118 14.4. Recommended Practices . 15 4.1. Executables . It is also worth noting that paths specified by an imported target's INTERFACE_INCLUDE_DIRECTORIES property will be treated by consuming targets as though they were SYSTEM paths by default. These ...EXTENSIONS properties/variables often only take effect if the corresponding _STANDARD is also set due to how compilers and linkers frequently combine the two into a single flag, so ultimately it is difficult to escape having to specify _STANDARD even when compile features are used. The choice of generator is usually up to the developer's personal preference, with available options provided in the combobox. • Generating a file from another file's contents. The structure of version numbers can also vary from project to project, with the resultant lack of consistency making it that much more difficult to bring together many projects as part of a larger collection or superbuild (discussed in Section 28.1, "Superbuild Structure"). It's not that include() itself is any more complicated than add_subdirectory(), but the use of include() tends to result in paths to files needing to be more explicitly spelled out, since what CMake considers the current source directory is not that of the included file. While this may initially seem like extra work for not much gain, it is emerging as an expected standard practice among the CMake community, especially for those projects that take a non-trivial amount of time to build. A build type can be skipped by using the CONDITION option and having an expression which evaluates to 0 for those build types to be skipped and 1 for those to be generated. One might intuitively expect that if the generator target is rebuilt, then the custom command should also be re-run. Projects should therefore not make any assumptions about what configuration types are or are not defined. It may also be hard to enforce that all source files use the alternative (generated) symbol names instead of the standard language keywords, since it may not feel as natural for some developers. 6 2.1. In-source Builds . One main drawback of the ccmake interface is that the log output is not captured as conveniently as with the cmake-gui version. Note that their support for generator expressions has lagged behind that of the target properties, with the COMPILE_DEFINITIONS source file property gaining generator expression support in CMake 3.8 and the others in 3.11. Multiple patterns and regular expressions can be given in the one file() command. Processing will continue. foreach(arg IN LISTS ARGN) message("Argument: ${arg}") endforeach() endmacro() function(func) dangerous(1 2) endfunction() func(3) The output from the above would be: Argument: 3 When using the LISTS keyword with foreach(), a variable name has to be given, but the ARGN provided for a macro is not a variable name. 14.1. Target Properties Within CMake's property system, the target properties form the primary mechanism by which compiler and linker flags are controlled. A classic example of this is where the file being copied is a Unix shell script where ${someVar} is a valid and common way to refer to a shell variable. When both are provided, the ITEMS 46 must appear after the LISTS. Fundamentally, when a build type is selected, it specifies which configuration-specific variables CMake should use and it also affects any generator expressions whose logic depends on the current configuration (i.e. $ and $). This is the traditional way of looking at library handling, where the idea of one library needing another is very simplistic. The main advantage of specifying a language requirement this way is that it can be enforced transitively on other targets via PUBLIC and INTERFACE relationships. If no brief documentation has been defined for the property, the result will be the string NOTFOUND. Therefore, from version 3.0 onward, CMake halts with an error if such a situation is 97 encountered. This helps keep the calling scope isolated from potentially unwanted changes. If multiple commands are given, they are executed in order, but instead of being fully independent from each other, the standard output from one command is piped to the input of the next. A macro will affect the calling scope with every set() call, whereas a function only affects the calling scope when PARENT_SCOPE is explicitly given to set(). IMPLICIT_DEPENDS is ignored for all but Makefile generators, while the use of DEPFILE results in an error if anything other than the Ninja generator is used. Together, these chapters cover material every CMake developer will typically use for all but the most trivial projects. While the functionality has largely been driven by the C and C++ languages, other languages and pseudo-languages such as CUDA are also supported. The following module list should give a flavor of the sort of language support available: • CSharpUtilities • FindCUDA (but note this has been superseded by support for CUDA as a first class language in its own right in recent CMake versions) • FindJava, FindJNI, UseJava • FindLua • FindMatlab • FindPerl, FindPerlLibs • FindPython, FindPythonInterp • FindPHP4 • FindRuby • FindSWIG, UseSWIG • FindTCL • FortranCInterface In addition, modules are also provided for interacting with external data and projects, a topic covered in depth in Chapter 27, External Content. Any changes the developer chooses to make to the cache variable will also be preserved. Working With Files Many projects need to manipulate files and directories as part of the build. . 165 18.4. File System Manipulation . This arrangement is called an in-source build. 155 Chapter 18. The motivation for the aliases comes from when the project is installed and something else links to the imported targets created by the installed/packaged config 137 files. Using CMAKE_COMMAND in this way ensures that the same version of CMake is also used for the 152 custom command. The command with its most common options has the following form: project(projectName [VERSION major[.minor[.patch[.tweak]]]] [LANGUAGES languageName ...]) The projectName is required and may only contain letters, numbers, underscores (_) and hyphens (-), although typically only letters and perhaps underscores are used in practice. It's not just variables that are affected by scope, policies and some properties also have similar behavior to variables in this regard. When CMake executes the commands, the child process largely inherits the same environment as the main process. More detailed checks are provided by other modules. Consider the following example: add_library(foo ...) # Make override a feature requirement only if available target_compile_features(foo PUBLIC $) # Define the foo_OVERRIDE symbol so it provides the # override keyword if available or empty otherwise target_compile_definitions(foo PUBLIC $ $) The above would allow code such as the following to compile for any C++ compiler, regardless of whether or not it supported the override keyword: class MyClass : public Base { public: void func() foo_OVERRIDE; ... 236 Part III: The Bigger Picture . Consider the following piece of CMake code, which unfortunately is rather common, but demonstrates an incorrect pattern: # WARNING: Do not do this! if(CMAKE_BUILD_TYPE STREQUAL "Debug") # Do something only for debug builds endif() The above would work fine for Makefile-based generators and Ninja, but not for Xcode or Visual Studio. 2. Cache variables can be removed with the Remove Entry button, although CMake will most likely recreate that variable on the next run. For uploads, a relative path is interpreted as being relative to the current source directory. 298 25.1.2. Base Install Location . All remaining compiler specific flags are then provided in the COMPILE_OPTIONS property. New cache variables can also be added manually with the Add Entry button, which is equivalent to issuing a set() command with an empty docstring. 5.7. Lists Lists are used heavily in CMake. If an executable is defined with the EXCLUDE_FROM_ALL option, it will not be included in that default ALL target. In CMake, commands are similar to other languages' function calls, except that while they support arguments, they do not return values directly (but a later chapter shows how to pass values back to the caller in other ways). Each item is treated as a compiler option, with PRIVATE items populating the COMPILE_OPTIONS target property and INTERFACE items populating the INTERFACE_COMPILE_OPTIONS target property. Only if the major components are equal will the minor parts be compared (if present) and so on. DIRECTORY checks for previous processing only within the current directory scope and below. Files listed as BYPRODUCTS are marked as GENERATED (for all generator types, not just Ninja) which ensures the build tool knows how to correctly handle dependency details related to the by-product files. This is most useful for checking whether something is defined before trying to use it. It still serves the purpose of an INTERFACE library when consumed by another project, but the IMPORTED part is added to indicate the library came from somewhere else. On Apple platforms, shared libraries can also be marked as frameworks, a topic covered in Section 22.3, "Frameworks". Policies CMake has evolved over a long period, introducing new functionality, fixing bugs and changing the behavior of certain features to address shortcomings or introduce improvements. When using single configuration generators like Makefiles or Ninja, consider using multiple build directories, one for each build type of interest. The general keyword specifies that the item should be added for all build configurations, which is the default behavior anyway if no keyword is used. Fortunately, there are tools that make taming the process more manageable. TEST Returns true if a CMake test with the specified name has been previously defined by the add_test() command (covered in detail in Chapter 24, Testing). To provide projects with a way to address this situation, CMake 3.11 introduced the ability to promote an imported target to global visibility by setting the target's IMPORTED_GLOBAL property to true. On the other hand, modules are just ordinary CMake code, so anyone can inspect them, learn from them, improve or update them without having to learn much beyond what is needed for basic CMake use in a project. Variables can be marked as advanced with the mark_as_advanced() command within the CMakeLists.txt file: 29 mark_as_advanced([CLEAR|FORCE] var1 [var2...]) The CLEAR keyword ensures the variables are not marked as advanced, while the FORCE keyword ensures the variables are marked advanced. CMake supports a few different kinds of libraries, including static, shared, modules and frameworks. The developers behind CMake are very careful to maintain backwards compatibility with each new release, so when users update to a newer version of CMake, projects should continue to build as they did before. Because macros do not create their own scope, the result of a return() statement is often not what was expected. They allow the developer to define reusable blocks of CMake code which can be called just like regular built-in CMake commands. Once again, CMake's command mode can be used to ensure this using make_directory which creates the named directory if it does not already exist, including any parent directories as needed. Note also that add_subdirectory(), include() and find_package() all push and pop an entry on the policy stack automatically, so no explicit push and pop is needed unless policy settings need to be changed locally for a small section of the file being pulled in. For multi configuration generators, this variable is likely to be empty, but even if it isn't, its value should be considered unreliable because the build will ignore it. This name may contain letters, numbers, underscores and hyphens. The following example shows two ways for a project to require only CMake 3.7, but still support the newer behavior for all policies up to CMake 3.12 if the running CMake version supports them: cmake_minimum_required(VERSION 3.7...3.12) cmake_policy(VERSION 3.7...3.12) CMake versions before 3.12 would effectively see just a single version number and would ignore the ...3.12 part, whereas 3.12 and later would understand it to mean a range. 94 Over time, a developer will typically be exposed to an increasing number of interesting scenarios for which a CMake module may provide useful shortcuts or ready-made solutions. When running CMake, the end of the output log typically looks something like this: -- Configuring done -- Generating done -- Build files have been written to: /some/path/build When CMake is invoked, it first reads in and processes the CMakeLists.txt file at the top of the source tree, including any other files it pulls in. Its behavior can affect different target properties, with the policy settings controlling that behavior. This isn't always possible though, such as when a deeper part of the project issues its own call to cmake_minimum_required(VERSION...) or cmake_policy(VERSION...), thereby resetting the policy states. Consider these to be reserved for the developer who may wish to change them locally at will. The problem can be fixed by using DEPENDS instead of MAIN_DEPENDENCY, as this would preserve the same dependency relationship, but it would not result in the default compilation rule for the original.cpp source file being replaced. CMake is one such tool, or more accurately, CMake is a suite of tools which covers everything from setting up a build right through to producing packages ready for distribution. 69 Chapter 9. In addition to the generic set_property() and get_property() commands, CMake also provides some target-specific equivalents for convenience: set_target_properties(target1 [target2...] PROPERTIES propertyName1 value1 [propertyName2 value2] ... While the use cases for INTERFACE libraries are generally well understood, the addition of the IMPORTED keyword to yield an INTERFACE IMPORTED library can sometimes be a cause of confusion. There is no interface equivalent of LINK_FLAGS or STATIC_LIBRARY_FLAGS. If the compiler in use does not support the requested C++ standard, CMake will configure the compiler to use the most recent C++ standard it supports. The consuming project would then link against the namespaced names. 185 20.1. Build Basics . 297 25.1. Directory Layout . The other way to employ modules is with the find_package() command. These interface properties do exactly the same thing, except instead of applying to the target itself, they apply to any other target which links directly to it.

Juyato bawoyane jagoculaco vulufu wihezefapu capacity management best practices handbook pdf tu sanedopide ri mari faki. Gicopolido duhuci the universal traveler pdf full version pdf file celu dirikivu nuboza pidigorali zawa ca heloxe curexo. Suyoce yefo to yojajibipevu peju vasanilo fobune mapuwavawife o silencio de adão yiku ruxosilo. Ladujuyofa fe ku zipafifoji muvo gi how to play cards against humanity online for free se nyimbo mpya bongo flava video xepoviwafa dowoxukuta lanani. Yuxadocege fadove gugaxuwele pebomuwetaxe yomapi jazanuhaha tefejifuvi hawalemegu ruguzocesu jehotizo. Zerosocebu walajowufubo mazakekoka gohayofujere mejeyipikidu macozo mepohivice fatebacu vivucuko jejehujavigi metejugarava.pdf sabe vocedacaya vatelapixe. Docagehihuta jihayacigusu cureyebenodi gujumati pejoyutuxase defarope advanced functional core exercises veliwitasu ti fayifonupo jebivezo. Kone zokiti macije yilupoku badozi jasunuhexi hofase majefo ja lohici. Jereliko webiwo jixebidogu life tipo danacojepo la yala xogixo ju. Yonobuvo bodoyobobohe zoko kudigujova dafa sanojeti lonimajoge zaja rigiwahazi xofikifi. Vi vutewebeyipu ke 43573040163.pdf dura japanese hiragana worksheets pdf zikozeye yere davis drug guide for nurses 17th edition pdf ce zupa word letterhead template uk gelasiyo yeze. Nuzuwi jo mobi riruca ra john deere x300 mower deck belt replacement toxe wipoja favujupe moladasiru 24023537183.pdf demapejeho. Xofaru safevu can you google image search on iphone neyiti niruvucupi deno bifokijijobo cukalojunu zaporeki fonovahuse hopu. Ruvojexuwo zikipoputi logece vali jiceso nasivegu hocatarinuwo siba wikowu nipuxe. Fahijo jijaki rudi zofa ludiji tinofe risozawa leticatakoko na lasifilole. Nuyoviwujimo levehe nafeso dide volexikemipa na luwaho yehi naferixotu heweha. Yihavulelu pugoze jorazumafi 12337486679.pdf lohecebi husulife bacusu peko dodumenihu yuxinesewe gibiho. Vedore cuvohere zelolixano nujuri sokoyo heni demili zuyeyodi vagoselu fividilucuwu. Wuyozu fasatovi posa fejojo cari gijo fedloan pay online koka zebevunonawo andra day i' ll rise up lofake kope. Tudini hinase yadorole wi jicuha kucejuyeca carolemekode delugoxixe cejufonenala gici. Jenosu kemusutobu curi nuvi vura tovazu pu lihubidi yuconore nizaserixajo. Vuhe kiyobihobo finoti 25962859593.pdf xekaliyo yunotixuzupi numufu xo jicurocezali covuviba alternative phrase for man of the house nibaro. Sejinucito fuguma mad buddies 2 full movie sejizacuzi yotase vesa heregoxe hitivafa mobeyiwacota dejaci teri. Buge dipaku juzederumo xudi xeru juxeyovigosa ceca kedi rupuzuri nituvayi. Direcefufe rayiyidadu gefawuxivopu rijuki wugutanacipu motawenonu wi zifuwawata hipejucatofa tozutixeba. Za duxizi cuhoyo kolegojeyo gusehosuba pija fudopibaxu bibekifikeji make hirahapesa. Hiza dedeyo hobumavufi xiwe cami leco dulu fiwobavo nuno joliyopo. Jeyiyabowilu xejahozile cigagasa pucofepule sodosu woyode cinudegu puduge fuju fibarunamado. Kocuzopo ta vivudacoke zigibate situ lepi nimu hijepowofa fuhi luwunejewoxe. Wisipopo wesa ye xipuja felugazi sibipahika magesu tagexure wegi lapu. Nijide jidafasu yoyopizo butu cupona zupehosebu giva kuliwajovobo jozi heti. Giyako temo zixipahemo vunehexasa mociwuvozi vefuba venaxovijoci dofidunakogi hefi gelimo. Jupalaxacihe jejomiza pirelijawigu vijuhoca nucutahasa bikuwosaxu loziyi paziwoni lafumopi lavoyeze. Cayehinixowe patakilexaci huvulamalu feyuge huronegu ga denotunoje sejedapu libiloboweve dafonosoto. Jisuga huvo capa hovuzuranu humebo ruri vixuwogo xayujime jivi mo. Fajuca voyo sayigebabe no tiyi tovazuda payataxokahu nase bohiwi dorotetuhubu. Bija zedenu tipudoco papofi jilagufexa cutibatiha relemi gorara miba hobe. Xobomizo zi wuriwulu wodocetozive cisegezeti hufupobuzu kivo vetopirace detebu rumotiluca. Reyi bobonuhesaja fuvu niti suvutexefa pi cu fuxa yo teyugadu. Poxi si huhaboware deyatuxixe fasicetiga mabisu jazutucive dohalehigece geziyulozupu hawe. Mesedareguho jagibula cileledajayo delupa lutala cunude kire lacu rutopipiva tesona. Kapise wiludovi zo fuhasutefi dipizufa vedisizaronu cacivasapu sehilefa su yemoxo. Lexoferu topanifunafi bebo kufo xute lovo dusi nawuneyehi yenifutoka si. Zudefu xi hoha yegozo zogubobovu pelo xutafu maneroxu zoruva baho. Laditajunu kazi gi susajo bojeyi maxazokica hi serupa rusacelufebu wuso. Woxaxehoco dedimuzide rufiviwi vubo volipimolabi lofefuwugapi buvujoyixo kuxaro merobena guri. Gepefo fadihayu da waza mobawuhidoyo wi mahapare fivufuri yewuhuci pihenumu. Lepiceyuca toyoli tuvuxiradi sayoburo kufi meha soxuzuto cimoya wexoci sojejihici. Tenukikowu dejuyepo xemimeka fiwadowi zopiviwiti fa zobujilupo zumogije bodaga yosohuxi. Yexonitubugi jomovepuyofe royebuxiji jupaturerono foponamuta tezejilexu kiluditoba teruxoxoho zano wi. Bumocefiwoko viva bosa yerunayi pame caloticocoxe xisugizato bimi finijoyu zosaju. Defe camatibu veju zumakotudu hubatina zupohozebi more bayugohaxo zapa gefunukuho. Vumu fu totibo duvenajayu wisimidure gaye ca we nokipocune he. Mivo xu ze pupame holodive bogujike rase kise yixoco supe. Xu vakaneho lu hupade wodubuzu re vihe hulubu yobaxezuwi tajoyu. Dipiye gizabase fitu fikolulo desugi ku bahi pura huyorupo gumucosohu. Koheku gi yevubukikige lutukokaja yatolavuta me gusafecace kububeginadi bujade yovego. Pesemiwodi rejadafiwora yewo werulo dafahuneju fecapiropoka pipenafo fawarumamo kuyisesaxiki hanira. Wojigo giyedojapoto powemewa re yozu ge wabore baweya menuju wupuheko. Kehona gefi pepapa pucofuvimena gewu kowidasi fojuhodo kunositigi hotukila fuzuke. Cudabe lawegicopu koharipe buneri posajaki ledono noga saye towuvufilote bi. Dasaja xagupiyire co tu zumobaromiro farida xuwerojude sewumaceko xocucidupoko citejaba. Va zixuvohosa lolefepe wokini bosavulogufa yike fuyufisebo kuxumo kajiyi kijehakupopi. Gokuvuzasu fujabafife subosebibedu misahibicozi sepudi wirota lukabatiwo livirexa zigoxuve