NAME

cmake-modules - CMake Modules Reference

ALL MODULES

AddFileDependencies

ADD_FILE_DEPENDENCIES(source_file depend_files...)

Adds the given files as dependencies to source_file

BundleUtilities

Functions to help assemble a standalone bundle application.

A collection of CMake utility functions useful for dealing with .app bundles on the Mac and bundle-like directories on any OS.

The following functions are provided by this module:

```
fixup_bundle
copy_and_fixup_bundle
verify_app
get_bundle_main_executable
get_dotapp_dir
get_bundle_and_executable
get_bundle_all_executables
get_item_key
clear_bundle_keys
set_bundle_key_values
get_bundle_keys
copy_resolved_item_into_bundle
copy_resolved_framework_into_bundle
fixup_bundle_item
verify_bundle_prerequisites
verify_bundle_symlinks
```

Requires CMake 2.6 or greater because it uses function, break and PARENT_SCOPE. Also depends on GetPrerequisites.cmake.

```
FIXUP_BUNDLE(<app> <libs> <dirs>)
```

Fix up a bundle in-place and make it standalone, such that it can be drag-n-drop copied to another machine and run on that machine as long as all of the system libraries are compatible.

If you pass plugins to fixup_bundle as the libs parameter, you should install them or copy them into the bundle before calling fixup_bundle. The libs parameter is a list of libraries that must be fixed up, but that cannot be determined by otool output analysis. (i.e., plugins)

Gather all the keys for all the executables and libraries in a bundle, and then, for each key, copy each prerequisite into the bundle. Then fix each one up according to its own list of prerequisites.

Then clear all the keys and call verify_app on the final bundle to ensure that it is truly standalone.

```
COPY_AND_FIXUP_BUNDLE(<src> <dst> <libs> <dirs>)
```

Makes a copy of the bundle <src> at location <dst> and then fixes up the new copied bundle in-place at <dst>...

```
VERIFY_APP(<app>)
```

Verifies that an application <app> appears valid based on running analysis tools on it. Calls message(FATAL_ERROR if the application is not verified.

```
GET_BUNDLE_MAIN_EXECUTABLE(<bundle> <result_var>)
```

The result will be the full path name of the bundles main executable file or an error: prefixed string if it could not be determined.

```
GET_DOTAPP_DIR(<exe> <dotapp_dir_var>)
```

Returns the nearest parent dir whose name ends with .app given the full path to an executable. If there is no such parent dir, then simply return the dir containing the executable.

The returned directory may or may not exist.

```
GET_BUNDLE_AND_EXECUTABLE(<app> <bundle_var> <executable_var> <valid_var>)
```

Takes either a .app directory name or the name of an executable nested inside a .app directory and returns the path to the .app directory in <bundle_var> and the path to its main executable in <executable_var>

```
GET_BUNDLE_ALL_EXECUTABLES(<bundle> <exes_var>)
```

Scans the given bundle recursively for all executable files and accumulates them into a variable.

```
GET_ITEM_KEY(<item> <key_var>)
```

Given a file (item) name, generate a key that should be unique considering the set of libraries that need copying or fixing up to make a bundle standalone. This is essentially the file name including extension with . replaced by _

This key is used as a prefix for CMake variables so that we can associate a set of variables with a given item based on its key.

```
CLEAR_BUNDLE_KEYS(<keys_var>)
```

Loop over the list of keys, clearing all the variables associated with each key. After the loop, clear the list of keys itself.

Caller of get_bundle_keys should call clear_bundle_keys when done with list of keys.

```
SET_BUNDLE_KEY_VALUES(<keys_var> <context> <item> <exepath> <dirs>
<copyflag>)
```

Add a key to the list (if necessary) for the given item. If added, also set all the variables associated with that key.

```
GET_BUNDLE_KEYS(<app> <libs> <dirs> <keys_var>)
```

Loop over all the executable and library files within the bundle (and given as extra <libs>) and accumulate a list of keys representing them. Set values associated with each key such that we can loop over all of them and copy prerequisite libs into the bundle and then do appropriate install_name_tool fixups.

```
COPY_RESOLVED_ITEM_INTO_BUNDLE(<resolved_item> <resolved_embedded_item>)
```

Copy a resolved item into the bundle if necessary. Copy is not necessary if the resolved_item is the same as the resolved_embedded_item.

```
COPY_RESOLVED_FRAMEWORK_INTO_BUNDLE(<resolved_item> <resolved_embedded_item>)
```

Copy a resolved framework into the bundle if necessary. Copy is not necessary if the resolved_item is the same as the resolved_embedded_item.

By default, BU_COPY_FULL_FRAMEWORK_CONTENTS is not set. If you want full frameworks embedded in your bundles, set BU_COPY_FULL_FRAMEWORK_CONTENTS to ON before calling fixup_bundle. By default, COPY_RESOLVED_FRAMEWORK_INTO_BUNDLE copies the framework dylib itself plus the framework Resources directory.

```
FIXUP_BUNDLE_ITEM(<resolved_embedded_item> <exepath> <dirs>)
```

Get the direct/non-system prerequisites of the resolved embedded item. For each prerequisite, change the way it is referenced to the value of the _EMBEDDED_ITEM keyed variable for that prerequisite. (Most likely changing to an @executable_path style reference.)

This function requires that the resolved_embedded_item be inside the bundle already. In other words, if you pass plugins to fixup_bundle as the libs parameter, you should install them or copy them into the bundle before calling fixup_bundle. The libs parameter is a list of libraries that must be fixed up, but that cannot be determined by otool output analysis. (i.e., plugins)

Also, change the id of the item being fixed up to its own _EMBEDDED_ITEM value.

Accumulate changes in a local variable and make *one* call to install_name_tool at the end of the function with all the changes at once.

If the BU_CHMOD_BUNDLE_ITEMS variable is set then bundle items will be marked writable before install_name_tool tries to change them.

```
VERIFY_BUNDLE_PREREQUISITES(<bundle> <result_var> <info_var>)
```

Verifies that the sum of all prerequisites of all files inside the bundle are contained within the bundle or are system libraries, presumed to exist everywhere.

```
VERIFY_BUNDLE_SYMLINKS(<bundle> <result_var> <info_var>)
```

Verifies that any symlinks found in the bundle point to other files that are already also in the bundle... Anything that points to an external file causes this function to fail the verification.

CheckCCompilerFlag

Check whether the C compiler supports a given flag.

CHECK_C_COMPILER_FLAG(<flag> <var>)

```
<flag> - the compiler flag
<var> - variable to store the result
```

This internally calls the check_c_source_compiles macro and sets CMAKE_REQUIRED_DEFINITIONS to <flag>. See help for CheckCSourceCompiles for a listing of variables that can otherwise modify the build. The result only tells that the compiler does not give an error message when it encounters the flag. If the flag has any effect or even a specific one is beyond the scope of this module.

CheckCSourceCompiles

Check if given C source compiles and links into an executable

CHECK_C_SOURCE_COMPILES(<code> <var> [FAIL_REGEX <fail-regex>])

```
<code> - source code to try to compile, must define 'main'
<var> - variable to store whether the source code compiled
<fail-regex> - fail if test output matches this regex
```

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CheckCSourceRuns

Check if the given C source code compiles and runs.

CHECK_C_SOURCE_RUNS(<code> <var>)

```
<code> - source code to try to compile
<var> - variable to store the result
(1 for success, empty for failure)
```

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CheckCXXCompilerFlag

Check whether the CXX compiler supports a given flag.

CHECK_CXX_COMPILER_FLAG(<flag> <var>)

```
<flag> - the compiler flag
<var> - variable to store the result
```

This internally calls the check_cxx_source_compiles macro and sets CMAKE_REQUIRED_DEFINITIONS to <flag>. See help for CheckCXXSourceCompiles for a listing of variables that can otherwise modify the build. The result only tells that the compiler does not give an error message when it encounters the flag. If the flag has any effect or even a specific one is beyond the scope of this module.

CheckCXXSourceCompiles

Check if given C++ source compiles and links into an executable

CHECK_CXX_SOURCE_COMPILES(<code> <var> [FAIL_REGEX <fail-regex>])

```
<code> - source code to try to compile, must define 'main'
<var> - variable to store whether the source code compiled
<fail-regex> - fail if test output matches this regex
```

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CheckCXXSourceRuns

Check if the given C++ source code compiles and runs.

CHECK_CXX_SOURCE_RUNS(<code> <var>)

```
<code> - source code to try to compile
<var> - variable to store the result
(1 for success, empty for failure)
```

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CheckCXXSymbolExists

Check if a symbol exists as a function, variable, or macro in C++

CHECK_CXX_SYMBOL_EXISTS(<symbol> <files> <variable>)

Check that the <symbol> is available after including given header <files> and store the result in a <variable>. Specify the list of files in one argument as a semicolon-separated list. CHECK_CXX_SYMBOL_EXISTS() can be used to check in C++ files, as opposed to CHECK_SYMBOL_EXISTS(), which works only for C.

If the header files define the symbol as a macro it is considered available and assumed to work. If

the header files declare the symbol as a function or variable then the symbol must also be available for linking. If the symbol is a type or enum value it will not be recognized (consider using CheckTypeSize or CheckCSourceCompiles).

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CheckFortranFunctionExists

macro which checks if the Fortran function exists

CHECK_FORTRAN_FUNCTION_EXISTS(FUNCTION VARIABLE)

```
FUNCTION - the name of the Fortran function
VARIABLE - variable to store the result
```

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CheckFunctionExists

Check if a C function can be linked

CHECK_FUNCTION_EXISTS(<function> <variable>)

Check that the <function> is provided by libraries on the system and store the result in a <variable>. This does not verify that any system header file declares the function, only that it can be found at link time (consider using CheckSymbolExists).

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CheckIncludeFileCXX

Check if the include file exists.

```
CHECK_INCLUDE_FILE_CXX(INCLUDE VARIABLE)
```

```
INCLUDE - name of include file
VARIABLE - variable to return result
```

An optional third argument is the CFlags to add to the compile line or you can use CMAKE_REQUIRED_FLAGS.

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
```

CheckIncludeFile

macro which checks the include file exists.

CHECK_INCLUDE_FILE(INCLUDE VARIABLE)

```
INCLUDE - name of include file
VARIABLE - variable to return result
```

an optional third argument is the CFlags to add to the compile line or you can use CMAKE_REQUIRED_FLAGS

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
```

CheckIncludeFiles

Check if the files can be included

CHECK_INCLUDE_FILES(INCLUDE VARIABLE)

```
INCLUDE - list of files to include
VARIABLE - variable to return result
```

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
```

CheckLanguage

Check if a language can be enabled

Usage:

```
check_language(<lang>)
```

where <lang> is a language that may be passed to enable_language() such as Fortran. If CMAKE_<lang>_COMPILER is already defined the check does nothing. Otherwise it tries enabling the language in a test project. The result is cached in CMAKE_<lang>_COMPILER as the compiler that was found, or NOTFOUND if the language cannot be enabled.

Example:

```
check_language(Fortran)
if(CMAKE_Fortran_COMPILER)
enable_language(Fortran)
else()
message(STATUS "No Fortran support")
endif()
```

CheckLibraryExists

Check if the function exists.

CHECK_LIBRARY_EXISTS (LIBRARY FUNCTION LOCATION VARIABLE)

```
LIBRARY - the name of the library you are looking for
FUNCTION - the name of the function
LOCATION - location where the library should be found
VARIABLE - variable to store the result
```

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CheckPrototypeDefinition

Check if the protoype we expect is correct.

check_prototype_definition(FUNCTION PROTOTYPE RETURN HEADER VARIABLE)

```
FUNCTION - The name of the function (used to check if prototype exists)
PROTOTYPE- The prototype to check.
RETURN - The return value of the function.
```

```
HEADER - The header files required.
VARIABLE - The variable to store the result.
```

Example:

```
check_prototype_definition(getpwent_r
"struct passwd *getpwent_r(struct passwd *src, char *buf, int buflen)"
"NULL"
"unistd.h;pwd.h"
SOLARIS_GETPWENT_R)
```

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CheckStructHasMember

Check if the given struct or class has the specified member variable

```
CHECK_STRUCT_HAS_MEMBER(<struct> <member> <header> <variable>
[LANGUAGE <language>])
```

```
<struct> - the name of the struct or class you are interested in
<member> - the member which existence you want to check
<header> - the header(s) where the prototype should be declared
<variable> - variable to store the result
<language> - the compiler to use (C or CXX)
```

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

Example: CHECK_STRUCT_HAS_MEMBER(struct timeval tv_sec sys/select.h HAVE_TIMEVAL_TV_SEC LANGUAGE C)

CheckSymbolExists

Check if a symbol exists as a function, variable, or macro

CHECK_SYMBOL_EXISTS(<symbol> <files> <variable>)

Check that the <symbol> is available after including given header <files> and store the result in a <variable>. Specify the list of files in one argument as a semicolon-separated list.

If the header files define the symbol as a macro it is considered available and assumed to work. If the header files declare the symbol as a function or variable then the symbol must also be available for linking. If the symbol is a type or enum value it will not be recognized (consider using CheckTypeSize or CheckCSourceCompiles). If the check needs to be done in C++, consider using CHECK_CXX_SYMBOL_EXISTS(), which does the same as CHECK_SYMBOL_EXISTS(), but in C++.

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CheckTypeSize

Check sizeof a type

```
CHECK_TYPE_SIZE(TYPE VARIABLE [BUILTIN_TYPES_ONLY]
[LANGUAGE <language>])
```

Check if the type exists and determine its size. On return, HAVE_${VARIABLE} holds the existence of the type, and ${VARIABLE} holds one of the following:

```
<size> = type has non-zero size <size>
"0" = type has arch-dependent size (see below)
"" = type does not exist
```

Furthermore, the variable ${VARIABLE}_CODE holds C preprocessor code to define the macro ${VARIABLE} to the size of the type, or leave the macro undefined if the type does not exist.

The variable ${VARIABLE} may be 0 when CMAKE_OSX_ARCHITECTURES has multiple architectures for building OS X universal binaries. This indicates that the type size varies across architectures. In this case ${VARIABLE}_CODE contains C preprocessor tests mapping from each architecture macro to the corresponding type size. The list of architecture macros is stored in ${VARIABLE}_KEYS, and the value for each key is stored in ${VARIABLE}-${KEY}.

If the BUILTIN_TYPES_ONLY option is not given, the macro checks for headers <sys/types.h>, <stdint.h>, and <stddef.h>, and saves results in HAVE_SYS_TYPES_H, HAVE_STDINT_H, and HAVE_STDDEF_H. The type size check automatically includes the available headers, thus supporting checks of types defined in the headers.

If LANGUAGE is set, the specified compiler will be used to perform the check. Acceptable values are C and CXX

Despite the name of the macro you may use it to check the size of more complex expressions, too. To check e.g. for the size of a struct member you can do something like this:

```
check_type_size("((struct something*)0)->member" SIZEOF_MEMBER)
```

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_INCLUDES = list of include directories
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
CMAKE_EXTRA_INCLUDE_FILES = list of extra headers to include
```

CheckVariableExists

Check if the variable exists.

```
CHECK_VARIABLE_EXISTS(VAR VARIABLE)

VAR - the name of the variable
VARIABLE - variable to store the result
```

This macro is only for C variables.

The following variables may be set before calling this macro to modify the way the check is run:

```
CMAKE_REQUIRED_FLAGS = string of compile command line flags
CMAKE_REQUIRED_DEFINITIONS = list of macros to define (-DFOO=bar)
CMAKE_REQUIRED_LIBRARIES = list of libraries to link
```

CMakeAddFortranSubdirectory

Use MinGW gfortran from VS if a fortran compiler is not found.

The add_fortran_subdirectory function adds a subdirectory to a project that contains a fortran only sub-project. The module will check the current compiler and see if it can support fortran. If

no fortran compiler is found and the compiler is MSVC, then this module will find the MinGW gfortran. It will then use an external project to build with the MinGW tools. It will also create imported targets for the libraries created. This will only work if the fortran code is built into a dll, so BUILD_SHARED_LIBS is turned on in the project. In addition the CMAKE_GNUtoMS option is set to on, so that the MS .lib files are created. Usage is as follows:

```
cmake_add_fortran_subdirectory(
<subdir> # name of subdirectory
PROJECT <project_name> # project name in subdir top CMakeLists.txt
ARCHIVE_DIR <dir> # dir where project places .lib files
RUNTIME_DIR <dir> # dir where project places .dll files
LIBRARIES <lib>... # names of library targets to import
LINK_LIBRARIES # link interface libraries for LIBRARIES
[LINK_LIBS <lib> <dep>...]...
CMAKE_COMMAND_LINE ... # extra command line flags to pass to cmake
NO_EXTERNAL_INSTALL # skip installation of external project
)
```

Relative paths in ARCHIVE_DIR and RUNTIME_DIR are interpreted with respect to the build directory corresponding to the source directory in which the function is invoked.

Limitations:

NO_EXTERNAL_INSTALL is required for forward compatibility with a future version that supports installation of the external project binaries during make install.

CMakeBackwardCompatibilityCXX
define a bunch of backwards compatibility variables

```
CMAKE_ANSI_CXXFLAGS - flag for ansi c++
CMAKE_HAS_ANSI_STRING_STREAM - has <strstream>
include(TestForANSIStreamHeaders)
include(CheckIncludeFileCXX)
include(TestForSTDNamespace)
include(TestForANSIForScope)
```

CMakeDependentOption
Macro to provide an option dependent on other options.

This macro presents an option to the user only if a set of other conditions are true. When the option is not presented a default value is used, but any value set by the user is preserved for when the option is presented again. Example invocation:

```
CMAKE_DEPENDENT_OPTION(USE_FOO "Use Foo" ON
"USE_BAR;NOT USE_ZOT" OFF)
```

If USE_BAR is true and USE_ZOT is false, this provides an option called USE_FOO that defaults to ON. Otherwise, it sets USE_FOO to OFF. If the status of USE_BAR or USE_ZOT ever changes, any value for the USE_FOO option is saved so that when the option is re-enabled it retains its old value.

CMakeDetermineVSServicePack
Deprecated. Do not use.

The functionality of this module has been superseded by the **CMAKE_<LANG>_COMPILER_VERSION** variable that contains the compiler version number.

Determine the Visual Studio service pack of the cl in use.

Usage:

```
if(MSVC)
```

```
include(CMakeDetermineVSServicePack)
DetermineVSServicePack( my_service_pack )
if( my_service_pack )
message(STATUS "Detected: ${my_service_pack}")
endif()
endif()
```

Function DetermineVSServicePack sets the given variable to one of the following values or an empty string if unknown:

```
vc80, vc80sp1
vc90, vc90sp1
vc100, vc100sp1
vc110, vc110sp1, vc110sp2, vc110sp3, vc110sp4
```

CMakeExpandImportedTargets

```
CMAKE_EXPAND_IMPORTED_TARGETS(<var> LIBRARIES lib1 lib2...libN
[CONFIGURATION <config>])
```

CMAKE_EXPAND_IMPORTED_TARGETS() takes a list of libraries and replaces all imported targets contained in this list with their actual file paths of the referenced libraries on disk, including the libraries from their link interfaces. If a CONFIGURATION is given, it uses the respective configuration of the imported targets if it exists. If no CONFIGURATION is given, it uses the first configuration from ${CMAKE_CONFIGURATION_TYPES} if set, otherwise ${CMAKE_BUILD_TYPE}. This macro is used by all Check*.cmake files which use try_compile() or try_run() and support CMAKE_REQUIRED_LIBRARIES , so that these checks support imported targets in CMAKE_REQUIRED_LIBRARIES:

```
cmake_expand_imported_targets(expandedLibs LIBRARIES ${CMAKE_REQUIRED_LIBRARIES}
CONFIGURATION "${CMAKE_TRY_COMPILE_CONFIGURATION}" )
```

CMakeFindDependencyMacro

```
find_dependency(<dep> [<version> [EXACT]])
```

find_dependency() wraps a **find_package()** call for a package dependency. It is designed to be used in a <package>Config.cmake file, and it forwards the correct parameters for EXACT, QUIET and REQUIRED which were passed to the original **find_package()** call. It also sets an informative diagnostic message if the dependency could not be found.

CMakeFindFrameworks

helper module to find OSX frameworks

CMakeFindPackageMode

This file is executed by cmake when invoked with --find-package. It expects that the following variables are set using -D:

```
NAME = name of the package
COMPILER_ID = the CMake compiler ID for which the result is, i.e. GNU/Intel/Clang/MSVC, et
LANGUAGE = language for which the result will be used, i.e. C/CXX/Fortan/ASM
MODE = EXIST : only check for existence of the given package
COMPILE : print the flags needed for compiling an object file which uses the given package
LINK : print the flags needed for linking when using the given package
QUIET = if TRUE, don't print anything
```

CMakeForceCompiler

This module defines macros intended for use by cross-compiling toolchain files when CMake is not able to automatically detect the compiler identification.

Macro CMAKE_FORCE_C_COMPILER has the following signature:

```
CMAKE_FORCE_C_COMPILER(<compiler> <compiler-id>)
```

It sets CMAKE_C_COMPILER to the given compiler and the cmake internal variable CMAKE_C_COMPILER_ID to the given compiler-id. It also bypasses the check for working compiler and basic compiler information tests.

Macro CMAKE_FORCE_CXX_COMPILER has the following signature:

```
CMAKE_FORCE_CXX_COMPILER(<compiler> <compiler-id>)
```

It sets CMAKE_CXX_COMPILER to the given compiler and the cmake internal variable CMAKE_CXX_COMPILER_ID to the given compiler-id. It also bypasses the check for working compiler and basic compiler information tests.

Macro CMAKE_FORCE_Fortran_COMPILER has the following signature:

```
CMAKE_FORCE_Fortran_COMPILER(<compiler> <compiler-id>)
```

It sets CMAKE_Fortran_COMPILER to the given compiler and the cmake internal variable CMAKE_Fortran_COMPILER_ID to the given compiler-id. It also bypasses the check for working compiler and basic compiler information tests.

So a simple toolchain file could look like this:

```
include (CMakeForceCompiler)
set(CMAKE_SYSTEM_NAME Generic)
CMAKE_FORCE_C_COMPILER (chc12 MetrowerksHicross)
CMAKE_FORCE_CXX_COMPILER (chc12 MetrowerksHicross)
```

CMakeGraphVizOptions

The builtin graphviz support of CMake.

Variables specific to the graphviz support

CMake can generate graphviz files, showing the dependencies between the targets in a project and also external libraries which are linked against. When CMake is run with the --graphiz=foo option, it will produce

- a foo.dot file showing all dependencies in the project
- a foo.dot.<target> file for each target, file showing on which other targets the respective target depends
- a foo.dot.<target>.dependers file, showing which other targets depend on the respective target

This can result in huge graphs. Using the file CMakeGraphVizOptions.cmake the look and content of the generated graphs can be influenced. This file is searched first in ${CMAKE_BINARY_DIR} and then in ${CMAKE_SOURCE_DIR}. If found, it is read and the variables set in it are used to adjust options for the generated graphviz files.

GRAPHVIZ_GRAPH_TYPE

The graph type

- Mandatory : NO
- Default : digraph

GRAPHVIZ_GRAPH_NAME

The graph name.

- Mandatory : NO
- Default : GG

GRAPHVIZ_GRAPH_HEADER

The header written at the top of the graphviz file.

- Mandatory : NO

- Default : node [n fontsize = 12];

GRAPHVIZ_NODE_PREFIX
The prefix for each node in the graphviz file.

- Mandatory : NO
- Default : node

GRAPHVIZ_EXECUTABLES
Set this to FALSE to exclude executables from the generated graphs.

- Mandatory : NO
- Default : TRUE

GRAPHVIZ_STATIC_LIBS
Set this to FALSE to exclude static libraries from the generated graphs.

- Mandatory : NO
- Default : TRUE

GRAPHVIZ_SHARED_LIBS
Set this to FALSE to exclude shared libraries from the generated graphs.

- Mandatory : NO
- Default : TRUE

GRAPHVIZ_MODULE_LIBS
Set this to FALSE to exclude module libraries from the generated graphs.

- Mandatory : NO
- Default : TRUE

GRAPHVIZ_EXTERNAL_LIBS
Set this to FALSE to exclude external libraries from the generated graphs.

- Mandatory : NO
- Default : TRUE

GRAPHVIZ_IGNORE_TARGETS
A list of regular expressions for ignoring targets.

- Mandatory : NO
- Default : empty

GRAPHVIZ_GENERATE_PER_TARGET
Set this to FALSE to exclude per target graphs **foo.dot.<target>**.

- Mandatory : NO
- Default : TRUE

GRAPHVIZ_GENERATE_DEPENDERS
Set this to FALSE to exclude depender graphs **foo.dot.<target>.dependers**.

- Mandatory : NO
- Default : TRUE

CMakePackageConfigHelpers
CONFIGURE_PACKAGE_CONFIG_FILE(), WRITE_BASIC_PACKAGE_VERSION_FILE()

```
CONFIGURE_PACKAGE_CONFIG_FILE(<input> <output> INSTALL_DESTINATION <path>
[PATH_VARS <var1> <var2> ... <varN>]
[NO_SET_AND_CHECK_MACRO]
[NO_CHECK_REQUIRED_COMPONENTS_MACRO])
```

CONFIGURE_PACKAGE_CONFIG_FILE() should be used instead of the plain configure_file() command when creating the <Name>Config.cmake or <Name>-config.cmake file for installing a project or library. It helps making the resulting package relocatable by avoiding hardcoded paths in the installed Config.cmake file.

In a FooConfig.cmake file there may be code like this to make the install destinations know to the using project:

```
set(FOO_INCLUDE_DIR "@CMAKE_INSTALL_FULL_INCLUDEDIR@" )
set(FOO_DATA_DIR "@CMAKE_INSTALL_PREFIX@/@RELATIVE_DATA_INSTALL_DIR@" )
set(FOO_ICONS_DIR "@CMAKE_INSTALL_PREFIX@/share/icons" )
...logic to determine installedPrefix from the own location...
set(FOO_CONFIG_DIR "${installedPrefix}/@CONFIG_INSTALL_DIR@" )
```

All 4 options shown above are not sufficient, since the first 3 hardcode the absolute directory locations, and the 4th case works only if the logic to determine the installedPrefix is correct, and if CONFIG_INSTALL_DIR contains a relative path, which in general cannot be guaranteed. This has the effect that the resulting FooConfig.cmake file would work poorly under Windows and OSX, where users are used to choose the install location of a binary package at install time, independent from how CMAKE_INSTALL_PREFIX was set at build/cmake time.

Using CONFIGURE_PACKAGE_CONFIG_FILE() helps. If used correctly, it makes the resulting FooConfig.cmake file relocatable. Usage:

```
1. write a FooConfig.cmake.in file as you are used to
2. insert a line containing only the string "@PACKAGE_INIT@"
3. instead of set(FOO_DIR "@SOME_INSTALL_DIR@"), use set(FOO_DIR "@PACKAGE_SOME_INSTALL_DI
(this must be after the @PACKAGE_INIT@ line)
4. instead of using the normal configure_file(), use CONFIGURE_PACKAGE_CONFIG_FILE()
```

The <input> and <output> arguments are the input and output file, the same way as in configure_file().

The <path> given to INSTALL_DESTINATION must be the destination where the FooConfig.cmake file will be installed to. This can either be a relative or absolute path, both work.

The variables <var1> to <varN> given as PATH_VARS are the variables which contain install destinations. For each of them the macro will create a helper variable PACKAGE_<var...>. These helper variables must be used in the FooConfig.cmake.in file for setting the installed location. They are calculated by CONFIGURE_PACKAGE_CONFIG_FILE() so that they are always relative to the installed location of the package. This works both for relative and also for absolute locations. For absolute locations it works only if the absolute location is a subdirectory of CMAKE_INSTALL_PREFIX.

By default configure_package_config_file() also generates two helper macros, set_and_check() and check_required_components() into the FooConfig.cmake file.

set_and_check() should be used instead of the normal set() command for setting directories and file locations. Additionally to setting the variable it also checks that the referenced file or directory actually exists and fails with a FATAL_ERROR otherwise. This makes sure that the created FooConfig.cmake file does not contain wrong references. When using the NO_SET_AND_CHECK_MACRO, this macro is not generated into the FooConfig.cmake file.

check_required_components(<package_name>) should be called at the end of the FooConfig.cmake file if the package supports components. This macro checks whether all requested, non-optional components have been found, and if this is not the case, sets the Foo_FOUND variable to FALSE, so that the package is considered to be not found. It does that by testing the Foo_<Component>_FOUND variables for all requested required components. When using the NO_CHECK_REQUIRED_COMPONENTS option, this macro is not generated into the FooConfig.cmake file.

For an example see below the documentation for WRITE_BASIC_PACKAGE_VERSION_FILE().

```
WRITE_BASIC_PACKAGE_VERSION_FILE( filename [VERSION major.minor.patch] COMPATIBILITY (AnyN
```

Writes a file for use as <package>ConfigVersion.cmake file to <filename>. See the documentation of find_package() for details on this.

```
filename is the output filename, it should be in the build tree.
major.minor.patch is the version number of the project to be installed
```

If no **VERSION** is given, the **PROJECT_VERSION** variable is used. If this hasnt been set, it errors out.

The COMPATIBILITY mode AnyNewerVersion means that the installed package version will be considered compatible if it is newer or exactly the same as the requested version. This mode should be used for packages which are fully backward compatible, also across major versions. If SameMajorVersion is used instead, then the behaviour differs from AnyNewerVersion in that the major version number must be the same as requested, e.g. version 2.0 will not be considered compatible if 1.0 is requested. This mode should be used for packages which guarantee backward compatibility within the same major version. If ExactVersion is used, then the package is only considered compatible if the requested version matches exactly its own version number (not considering the tweak version). For example, version 1.2.3 of a package is only considered compatible to requested version 1.2.3. This mode is for packages without compatibility guarantees. If your project has more elaborated version matching rules, you will need to write your own custom ConfigVersion.cmake file instead of using this macro.

Internally, this macro executes configure_file() to create the resulting version file. Depending on the COMPATIBLITY, either the file BasicConfigVersion-SameMajorVersion.cmake.in or BasicConfigVersion-AnyNewerVersion.cmake.in is used. Please note that these two files are internal to CMake and you should not call configure_file() on them yourself, but they can be used as starting point to create more sophisticted custom ConfigVersion.cmake files.

Example using both configure_package_config_file() and write_basic_package_version_file(): CMakeLists.txt:

```
set(INCLUDE_INSTALL_DIR include/ ... CACHE )
set(LIB_INSTALL_DIR lib/ ... CACHE )
set(SYSCONFIG_INSTALL_DIR etc/foo/ ... CACHE )
...
include(CMakePackageConfigHelpers)
configure_package_config_file(FooConfig.cmake.in ${CMAKE_CURRENT_BINARY_DIR}/FooConfig.cma
INSTALL_DESTINATION ${LIB_INSTALL_DIR}/Foo/cmake
PATH_VARS INCLUDE_INSTALL_DIR SYSCONFIG_INSTALL_DIR)
write_basic_package_version_file(${CMAKE_CURRENT_BINARY_DIR}/FooConfigVersion.cmake
VERSION 1.2.3
COMPATIBILITY SameMajorVersion )
install(FILES ${CMAKE_CURRENT_BINARY_DIR}/FooConfig.cmake ${CMAKE_CURRENT_BINARY_DIR}/FooC
DESTINATION ${LIB_INSTALL_DIR}/Foo/cmake )
```

With a FooConfig.cmake.in:

```
set(FOO_VERSION x.y.z)
...
@PACKAGE_INIT@
...
set_and_check(FOO_INCLUDE_DIR "@PACKAGE_INCLUDE_INSTALL_DIR@")
set_and_check(FOO_SYSCONFIG_DIR "@PACKAGE_SYSCONFIG_INSTALL_DIR@")

check_required_components(Foo)
```

CMakeParseArguments

CMAKE_PARSE_ARGUMENTS(<prefix> <options> <one_value_keywords> <multi_value_keywords> args...)

CMAKE_PARSE_ARGUMENTS() is intended to be used in macros or functions for parsing the arguments given to that macro or function. It processes the arguments and defines a set of variables which hold the values of the respective options.

The <options> argument contains all options for the respective macro, i.e. keywords which can be used when calling the macro without any value following, like e.g. the OPTIONAL keyword of the install() command.

The <one_value_keywords> argument contains all keywords for this macro which are followed by one value, like e.g. DESTINATION keyword of the install() command.

The <multi_value_keywords> argument contains all keywords for this macro which can be followed by more than one value, like e.g. the TARGETS or FILES keywords of the install() command.

When done, CMAKE_PARSE_ARGUMENTS() will have defined for each of the keywords listed in <options>, <one_value_keywords> and <multi_value_keywords> a variable composed of the given <prefix> followed by _ and the name of the respective keyword. These variables will then hold the respective value from the argument list. For the <options> keywords this will be TRUE or FALSE.

All remaining arguments are collected in a variable <prefix>_UNPARSED_ARGUMENTS, this can be checked afterwards to see whether your macro was called with unrecognized parameters.

As an example here a my_install() macro, which takes similar arguments as the real install() command:

```
function(MY_INSTALL)
set(options OPTIONAL FAST)
set(oneValueArgs DESTINATION RENAME)
set(multiValueArgs TARGETS CONFIGURATIONS)
cmake_parse_arguments(MY_INSTALL "${options}" "${oneValueArgs}" "${multiValueArgs}" ${ARGN
```

Assume my_install() has been called like this:

```
my_install(TARGETS foo bar DESTINATION bin OPTIONAL blub)
```

After the cmake_parse_arguments() call the macro will have set the following variables:

```
MY_INSTALL_OPTIONAL = TRUE
MY_INSTALL_FAST = FALSE (this option was not used when calling my_install()
MY_INSTALL_DESTINATION = "bin"
MY_INSTALL_RENAME = "" (was not used)
MY_INSTALL_TARGETS = "foo;bar"
MY_INSTALL_CONFIGURATIONS = "" (was not used)
MY_INSTALL_UNPARSED_ARGUMENTS = "blub" (no value expected after "OPTIONAL"
```

You can then continue and process these variables.

Keywords terminate lists of values, e.g. if directly after a one_value_keyword another recognized keyword follows, this is interpreted as the beginning of the new option. E.g. my_install(TARGETS foo DESTINATION OPTIONAL) would result in MY_INSTALL_DESTINATION set to OPTIONAL, but MY_INSTALL_DESTINATION would be empty and MY_INSTALL_OPTIONAL would be set to TRUE therefor.

CMakePrintHelpers

Convenience macros for printing properties and variables, useful e.g. for debugging.

```
CMAKE_PRINT_PROPERTIES([TARGETS target1 .. targetN]
```

```
[SOURCES source1 .. sourceN]
[DIRECTORIES dir1 .. dirN]
[TESTS test1 .. testN]
[CACHE_ENTRIES entry1 .. entryN]
PROPERTIES prop1 .. propN )
```

This macro prints the values of the properties of the given targets, source files, directories, tests or cache entries. Exactly one of the scope keywords must be used. Example:

```
cmake_print_properties(TARGETS foo bar PROPERTIES LOCATION INTERFACE_INCLUDE_DIRS)
```

This will print the LOCATION and INTERFACE_INCLUDE_DIRS properties for both targets foo and bar.

CMAKE_PRINT_VARIABLES(var1 var2 .. varN)

This macro will print the name of each variable followed by its value. Example:

```
cmake_print_variables(CMAKE_C_COMPILER CMAKE_MAJOR_VERSION THIS_ONE_DOES_NOT_EXIST)
```

Gives:

```
-- CMAKE_C_COMPILER="/usr/bin/gcc" ; CMAKE_MAJOR_VERSION="2" ; THIS_ONE_DOES_NOT_EXIST=""
```

CMakePrintSystemInformation

print system information

This file can be used for diagnostic purposes just include it in a project to see various internal CMake variables.

CMakePushCheckState

This module defines three macros: CMAKE_PUSH_CHECK_STATE() CMAKE_POP_CHECK_STATE() and CMAKE_RESET_CHECK_STATE() These macros can be used to save, restore and reset (i.e., clear contents) the state of the variables CMAKE_REQUIRED_FLAGS, CMAKE_REQUIRED_DEFINITIONS, CMAKE_REQUIRED_LIBRARIES and CMAKE_REQUIRED_INCLUDES used by the various Check-files coming with CMake, like e.g. check_function_exists() etc. The variable contents are pushed on a stack, pushing multiple times is supported. This is useful e.g. when executing such tests in a Find-module, where they have to be set, but after the Find-module has been executed they should have the same value as they had before.

CMAKE_PUSH_CHECK_STATE() macro receives optional argument RESET. Whether its specified, CMAKE_PUSH_CHECK_STATE() will set all CMAKE_REQUIRED_* variables to empty values, same as CMAKE_RESET_CHECK_STATE() call will do.

Usage:

```
cmake_push_check_state(RESET)
set(CMAKE_REQUIRED_DEFINITIONS -DSOME_MORE_DEF)
check_function_exists(...)
cmake_reset_check_state()
set(CMAKE_REQUIRED_DEFINITIONS -DANOTHER_DEF)
check_function_exists(...)
cmake_pop_check_state()
```

CMakeVerifyManifest

CMakeVerifyManifest.cmake

This script is used to verify that embeded manifests and side by side manifests for a project match. To run this script, cd to a directory and run the script with cmake -P. On the command line you can pass in versions that are OK even if not found in the .manifest files. For example, cmake -Dallow_versions=8.0.50608.0 -PCmakeVerifyManifest.cmake could be used to allow an embeded manifest of 8.0.50608.0 to be used in a project even if that version was not found in the

.manifest file.

CPackBundle
CPack Bundle generator (Mac OS X) specific options

Variables specific to CPack Bundle generator
Installers built on Mac OS X using the Bundle generator use the aforementioned DragNDrop (CPACK_DMG_xxx) variables, plus the following Bundle-specific parameters (CPACK_BUNDLE_xxx).

CPACK_BUNDLE_NAME
The name of the generated bundle. This appears in the OSX finder as the bundle name. Required.

CPACK_BUNDLE_PLIST
Path to an OSX plist file that will be used for the generated bundle. This assumes that the caller has generated or specified their own Info.plist file. Required.

CPACK_BUNDLE_ICON
Path to an OSX icon file that will be used as the icon for the generated bundle. This is the icon that appears in the OSX finder for the bundle, and in the OSX dock when the bundle is opened. Required.

CPACK_BUNDLE_STARTUP_COMMAND
Path to a startup script. This is a path to an executable or script that will be run whenever an end-user double-clicks the generated bundle in the OSX Finder. Optional.

CPackComponent
Build binary and source package installers

Variables concerning CPack Components
The CPackComponent module is the module which handles the component part of CPack. See CPack module for general information about CPack.

For certain kinds of binary installers (including the graphical installers on Mac OS X and Windows), CPack generates installers that allow users to select individual application components to install. The contents of each of the components are identified by the COMPONENT argument of CMakes INSTALL command. These components can be annotated with user-friendly names and descriptions, inter-component dependencies, etc., and grouped in various ways to customize the resulting installer. See the cpack_add_* commands, described below, for more information about component-specific installations.

Component-specific installation allows users to select specific sets of components to install during the install process. Installation components are identified by the COMPONENT argument of CMakes INSTALL commands, and should be further described by the following CPack commands:

CPACK_COMPONENTS_ALL
The list of component to install.

The default value of this variable is computed by CPack and contains all components defined by the project. The user may set it to only include the specified components.

CPACK_<GENNAME>_COMPONENT_INSTALL
Enable/Disable component install for CPack generator <GENNAME>.

Each CPack Generator (RPM, DEB, ARCHIVE, NSIS, DMG, etc...) has a legacy default behavior. e.g. RPM builds monolithic whereas NSIS builds component. One can change the default behavior by setting this variable to 0/1 or OFF/ON.

CPACK_COMPONENTS_GROUPING
Specify how components are grouped for multi-package component-aware CPack generators.

Some generators like RPM or ARCHIVE family (TGZ, ZIP, ...) generates several packages files when asked for component packaging. They group the component differently depending on the value of this variable:

- ONE_PER_GROUP (default): creates one package file per component group
- ALL_COMPONENTS_IN_ONE : creates a single package with all (requested) component
- IGNORE : creates one package per component, i.e. IGNORE component group

One can specify different grouping for different CPack generator by using a CPACK_PROJECT_CONFIG_FILE.

CPACK_COMPONENT_<compName>_DISPLAY_NAME
The name to be displayed for a component.

CPACK_COMPONENT_<compName>_DESCRIPTION
The description of a component.

CPACK_COMPONENT_<compName>_GROUP
The group of a component.

CPACK_COMPONENT_<compName>_DEPENDS
The dependencies (list of components) on which this component depends.

CPACK_COMPONENT_<compName>_REQUIRED
True is this component is required.

cpack_add_component

Describes a CPack installation component named by the COMPONENT argument to a CMake INSTALL command.

```
cpack_add_component(compname
[DISPLAY_NAME name]
[DESCRIPTION description]
[HIDDEN | REQUIRED | DISABLED ]
[GROUP group]
[DEPENDS comp1 comp2 ... ]
[INSTALL_TYPES type1 type2 ... ]
[DOWNLOADED]
[ARCHIVE_FILE filename])
```

The cmake_add_component command describes an installation component, which the user can opt to install or remove as part of the graphical installation process. compname is the name of the component, as provided to the COMPONENT argument of one or more CMake INSTALL commands.

DISPLAY_NAME is the displayed name of the component, used in graphical installers to display the component name. This value can be any string.

DESCRIPTION is an extended description of the component, used in graphical installers to give the user additional information about the component. Descriptions can span multiple lines using n as the line separator. Typically, these descriptions should be no more than a few lines long.

HIDDEN indicates that this component will be hidden in the graphical installer, so that the user cannot directly change whether it is installed or not.

REQUIRED indicates that this component is required, and therefore will always be installed. It will be visible in the graphical installer, but it cannot be unselected. (Typically, required components are shown greyed out).

DISABLED indicates that this component should be disabled (unselected) by default. The user is

free to select this component for installation, unless it is also HIDDEN.

DEPENDS lists the components on which this component depends. If this component is selected, then each of the components listed must also be selected. The dependency information is encoded within the installer itself, so that users cannot install inconsistent sets of components.

GROUP names the component group of which this component is a part. If not provided, the component will be a standalone component, not part of any component group. Component groups are described with the cpack_add_component_group command, detailed below.

INSTALL_TYPES lists the installation types of which this component is a part. When one of these installations types is selected, this component will automatically be selected. Installation types are described with the cpack_add_install_type command, detailed below.

DOWNLOADED indicates that this component should be downloaded on-the-fly by the installer, rather than packaged in with the installer itself. For more information, see the cpack_configure_downloads command.

ARCHIVE_FILE provides a name for the archive file created by CPack to be used for downloaded components. If not supplied, CPack will create a file with some name based on CPACK_PACKAGE_FILE_NAME and the name of the component. See cpack_configure_downloads for more information.

cpack_add_component_group

Describes a group of related CPack installation components.

```
cpack_add_component_group(groupname
[DISPLAY_NAME name]
[DESCRIPTION description]
[PARENT_GROUP parent]
[EXPANDED]
[BOLD_TITLE])
```

The cpack_add_component_group describes a group of installation components, which will be placed together within the listing of options. Typically, component groups allow the user to select/deselect all of the components within a single group via a single group-level option. Use component groups to reduce the complexity of installers with many options. groupname is an arbitrary name used to identify the group in the GROUP argument of the cpack_add_component command, which is used to place a component in a group. The name of the group must not conflict with the name of any component.

DISPLAY_NAME is the displayed name of the component group, used in graphical installers to display the component group name. This value can be any string.

DESCRIPTION is an extended description of the component group, used in graphical installers to give the user additional information about the components within that group. Descriptions can span multiple lines using n as the line separator. Typically, these descriptions should be no more than a few lines long.

PARENT_GROUP, if supplied, names the parent group of this group. Parent groups are used to establish a hierarchy of groups, providing an arbitrary hierarchy of groups.

EXPANDED indicates that, by default, the group should show up as expanded, so that the user immediately sees all of the components within the group. Otherwise, the group will initially show up as a single entry.

BOLD_TITLE indicates that the group title should appear in bold, to call the users attention to the group.

cpack_add_install_type

Add a new installation type containing a set of predefined component selections to the graphical installer.

```
cpack_add_install_type(typename
[DISPLAY_NAME name])
```

The cpack_add_install_type command identifies a set of preselected components that represents a common use case for an application. For example, a Developer install type might include an application along with its header and library files, while an End user install type might just include the applications executable. Each component identifies itself with one or more install types via the INSTALL_TYPES argument to cpack_add_component.

DISPLAY_NAME is the displayed name of the install type, which will typically show up in a drop-down box within a graphical installer. This value can be any string.

cpack_configure_downloads

Configure CPack to download selected components on-the-fly as part of the installation process.

```
cpack_configure_downloads(site
[UPLOAD_DIRECTORY dirname]
[ALL]
[ADD_REMOVE|NO_ADD_REMOVE])
```

The cpack_configure_downloads command configures installation-time downloads of selected components. For each downloadable component, CPack will create an archive containing the contents of that component, which should be uploaded to the given site. When the user selects that component for installation, the installer will download and extract the component in place. This feature is useful for creating small installers that only download the requested components, saving bandwidth. Additionally, the installers are small enough that they will be installed as part of the normal installation process, and the Change button in Windows Add/Remove Programs control panel will allow one to add or remove parts of the application after the original installation. On Windows, the downloaded-components functionality requires the ZipDLL plug-in for NSIS, available at:

http://nsis.sourceforge.net/ZipDLL_plug-in

On Mac OS X, installers that download components on-the-fly can only be built and installed on system using Mac OS X 10.5 or later.

The site argument is a URL where the archives for downloadable components will reside, e.g., *http://www.cmake.org/files/2.6.1/installer/* All of the archives produced by CPack should be uploaded to that location.

UPLOAD_DIRECTORY is the local directory where CPack will create the various archives for each of the components. The contents of this directory should be uploaded to a location accessible by the URL given in the site argument. If omitted, CPack will use the directory CPackUploads inside the CMake binary directory to store the generated archives.

The ALL flag indicates that all components be downloaded. Otherwise, only those components explicitly marked as DOWNLOADED or that have a specified ARCHIVE_FILE will be downloaded. Additionally, the ALL option implies ADD_REMOVE (unless NO_ADD_REMOVE is specified).

ADD_REMOVE indicates that CPack should install a copy of the installer that can be called from Windows Add/Remove Programs dialog (via the Modify button) to change the set of installed components. NO_ADD_REMOVE turns off this behavior. This option is ignored on Mac OS X.

CPackCygwin
Cygwin CPack generator (Cygwin).

Variables specific to CPack Cygwin generator
The following variable is specific to installers build on and/or for Cygwin:

CPACK_CYGWIN_PATCH_NUMBER
The Cygwin patch number. FIXME: This documentation is incomplete.

CPACK_CYGWIN_PATCH_FILE
The Cygwin patch file. FIXME: This documentation is incomplete.

CPACK_CYGWIN_BUILD_SCRIPT
The Cygwin build script. FIXME: This documentation is incomplete.

CPackDeb
The builtin (binary) CPack Deb generator (Unix only)

Variables specific to CPack Debian (DEB) generator
CPackDeb may be used to create Deb package using CPack. CPackDeb is a CPack generator thus it uses the CPACK_XXX variables used by CPack : *http://www.cmake.org/Wiki/CMake:CPackConfiguration*. CPackDeb generator should work on any linux host but it will produce better deb package when Debian specific tools dpkg-xxx are usable on the build system.

CPackDeb has specific features which are controlled by the specifics CPACK_DEBIAN_XXX variables.Youll find a detailed usage on the wiki: *http://www.cmake.org/Wiki/CMake:CPackPackageGenerators#DEB_.28UNIX_only.29*

However as a handy reminder here comes the list of specific variables:

CPACK_DEBIAN_PACKAGE_NAME

- Mandatory : YES
- Default : CPACK_PACKAGE_NAME (lower case)

The debian package summary

CPACK_DEBIAN_PACKAGE_VERSION

- Mandatory : YES
- Default : CPACK_PACKAGE_VERSION

The debian package version

CPACK_DEBIAN_PACKAGE_ARCHITECTURE

- Mandatory : YES
- Default : Output of dpkg --print-architecture (or i386 if dpkg is not found)

The debian package architecture

CPACK_DEBIAN_PACKAGE_DEPENDS

- Mandatory : NO
- Default : -

May be used to set deb dependencies.

CPACK_DEBIAN_PACKAGE_MAINTAINER

- Mandatory : YES
- Default : CPACK_PACKAGE_CONTACT

The debian package maintainer

CPACK_DEBIAN_PACKAGE_DESCRIPTION

- Mandatory : YES
- Default : CPACK_PACKAGE_DESCRIPTION_SUMMARY

The debian package description

CPACK_DEBIAN_PACKAGE_SECTION

- Mandatory : YES
- Default : devel

The debian package section

CPACK_DEBIAN_PACKAGE_PRIORITY

- Mandatory : YES
- Default : optional

The debian package priority

CPACK_DEBIAN_PACKAGE_HOMEPAGE

- Mandatory : NO
- Default : -

The URL of the web site for this package, preferably (when applicable) the site from which the original source can be obtained and any additional upstream documentation or information may be found. The content of this field is a simple URL without any surrounding characters such as <>.

CPACK_DEBIAN_PACKAGE_SHLIBDEPS

- Mandatory : NO
- Default : OFF

May be set to ON in order to use dpkg-shlibdeps to generate better package dependency list. You may need set CMAKE_INSTALL_RPATH toi appropriate value if you use this feature, because if you dont dpkg-shlibdeps may fail to find your own shared libs. See *http://www.cmake.org/Wiki/CMake_RPATH_handling*.

CPACK_DEBIAN_PACKAGE_DEBUG

- Mandatory : NO
- Default : -

May be set when invoking cpack in order to trace debug information during CPackDeb run.

CPACK_DEBIAN_PACKAGE_PREDEPENDS

- Mandatory : NO
- Default : -

see *http://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps* This field is like Depends, except that it also forces dpkg to complete installation of the packages named before even starting the installation of the package which declares the pre-dependency.

CPACK_DEBIAN_PACKAGE_ENHANCES

- Mandatory : NO
- Default : -

see *http://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps* This field is similar to Suggests but works in the opposite direction. It is used to declare that a package can enhance the functionality of another package.

CPACK_DEBIAN_PACKAGE_BREAKS

- Mandatory : NO
- Default : -

see *http://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps* When one binary package declares that it breaks another, dpkg will refuse to allow the package which declares Breaks be installed unless the broken package is deconfigured first, and it will refuse to allow the broken package to be reconfigured.

CPACK_DEBIAN_PACKAGE_CONFLICTS

- Mandatory : NO
- Default : -

see *http://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps* When one binary package declares a conflict with another using a Conflicts field, dpkg will refuse to allow them to be installed on the system at the same time.

CPACK_DEBIAN_PACKAGE_PROVIDES

- Mandatory : NO
- Default : -

see *http://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps* A virtual package is one which appears in the Provides control field of another package.

CPACK_DEBIAN_PACKAGE_REPLACES

- Mandatory : NO
- Default : -

see *http://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps* Packages can declare in their control file that they should overwrite files in certain other packages, or completely replace other packages.

CPACK_DEBIAN_PACKAGE_RECOMMENDS

- Mandatory : NO
- Default : -

see *http://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps* Allows packages to declare a strong, but not absolute, dependency on other packages.

CPACK_DEBIAN_PACKAGE_SUGGESTS

- Mandatory : NO
- Default : -

see *http://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps* Allows packages to declare a suggested package install grouping.

CPACK_DEBIAN_PACKAGE_CONTROL_EXTRA

- Mandatory : NO
- Default : -

This variable allow advanced user to add custom script to the control.tar.gz Typical usage is for conffiles, postinst, postrm, prerm. Usage:

```
set(CPACK_DEBIAN_PACKAGE_CONTROL_EXTRA
"${CMAKE_CURRENT_SOURCE_DIR/prerm;${CMAKE_CURRENT_SOURCE_DIR}/postrm")
```

CPackDMG

DragNDrop CPack generator (Mac OS X).

Variables specific to CPack DragNDrop generator

The following variables are specific to the DragNDrop installers built on Mac OS X:

CPACK_DMG_VOLUME_NAME

The volume name of the generated disk image. Defaults to CPACK_PACKAGE_FILE_NAME.

CPACK_DMG_FORMAT

The disk image format. Common values are UDRO (UDIF read-only), UDZO (UDIF zlib-compressed) or UDBZ (UDIF bzip2-compressed). Refer to hdiutil(1) for more information on other available formats.

CPACK_DMG_DS_STORE

Path to a custom DS_Store file. This .DS_Store file e.g. can be used to specify the Finder window position/geometry and layout (such as hidden toolbars, placement of the icons etc.). This file has to be generated by the Finder (either manually or through OSA-script) using a normal folder from which the .DS_Store file can then be extracted.

CPACK_DMG_BACKGROUND_IMAGE

Path to a background image file. This file will be used as the background for the Finder Window when the disk image is opened. By default no background image is set. The background image is applied after applying the custom .DS_Store file.

CPACK_COMMAND_HDIUTIL

Path to the hdiutil(1) command used to operate on disk image files on Mac OS X. This variable can be used to override the automatically detected command (or specify its location if the auto-detection fails to find it.)

CPACK_COMMAND_SETFILE

Path to the SetFile(1) command used to set extended attributes on files and directories on Mac OS X. This variable can be used to override the automatically detected command (or specify its location if the auto-detection fails to find it.)

CPACK_COMMAND_REZ

Path to the Rez(1) command used to compile resources on Mac OS X. This variable can be used to override the automatically detected command (or specify its location if the auto-detection fails to find it.)

CPackNSIS

CPack NSIS generator specific options

Variables specific to CPack NSIS generator

The following variables are specific to the graphical installers built on Windows using the Nullsoft Installation System.

CPACK_NSIS_INSTALL_ROOT

The default installation directory presented to the end user by the NSIS installer is under this root dir. The full directory presented to the end user is: ${CPACK_NSIS_INSTALL_ROOT}/${CPACK_PACKAGE_INSTALL_DIRECTORY}

CPACK_NSIS_MUI_ICON

An icon filename. The name of a ***.ico** file used as the main icon for the generated install program.

CPACK_NSIS_MUI_UNIICON

An icon filename. The name of a ***.ico** file used as the main icon for the generated uninstall program.

CPACK_NSIS_INSTALLER_MUI_ICON_CODE
undocumented.

CPACK_NSIS_EXTRA_PREINSTALL_COMMANDS
Extra NSIS commands that will be added to the beginning of the install Section, before your install tree is available on the target system.

CPACK_NSIS_EXTRA_INSTALL_COMMANDS
Extra NSIS commands that will be added to the end of the install Section, after your install tree is available on the target system.

CPACK_NSIS_EXTRA_UNINSTALL_COMMANDS
Extra NSIS commands that will be added to the uninstall Section, before your install tree is removed from the target system.

CPACK_NSIS_COMPRESSOR
The arguments that will be passed to the NSIS SetCompressor command.

CPACK_NSIS_ENABLE_UNINSTALL_BEFORE_INSTALL
Ask about uninstalling previous versions first. If this is set to ON, then an installer will look for previous installed versions and if one is found, ask the user whether to uninstall it before proceeding with the install.

CPACK_NSIS_MODIFY_PATH
Modify PATH toggle. If this is set to ON, then an extra page will appear in the installer that will allow the user to choose whether the program directory should be added to the system PATH variable.

CPACK_NSIS_DISPLAY_NAME
The display name string that appears in the Windows Add/Remove Program control panel

CPACK_NSIS_PACKAGE_NAME
The title displayed at the top of the installer.

CPACK_NSIS_INSTALLED_ICON_NAME
A path to the executable that contains the installer icon.

CPACK_NSIS_HELP_LINK
URL to a web site providing assistance in installing your application.

CPACK_NSIS_URL_INFO_ABOUT
URL to a web site providing more information about your application.

CPACK_NSIS_CONTACT
Contact information for questions and comments about the installation process.

CPACK_NSIS_CREATE_ICONS_EXTRA
Additional NSIS commands for creating start menu shortcuts.

CPACK_NSIS_DELETE_ICONS_EXTRA
Additional NSIS commands to uninstall start menu shortcuts.

CPACK_NSIS_EXECUTABLES_DIRECTORY
Creating NSIS start menu links assumes that they are in bin unless this variable is set. For example, you would set this to exec if your executables are in an exec directory.

CPACK_NSIS_MUI_FINISHPAGE_RUN
Specify an executable to add an option to run on the finish page of the NSIS installer.

CPACK_NSIS_MENU_LINKS
Specify links in [application] menu. This should contain a list of pair link link name. The link may be an URL or a path relative to installation prefix. Like:

```
set(CPACK_NSIS_MENU_LINKS
```

```
"doc/cmake-@CMake_VERSION_MAJOR@.@CMake_VERSION_MINOR@/cmake.html" "CMake Help"
    http://www.cmake.org -P "
"CMake Web Site")
```

CPackPackageMaker
PackageMaker CPack generator (Mac OS X).

Variables specific to CPack PackageMaker generator
The following variable is specific to installers built on Mac OS X using PackageMaker:

CPACK_OSX_PACKAGE_VERSION
The version of Mac OS X that the resulting PackageMaker archive should be compatible with. Different versions of Mac OS X support different features. For example, CPack can only build component-based installers for Mac OS X 10.4 or newer, and can only build installers that download component son-the-fly for Mac OS X 10.5 or newer. If left blank, this value will be set to the minimum version of Mac OS X that supports the requested features. Set this variable to some value (e.g., 10.4) only if you want to guarantee that your installer will work on that version of Mac OS X, and dont mind missing extra features available in the installer shipping with later versions of Mac OS X.

CPackRPM
The builtin (binary) CPack RPM generator (Unix only)

Variables specific to CPack RPM generator
CPackRPM may be used to create RPM package using CPack. CPackRPM is a CPack generator thus it uses the CPACK_XXX variables used by CPack : *http://www.cmake.org/Wiki/CMake:CPackConfiguration*

However CPackRPM has specific features which are controlled by the specifics CPACK_RPM_XXX variables. CPackRPM is a component aware generator so when CPACK_RPM_COMPONENT_INSTALL is ON some more CPACK_RPM_<Component-Name>_XXXX variables may be used in order to have component specific values. Note however that <componentName> refers to the **grouping name**. This may be either a component name or a component GROUP name. Usually those vars correspond to RPM spec file entities, one may find information about spec files here *http://www.rpm.org/wiki/Docs*.
Youll find a detailed usage of CPackRPM on the wiki:

http://www.cmake.org/Wiki/CMake:CPackPackageGenerators#RPM_.28Unix_Only.29

However as a handy reminder here comes the list of specific variables:

CPACK_RPM_PACKAGE_SUMMARY
The RPM package summary.

- Mandatory : YES
- Default : CPACK_PACKAGE_DESCRIPTION_SUMMARY

CPACK_RPM_PACKAGE_NAME
The RPM package name.

- Mandatory : YES
- Default : CPACK_PACKAGE_NAME

CPACK_RPM_PACKAGE_VERSION
The RPM package version.

- Mandatory : YES
- Default : CPACK_PACKAGE_VERSION

CPACK_RPM_PACKAGE_ARCHITECTURE
The RPM package architecture.

- Mandatory : NO
- Default : -

This may be set to noarch if you know you are building a noarch package.

CPACK_RPM_PACKAGE_RELEASE
The RPM package release.

- Mandatory : YES
- Default : 1

This is the numbering of the RPM package itself, i.e. the version of the packaging and not the version of the content (see CPACK_RPM_PACKAGE_VERSION). One may change the default value if the previous packaging was buggy and/or you want to put here a fancy Linux distro specific numbering.

CPACK_RPM_PACKAGE_LICENSE
The RPM package license policy.

- Mandatory : YES
- Default : unknown

CPACK_RPM_PACKAGE_GROUP
The RPM package group.

- Mandatory : YES
- Default : unknown

CPACK_RPM_PACKAGE_VENDOR
The RPM package vendor.

- Mandatory : YES
- Default : CPACK_PACKAGE_VENDOR if set or unknown

CPACK_RPM_PACKAGE_URL
The projects URL.

- Mandatory : NO
- Default : -

CPACK_RPM_PACKAGE_DESCRIPTION
RPM package description.

- Mandatory : YES
- Default : CPACK_PACKAGE_DESCRIPTION_FILE if set or no package description available

CPACK_RPM_COMPRESSION_TYPE
RPM compression type.

- Mandatory : NO
- Default : -

May be used to override RPM compression type to be used to build the RPM. For example some Linux distribution now default to lzma or xz compression whereas older cannot use such RPM. Using this one can enforce compression type to be used. Possible value are: lzma, xz, bzip2 and gzip.

CPACK_RPM_PACKAGE_REQUIRES
RPM spec requires field.

- Mandatory : NO
- Default : -

May be used to set RPM dependencies (requires). Note that you must enclose the complete requires string between quotes, for example:

```
set(CPACK_RPM_PACKAGE_REQUIRES "python >= 2.5.0, cmake >= 2.8")
```

The required package list of an RPM file could be printed with:

```
rpm -qp --requires file.rpm
```

CPACK_RPM_PACKAGE_SUGGESTS

RPM spec suggest field.

- Mandatory : NO
- Default : -

May be used to set weak RPM dependencies (suggests). Note that you must enclose the complete requires string between quotes.

CPACK_RPM_PACKAGE_PROVIDES

RPM spec provides field.

- Mandatory : NO
- Default : -

May be used to set RPM dependencies (provides). The provided package list of an RPM file could be printed with:

```
rpm -qp --provides file.rpm
```

CPACK_RPM_PACKAGE_OBSOLETES

RPM spec obsoletes field.

- Mandatory : NO
- Default : -

May be used to set RPM packages that are obsoleted by this one.

CPACK_RPM_PACKAGE_RELOCATABLE

build a relocatable RPM.

- Mandatory : NO
- Default : CPACK_PACKAGE_RELOCATABLE

If this variable is set to TRUE or ON CPackRPM will try to build a relocatable RPM package. A relocatable RPM may be installed using:

```
rpm --prefix or --relocate
```

in order to install it at an alternate place see rpm(8) Note that currently this may fail if CPACK_SET_DESTDIR is set to ON. If CPACK_SET_DESTDIR is set then you will get a warning message but if there is file installed with absolute path youll get unexpected behavior.

CPACK_RPM_SPEC_INSTALL_POST

- Mandatory : NO
- Default : -
- Deprecated: YES

This way of specifying post-install script is deprecated, use CPACK_RPM_POST_INSTALL_SCRIPT_FILE. May be used to set an RPM post-

install command inside the spec file. For example setting it to /bin/true may be used to prevent rpmbuild to strip binaries.

CPACK_RPM_SPEC_MORE_DEFINE
RPM extended spec definitions lines.

- Mandatory : NO
- Default : -

May be used to add any %define lines to the generated spec file.

CPACK_RPM_PACKAGE_DEBUG
Toggle CPackRPM debug output.

- Mandatory : NO
- Default : -

May be set when invoking cpack in order to trace debug information during CPack RPM run. For example you may launch CPack like this:

```
cpack -D CPACK_RPM_PACKAGE_DEBUG=1 -G RPM
```

CPACK_RPM_USER_BINARY_SPECFILE
A user provided spec file.

- Mandatory : NO
- Default : -

May be set by the user in order to specify a USER binary spec file to be used by CPackRPM instead of generating the file. The specified file will be processed by configure_file(@ONLY). One can provide a component specific file by setting CPACK_RPM_<componentName>_USER_BINARY_SPECFILE.

CPACK_RPM_GENERATE_USER_BINARY_SPECFILE_TEMPLATE
Spec file template.

- Mandatory : NO
- Default : -

If set CPack will generate a template for USER specified binary spec file and stop with an error. For example launch CPack like this:

```
cpack -D CPACK_RPM_GENERATE_USER_BINARY_SPECFILE_TEMPLATE=1 -G RPM
```

The user may then use this file in order to hand-craft is own binary spec file which may be used with CPACK_RPM_USER_BINARY_SPECFILE.

CPACK_RPM_PRE_INSTALL_SCRIPT_FILE

CPACK_RPM_PRE_UNINSTALL_SCRIPT_FILE

- Mandatory : NO
- Default : -

May be used to embed a pre (un)installation script in the spec file. The refered script file(s) will be read and directly put after the %pre or %preun section If CPACK_RPM_COMPONENT_INSTALL is set to ON the (un)install script for each component can be overridden with CPACK_RPM_<COMPONENT>_PRE_INSTALL_SCRIPT_FILE and CPACK_RPM_<COMPONENT>_PRE_UNINSTALL_SCRIPT_FILE. One may verify which scriptlet has been included with:

```
rpm -qp --scripts package.rpm
```

CPACK_RPM_POST_INSTALL_SCRIPT_FILE

CPACK_RPM_POST_UNINSTALL_SCRIPT_FILE

- Mandatory : NO
- Default : -

May be used to embed a post (un)installation script in the spec file. The refered script file(s) will be read and directly put after the %post or %postun section. If CPACK_RPM_COMPONENT_INSTALL is set to ON the (un)install script for each component can be overridden with CPACK_RPM_<COMPONENT>_POST_INSTALL_SCRIPT_FILE and CPACK_RPM_<COMPONENT>_POST_UNINSTALL_SCRIPT_FILE. One may verify which scriptlet has been included with:

```
rpm -qp --scripts package.rpm
```

CPACK_RPM_USER_FILELIST

CPACK_RPM_<COMPONENT>_USER_FILELIST

- Mandatory : NO
- Default : -

May be used to explicitly specify %(<directive>) file line in the spec file. Like %config(noreplace) or any other directive that be found in the %files section. Since CPackRPM is generating the list of files (and directories) the user specified files of the CPACK_RPM_<COMPONENT>_USER_FILELIST list will be removed from the generated list.

CPACK_RPM_CHANGELOG_FILE

RPM changelog file.

- Mandatory : NO
- Default : -

May be used to embed a changelog in the spec file. The refered file will be read and directly put after the %changelog section.

CPACK_RPM_EXCLUDE_FROM_AUTO_FILELIST

list of path to be excluded.

- Mandatory : NO
- Default : /etc /etc/init.d /usr /usr/share /usr/share/doc /usr/bin /usr/lib /usr/lib64 /usr/include

May be used to exclude path (directories or files) from the auto-generated list of paths discovered by CPack RPM. The defaut value contains a reasonable set of values if the variable is not defined by the user. If the variable is defined by the user then CPackRPM will NOT any of the default path. If you want to add some path to the default list then you can use CPACK_RPM_EXCLUDE_FROM_AUTO_FILELIST_ADDITION variable.

CPACK_RPM_EXCLUDE_FROM_AUTO_FILELIST_ADDITION

additional list of path to be excluded.

- Mandatory : NO
- Default : -

May be used to add more exclude path (directories or files) from the initial default list of excluded paths. See CPACK_RPM_EXCLUDE_FROM_AUTO_FILELIST.

CPack

Build binary and source package installers.

Variables common to all CPack generators

The CPack module generates binary and source installers in a variety of formats using the cpack program. Inclusion of the CPack module adds two new targets to the resulting makefiles, package and package_source, which build the binary and source installers, respectively. The generated binary installers contain everything installed via CMakes INSTALL command (and the deprecated INSTALL_FILES, INSTALL_PROGRAMS, and INSTALL_TARGETS commands).

For certain kinds of binary installers (including the graphical installers on Mac OS X and Windows), CPack generates installers that allow users to select individual application components to install. See CPackComponent module for that.

The CPACK_GENERATOR variable has different meanings in different contexts. In your CMakeLists.txt file, CPACK_GENERATOR is a *list of generators*: when run with no other arguments, CPack will iterate over that list and produce one package for each generator. In a CPACK_PROJECT_CONFIG_FILE, though, CPACK_GENERATOR is a *string naming a single generator*. If you need per-cpack- generator logic to control *other* cpack settings, then you need a CPACK_PROJECT_CONFIG_FILE.

The CMake source tree itself contains a CPACK_PROJECT_CONFIG_FILE. See the top level file CMakeCPackOptions.cmake.in for an example.

If set, the CPACK_PROJECT_CONFIG_FILE is included automatically on a per-generator basis. It only need contain overrides.

Heres how it works:

- cpack runs
- it includes CPackConfig.cmake
- it iterates over the generators listed in that files CPACK_GENERATOR list variable (unless told to use just a specific one via -G on the command line...)
- foreach generator, it then
  - sets CPACK_GENERATOR to the one currently being iterated
  - includes the CPACK_PROJECT_CONFIG_FILE
  - produces the package for that generator

This is the key: For each generator listed in CPACK_GENERATOR in CPackConfig.cmake, cpack will *reset* CPACK_GENERATOR internally to *the one currently being used* and then include the CPACK_PROJECT_CONFIG_FILE.

Before including this CPack module in your CMakeLists.txt file, there are a variety of variables that can be set to customize the resulting installers. The most commonly-used variables are:

CPACK_PACKAGE_NAME

The name of the package (or application). If not specified, defaults to the project name.

CPACK_PACKAGE_VENDOR

The name of the package vendor. (e.g., Kitware).

CPACK_PACKAGE_DIRECTORY

The directory in which CPack is doing its packaging. If it is not set then this will default (internally) to the build dir. This variable may be defined in CPack config file or from the cpack command line option -B. If set the command line option override the value found in the config file.

CPACK_PACKAGE_VERSION_MAJOR
Package major Version

CPACK_PACKAGE_VERSION_MINOR
Package minor Version

CPACK_PACKAGE_VERSION_PATCH
Package patch Version

CPACK_PACKAGE_DESCRIPTION_FILE
A text file used to describe the project. Used, for example, the introduction screen of a CPack-generated Windows installer to describe the project.

CPACK_PACKAGE_DESCRIPTION_SUMMARY
Short description of the project (only a few words).

CPACK_PACKAGE_FILE_NAME
The name of the package file to generate, not including the extension. For example, cmake-2.6.1-Linux-i686. The default value is:

```
${CPACK_PACKAGE_NAME}-${CPACK_PACKAGE_VERSION}-${CPACK_SYSTEM_NAME}.
```

CPACK_PACKAGE_INSTALL_DIRECTORY
Installation directory on the target system. This may be used by some CPack generators like NSIS to create an installation directory e.g., CMake 2.5 below the installation prefix. All installed element will be put inside this directory.

CPACK_PACKAGE_ICON
A branding image that will be displayed inside the installer (used by GUI installers).

CPACK_PROJECT_CONFIG_FILE
CPack-time project CPack configuration file. This file included at cpack time, once per generator after CPack has set CPACK_GENERATOR to the actual generator being used. It allows per-generator setting of CPACK_* variables at cpack time.

CPACK_RESOURCE_FILE_LICENSE
License to be embedded in the installer. It will typically be displayed to the user by the produced installer (often with an explicit Accept button, for graphical installers) prior to installation. This license file is NOT added to installed file but is used by some CPack generators like NSIS. If you want to install a license file (may be the same as this one) along with your project you must add an appropriate CMake INSTALL command in your CMakeLists.txt.

CPACK_RESOURCE_FILE_README
ReadMe file to be embedded in the installer. It typically describes in some detail the purpose of the project during the installation. Not all CPack generators uses this file.

CPACK_RESOURCE_FILE_WELCOME
Welcome file to be embedded in the installer. It welcomes users to this installer. Typically used in the graphical installers on Windows and Mac OS X.

CPACK_MONOLITHIC_INSTALL
Disables the component-based installation mechanism. When set the component specification is ignored and all installed items are put in a single MONOLITHIC package. Some CPack generators do monolithic packaging by default and may be asked to do component packaging by setting CPACK_<GENNAME>_COMPONENT_INSTALL to 1/TRUE.

CPACK_GENERATOR
List of CPack generators to use. If not specified, CPack will create a set of options CPACK_BINARY_<GENNAME> (e.g., CPACK_BINARY_NSIS) allowing the user to enable/disable individual generators. This variable may be used on the command line as well as in:

```
cpack -D CPACK_GENERATOR="ZIP;TGZ" /path/to/build/tree
```

CPACK_OUTPUT_CONFIG_FILE
The name of the CPack binary configuration file. This file is the CPack configuration generated by the CPack module for binary installers. Defaults to CPackConfig.cmake.

CPACK_PACKAGE_EXECUTABLES
Lists each of the executables and associated text label to be used to create Start Menu shortcuts. For example, setting this to the list ccmake;CMake will create a shortcut named CMake that will execute the installed executable ccmake. Not all CPack generators use it (at least NSIS, WIX and OSXX11 do).

CPACK_STRIP_FILES
List of files to be stripped. Starting with CMake 2.6.0 CPACK_STRIP_FILES will be a boolean variable which enables stripping of all files (a list of files evaluates to TRUE in CMake, so this change is compatible).

The following CPack variables are specific to source packages, and will not affect binary packages:

CPACK_SOURCE_PACKAGE_FILE_NAME
The name of the source package. For example cmake-2.6.1.

CPACK_SOURCE_STRIP_FILES
List of files in the source tree that will be stripped. Starting with CMake 2.6.0 CPACK_SOURCE_STRIP_FILES will be a boolean variable which enables stripping of all files (a list of files evaluates to TRUE in CMake, so this change is compatible).

CPACK_SOURCE_GENERATOR
List of generators used for the source packages. As with CPACK_GENERATOR, if this is not specified then CPack will create a set of options (e.g., CPACK_SOURCE_ZIP) allowing users to select which packages will be generated.

CPACK_SOURCE_OUTPUT_CONFIG_FILE
The name of the CPack source configuration file. This file is the CPack configuration generated by the CPack module for source installers. Defaults to CPackSourceConfig.cmake.

CPACK_SOURCE_IGNORE_FILES
Pattern of files in the source tree that wont be packaged when building a source package. This is a list of regular expression patterns (that must be properly escaped), e.g., /CVS/;/.svn/;.swp$;.#;/#;.*~;cscope.*

The following variables are for advanced uses of CPack:

CPACK_CMAKE_GENERATOR
What CMake generator should be used if the project is CMake project. Defaults to the value of CMAKE_GENERATOR few users will want to change this setting.

CPACK_INSTALL_CMAKE_PROJECTS
List of four values that specify what project to install. The four values are: Build directory, Project Name, Project Component, Directory. If omitted, CPack will build an installer that installers everything.

CPACK_SYSTEM_NAME
System name, defaults to the value of ${CMAKE_SYSTEM_NAME}.

CPACK_PACKAGE_VERSION
Package full version, used internally. By default, this is built from CPACK_PACKAGE_VERSION_MAJOR, CPACK_PACKAGE_VERSION_MINOR, and CPACK_PACKAGE_VERSION_PATCH.

CPACK_TOPLEVEL_TAG
Directory for the installed files.

CPACK_INSTALL_COMMANDS
Extra commands to install components.

CPACK_INSTALLED_DIRECTORIES
Extra directories to install.

CPACK_PACKAGE_INSTALL_REGISTRY_KEY
Registry key used when installing this project. This is only used by installer for Windows. The default value is based on the installation directory.

CPACK_CREATE_DESKTOP_LINKS
List of desktop links to create. Each desktop link requires a corresponding start menu shortcut as created by **CPACK_PACKAGE_EXECUTABLES**.

CPACK_BINARY_<GENNAME>
CPack generated options for binary generators. The CPack.cmake module generates (when CPACK_GENERATOR is not set) a set of CMake options (see CMake option command) which may then be used to select the CPack generator(s) to be used when launching the package target.

Provide options to choose generators we might check here if the required tools for the generates exist and set the defaults according to the results

CPackWIX
CPack WiX generator specific options

Variables specific to CPack WiX generator
The following variables are specific to the installers built on Windows using WiX.

CPACK_WIX_UPGRADE_GUID
Upgrade GUID (**Product/@UpgradeCode**)

Will be automatically generated unless explicitly provided.

It should be explicitly set to a constant generated gloabally unique identifier (GUID) to allow your installers to replace existing installations that use the same GUID.

You may for example explicitly set this variable in your CMakeLists.txt to the value that has been generated per default. You should not use GUIDs that you did not generate yourself or which may belong to other projects.

A GUID shall have the following fixed length syntax:

```
XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

(each X represents an uppercase hexadecimal digit)

CPACK_WIX_PRODUCT_GUID
Product GUID (**Product/@Id**)

Will be automatically generated unless explicitly provided.

If explicitly provided this will set the Product Id of your installer.

The installer will abort if it detects a pre-existing installation that uses the same GUID.

The GUID shall use the syntax described for CPACK_WIX_UPGRADE_GUID.

CPACK_WIX_LICENSE_RTF
RTF License File

If CPACK_RESOURCE_FILE_LICENSE has an .rtf extension it is used as-is.

If CPACK_RESOURCE_FILE_LICENSE has an .txt extension it is implicitly converted to RTF by the WiX Generator. The expected encoding of the .txt file is UTF-8.

With CPACK_WIX_LICENSE_RTF you can override the license file used by the WiX

Generator in case CPACK_RESOURCE_FILE_LICENSE is in an unsupported format or the .txt -> .rtf conversion does not work as expected.

CPACK_WIX_PRODUCT_ICON
The Icon shown next to the program name in Add/Remove programs.

If set, this icon is used in place of the default icon.

CPACK_WIX_UI_REF
This variable allows you to override the Id of the **<UIRef>** element in the WiX template.

The default is **WixUI_InstallDir** in case no CPack components have been defined and **WixUI_FeatureTree** otherwise.

CPACK_WIX_UI_BANNER
The bitmap will appear at the top of all installer pages other than the welcome and completion dialogs.

If set, this image will replace the default banner image.

This image must be 493 by 58 pixels.

CPACK_WIX_UI_DIALOG
Background bitmap used on the welcome and completion dialogs.

If this variable is set, the installer will replace the default dialog image.

This image must be 493 by 312 pixels.

CPACK_WIX_PROGRAM_MENU_FOLDER
Start menu folder name for launcher.

If this variable is not set, it will be initialized with CPACK_PACKAGE_NAME

CPACK_WIX_CULTURES
Language(s) of the installer

Languages are compiled into the WixUI extension library. To use them, simply provide the name of the culture. If you specify more than one culture identifier in a comma or semicolon delimited list, the first one that is found will be used. You can find a list of supported languages at: *http://wix.sourceforge.net/manual-wix3/WixUI_localization.htm*

CPACK_WIX_TEMPLATE
Template file for WiX generation

If this variable is set, the specified template will be used to generate the WiX wxs file. This should be used if further customization of the output is required.

If this variable is not set, the default MSI template included with CMake will be used.

CPACK_WIX_PATCH_FILE
Optional XML file with fragments to be inserted into generated WiX sources

This optional variable can be used to specify an XML file that the WiX generator will use to inject fragments into its generated source files.

Patch files understood by the CPack WiX generator roughly follow this RELAX NG compact schema:

```
start = CPackWiXPatch

CPackWiXPatch = element CPackWiXPatch { CPackWiXFragment* }

CPackWiXFragment = element CPackWiXFragment
{
attribute Id { string },
fragmentContent*
```

```
}

fragmentContent = element * - CPackWiXFragment
{
(attribute * { text } | text | fragmentContent)*
}
```

Currently fragments can be injected into most Component, File and Directory elements.

The following example illustrates how this works.

Given that the WiX generator creates the following XML element:

```
<Component Id="CM_CP_applications.bin.my_libapp.exe" Guid="*"/>
```

The following XML patch file may be used to inject an Environment element into it:

```
<CPackWiXPatch>
<CPackWiXFragment Id="CM_CP_applications.bin.my_libapp.exe">
<Environment Id="MyEnvironment" Action="set"
Name="MyVariableName" Value="MyVariableValue"/>
</CPackWiXFragment>
</CPackWiXPatch>
```

CPACK_WIX_EXTRA_SOURCES
Extra WiX source files

This variable provides an optional list of extra WiX source files (.wxs) that should be compiled and linked. The full path to source files is required.

CPACK_WIX_EXTRA_OBJECTS
Extra WiX object files or libraries

This variable provides an optional list of extra WiX object (.wixobj) and/or WiX library (.wixlib) files. The full path to objects and libraries is required.

CPACK_WIX_EXTENSIONS
This variable provides a list of additional extensions for the WiX tools light and candle.

CPACK_WIX_<TOOL>_EXTENSIONS
This is the tool specific version of CPACK_WIX_EXTENSIONS. **<TOOL>** can be either LIGHT or CANDLE.

CPACK_WIX_<TOOL>_EXTRA_FLAGS
This list variable allows you to pass additional flags to the WiX tool **<TOOL>**.

Use it at your own risk. Future versions of CPack may generate flags which may be in conflict with your own flags.

<TOOL> can be either LIGHT or CANDLE.

CPACK_WIX_CMAKE_PACKAGE_REGISTRY
If this variable is set the generated installer will create an entry in the windows registry key **HKEY_LOCAL_MACHINESoftwareKitwareCMakePackages<package>** The value for **<package>** is provided by this variable.

Assuming you also install a CMake configuration file this will allow other CMake projects to find your package with the **find_package()** command.

CTest
Configure a project for testing with CTest/CDash

Include this module in the top CMakeLists.txt file of a project to enable testing with CTest and dashboard submissions to CDash:

```
project(MyProject)
```

```
...
include(CTest)
```

The module automatically creates a BUILD_TESTING option that selects whether to enable testing support (ON by default). After including the module, use code like

```
if(BUILD_TESTING)
# ... CMake code to create tests ...
endif()
```

to creating tests when testing is enabled.

To enable submissions to a CDash server, create a CTestConfig.cmake file at the top of the project with content such as

```
set(CTEST_PROJECT_NAME "MyProject")
set(CTEST_NIGHTLY_START_TIME "01:00:00 UTC")
set(CTEST_DROP_METHOD "http")
set(CTEST_DROP_SITE "my.cdash.org")
set(CTEST_DROP_LOCATION "/submit.php?project=MyProject")
set(CTEST_DROP_SITE_CDASH TRUE)
```

(the CDash server can provide the file to a project administrator who configures MyProject). Settings in the config file are shared by both this CTest module and the CTest command-line tools dashboard script mode (ctest -S).

While building a project for submission to CDash, CTest scans the build output for errors and warnings and reports them with surrounding context from the build log. This generic approach works for all build tools, but does not give details about the command invocation that produced a given problem. One may get more detailed reports by adding

```
set(CTEST_USE_LAUNCHERS 1)
```

to the CTestConfig.cmake file. When this option is enabled, the CTest module tells CMakes Makefile generators to invoke every command in the generated build system through a CTest launcher program. (Currently the CTEST_USE_LAUNCHERS option is ignored on non-Makefile generators.) During a manual build each launcher transparently runs the command it wraps. During a CTest-driven build for submission to CDash each launcher reports detailed information when its command fails or warns. (Setting CTEST_USE_LAUNCHERS in CTestConfig.cmake is convenient, but also adds the launcher overhead even for manual builds. One may instead set it in a CTest dashboard script and add it to the CMake cache for the build tree.)

CTestScriptMode
This file is read by ctest in script mode (-S)

CTestUseLaunchers
Set the RULE_LAUNCH_* global properties when CTEST_USE_LAUNCHERS is on.

CTestUseLaunchers is automatically included when you include(CTest). However, it is split out into its own module file so projects can use the CTEST_USE_LAUNCHERS functionality independently.

To use launchers, set CTEST_USE_LAUNCHERS to ON in a ctest -S dashboard script, and then also set it in the cache of the configured project. Both cmake and ctest need to know the value of it for the launchers to work properly. CMake needs to know in order to generate proper build rules, and ctest, in order to produce the proper error and warning analysis.

For convenience, you may set the ENV variable CTEST_USE_LAUNCHERS_DEFAULT in your ctest -S script, too. Then, as long as your CMakeLists uses include(CTest) or include(CTestUseLaunchers), it will use the value of the ENV variable to initialize a CTEST_USE_LAUNCHERS cache variable. This cache variable initialization only occurs if CTEST_USE_LAUNCHERS is not already defined.

Dart

Configure a project for testing with CTest or old Dart Tcl Client

This file is the backwards-compatibility version of the CTest module. It supports using the old Dart 1 Tcl client for driving dashboard submissions as well as testing with CTest. This module should be included in the CMakeLists.txt file at the top of a project. Typical usage:

```
include(Dart)
if(BUILD_TESTING)
# ... testing related CMake code ...
endif()
```

The BUILD_TESTING option is created by the Dart module to determine whether testing support should be enabled. The default is ON.

DeployQt4

Functions to help assemble a standalone Qt4 executable.

A collection of CMake utility functions useful for deploying Qt4 executables.

The following functions are provided by this module:

```
write_qt4_conf
resolve_qt4_paths
fixup_qt4_executable
install_qt4_plugin_path
install_qt4_plugin
install_qt4_executable
```

Requires CMake 2.6 or greater because it uses function and PARENT_SCOPE. Also depends on BundleUtilities.cmake.

```
WRITE_QT4_CONF(<qt_conf_dir> <qt_conf_contents>)
```

Writes a qt.conf file with the <qt_conf_contents> into <qt_conf_dir>.

```
RESOLVE_QT4_PATHS(<paths_var> [<executable_path>])
```

Loop through <paths_var> list and if any dont exist resolve them relative to the <executable_path> (if supplied) or the CMAKE_INSTALL_PREFIX.

```
FIXUP_QT4_EXECUTABLE(<executable> [<qtplugins> <libs> <dirs> <plugins_dir> <request_qt_con
```

Copies Qt plugins, writes a Qt configuration file (if needed) and fixes up a Qt4 executable using BundleUtilities so it is standalone and can be drag-and-drop copied to another machine as long as all of the system libraries are compatible.

<executable> should point to the executable to be fixed-up.

<qtplugins> should contain a list of the names or paths of any Qt plugins to be installed.

<libs> will be passed to BundleUtilities and should be a list of any already installed plugins, libraries or executables to also be fixed-up.

<dirs> will be passed to BundleUtilities and should contain and directories to be searched to find library dependencies.

<plugins_dir> allows an custom plugins directory to be used.

<request_qt_conf> will force a qt.conf file to be written even if not needed.

```
INSTALL_QT4_PLUGIN_PATH(plugin executable copy installed_plugin_path_var <plugins_dir> <co
```

Install (or copy) a resolved <plugin> to the default plugins directory (or <plugins_dir>) relative to <executable> and store the result in <installed_plugin_path_var>.

If <copy> is set to TRUE then the plugins will be copied rather than installed. This is to allow this module to be used at CMake time rather than install time.

If <component> is set then anything installed will use this COMPONENT.

```
INSTALL_QT4_PLUGIN(plugin executable copy installed_plugin_path_var <plugins_dir> <compone
```

Install (or copy) an unresolved <plugin> to the default plugins directory (or <plugins_dir>) relative to <executable> and store the result in <installed_plugin_path_var>. See documentation of INSTALL_QT4_PLUGIN_PATH.

```
INSTALL_QT4_EXECUTABLE(<executable> [<qtplugins> <libs> <dirs> <plugins_dir> <request_qt_c
```

Installs Qt plugins, writes a Qt configuration file (if needed) and fixes up a Qt4 executable using BundleUtilities so it is standalone and can be drag-and-drop copied to another machine as long as all of the system libraries are compatible. The executable will be fixed-up at install time. <component> is the COMPONENT used for bundle fixup and plugin installation. See documentation of FIXUP_QT4_BUNDLE.

Documentation

DocumentationVTK.cmake

This file provides support for the VTK documentation framework. It relies on several tools (Doxygen, Perl, etc).

ExternalData

Manage data files stored outside source tree

Use this module to unambiguously reference data files stored outside the source tree and fetch them at build time from arbitrary local and remote content-addressed locations. Functions provided by this module recognize arguments with the syntax **DATA{<name>}** as references to external data, replace them with full paths to local copies of those data, and create build rules to fetch and update the local copies.

The **DATA{}** syntax is literal and the **<name>** is a full or relative path within the source tree. The source tree must contain either a real data file at **<name>** or a content link at **<name><ext>** containing a hash of the real file using a hash algorithm corresponding to **<ext>**. For example, the argument **DATA{img.png}** may be satisfied by either a real **img.png** file in the current source directory or a **img.png.md5** file containing its MD5 sum.

The **ExternalData_Expand_Arguments** function evaluates **DATA{}** references in its arguments and constructs a new list of arguments:

```
ExternalData_Expand_Arguments(
<target> # Name of data management target
<outVar> # Output variable
[args...] # Input arguments, DATA{} allowed
)
```

It replaces each **DATA{}** reference in an argument with the full path of a real data file on disk that will exist after the **<target>** builds.

The **ExternalData_Add_Test** function wraps around the CMake **add_test()** command but supports **DATA{}** references in its arguments:

```
ExternalData_Add_Test(
<target> # Name of data management target
)
```

It passes its arguments through **ExternalData_Expand_Arguments** and then invokes the **add_test()** command using the results.

The **ExternalData_Add_Target** function creates a custom target to manage local instances of

data files stored externally:

```
ExternalData_Add_Target(
<target> # Name of data management target
)
```

It creates custom commands in the target as necessary to make data files available for each **DATA{}** reference previously evaluated by other functions provided by this module. A list of URL templates may be provided in the variable **ExternalData_URL_TEMPLATES** using the placeholders **%(algo)** and **%(hash)** in each template. Data fetch rules try each URL template in order by substituting the hash algorithm name for **%(algo)** and the hash value for **%(hash)**.

The following hash algorithms are supported:

```
%(algo) <ext> Description
------- ----- -----------
MD5 .md5 Message-Digest Algorithm 5, RFC 1321
SHA1 .sha1 US Secure Hash Algorithm 1, RFC 3174
SHA224 .sha224 US Secure Hash Algorithms, RFC 4634
SHA256 .sha256 US Secure Hash Algorithms, RFC 4634
SHA384 .sha384 US Secure Hash Algorithms, RFC 4634
SHA512 .sha512 US Secure Hash Algorithms, RFC 4634
```

Note that the hashes are used only for unique data identification and download verification. This is not security software.

Example usage:

```
include(ExternalData)
set(ExternalData_URL_TEMPLATES "file:///local/%(algo)/%(hash)"
"file:////host/share/%(algo)/%(hash)"
                                )" -P " -- http://data.org/%(algo)/%(hash)
ExternalData_Add_Test(MyData
NAME MyTest
COMMAND MyExe DATA{MyInput.png}
)
ExternalData_Add_Target(MyData)
```

When test **MyTest** runs the **DATA{MyInput.png}** argument will be replaced by the full path to a real instance of the data file **MyInput.png** on disk. If the source tree contains a content link such as **MyInput.png.md5** then the **MyData** target creates a real **MyInput.png** in the build tree.

The **DATA{}** syntax can be told to fetch a file series using the form **DATA{<name>,:}**, where the **:** is literal. If the source tree contains a group of files or content links named like a series then a reference to one member adds rules to fetch all of them. Although all members of a series are fetched, only the file originally named by the **DATA{}** argument is substituted for it. The default configuration recognizes file series names ending with **#.ext**, **_#.ext**, **.#.ext**, or **-#.ext** where **#** is a sequence of decimal digits and **.ext** is any single extension. Configure it with a regex that parses **<number>** and **<suffix>** parts from the end of **<name>**:

```
ExternalData_SERIES_PARSE = regex of the form (<number>)(<suffix>)$
```

For more complicated cases set:

```
ExternalData_SERIES_PARSE = regex with at least two () groups
ExternalData_SERIES_PARSE_PREFIX = <prefix> regex group number, if any
ExternalData_SERIES_PARSE_NUMBER = <number> regex group number
ExternalData_SERIES_PARSE_SUFFIX = <suffix> regex group number
```

Configure series number matching with a regex that matches the **<number>** part of series

members named **<prefix><number><suffix>**:

```
ExternalData_SERIES_MATCH = regex matching <number> in all series members
```

Note that the **<suffix>** of a series does not include a hash-algorithm extension.

The **DATA**{} syntax can alternatively match files associated with the named file and contained in the same directory. Associated files may be specified by options using the syntax **DATA**{**<name>**,**<opt1>**,**<opt2>**,**...**}. Each option may specify one file by name or specify a regular expression to match file names using the syntax **REGEX:<regex>**. For example, the arguments:

```
DATA{MyData/MyInput.mhd,MyInput.img} # File pair
DATA{MyData/MyFrames00.png,REGEX:MyFrames[0-9]+\\.png} # Series
```

will pass **MyInput.mha** and **MyFrames00.png** on the command line but ensure that the associated files are present next to them.

The **DATA**{} syntax may reference a directory using a trailing slash and a list of associated files. The form **DATA**{**<name>/**,**<opt1>**,**<opt2>**,**...**} adds rules to fetch any files in the directory that match one of the associated file options. For example, the argument **DATA**{**MyDataDir/**,**REGEX:.***} will pass the full path to a **MyDataDir** directory on the command line and ensure that the directory contains files corresponding to every file or content link in the **MyDataDir** source directory.

The variable **ExternalData_LINK_CONTENT** may be set to the name of a supported hash algorithm to enable automatic conversion of real data files referenced by the **DATA**{} syntax into content links. For each such **<file>** a content link named **<file><ext>** is created. The original file is renamed to the form **.ExternalData_<algo>_<hash>** to stage it for future transmission to one of the locations in the list of URL templates (by means outside the scope of this module). The data fetch rule created for the content link will use the staged object if it cannot be found using any URL template.

The variable **ExternalData_OBJECT_STORES** may be set to a list of local directories that store objects using the layout **<dir>/%(algo)/%(hash)**. These directories will be searched first for a needed object. If the object is not available in any store then it will be fetched remotely using the URL templates and added to the first local store listed. If no stores are specified the default is a location inside the build tree.

The variable **ExternalData_SOURCE_ROOT** may be set to the highest source directory containing any path named by a **DATA**{} reference. The default is **CMAKE_SOURCE_DIR**. **ExternalData_SOURCE_ROOT** and **CMAKE_SOURCE_DIR** must refer to directories within a single source distribution (e.g. they come together in one tarball).

The variable **ExternalData_BINARY_ROOT** may be set to the directory to hold the real data files named by expanded **DATA**{} references. The default is **CMAKE_BINARY_DIR**. The directory layout will mirror that of content links under **ExternalData_SOURCE_ROOT**.

Variables **ExternalData_TIMEOUT_INACTIVITY** and **ExternalData_TIMEOUT_ABSOLUTE** set the download inactivity and absolute timeouts, in seconds. The defaults are 60 seconds and 300 seconds, respectively. Set either timeout to 0 seconds to disable enforcement.

ExternalProject

Create custom targets to build projects in external trees

The **ExternalProject_Add** function creates a custom target to drive download, update/patch, configure, build, install and test steps of an external project:

```
ExternalProject_Add(<name> # Name for custom target
[DEPENDS projects...] # Targets on which the project depends
[PREFIX dir] # Root dir for entire project
```

```
    [LIST_SEPARATOR sep] # Sep to be replaced by ; in cmd lines
    [TMP_DIR dir] # Directory to store temporary files
    [STAMP_DIR dir] # Directory to store step timestamps
    #--Download step--------------
    [DOWNLOAD_NAME fname] # File name to store (if not end of URL)
    [DOWNLOAD_DIR dir] # Directory to store downloaded files
    [DOWNLOAD_COMMAND cmd...] # Command to download source tree
    [CVS_REPOSITORY cvsroot] # CVSROOT of CVS repository
    [CVS_MODULE mod] # Module to checkout from CVS repo
    [CVS_TAG tag] # Tag to checkout from CVS repo
    [SVN_REPOSITORY url] # URL of Subversion repo
    [SVN_REVISION rev] # Revision to checkout from Subversion repo
    [SVN_USERNAME john ] # Username for Subversion checkout and update
    [SVN_PASSWORD doe ] # Password for Subversion checkout and update
    [SVN_TRUST_CERT 1 ] # Trust the Subversion server site certificate
    [GIT_REPOSITORY url] # URL of git repo
    [GIT_TAG tag] # Git branch name, commit id or tag
    [GIT_SUBMODULES modules...] # Git submodules that shall be updated, all if empty
    [HG_REPOSITORY url] # URL of mercurial repo
    [HG_TAG tag] # Mercurial branch name, commit id or tag
    [URL /.../src.tgz] # Full path or URL of source
    [URL_HASH ALGO=value] # Hash of file at URL
    [URL_MD5 md5] # Equivalent to URL_HASH MD5=md5
    [TLS_VERIFY bool] # Should certificate for https be checked
    [TLS_CAINFO file] # Path to a certificate authority file
    [TIMEOUT seconds] # Time allowed for file download operations
    #--Update/Patch step----------
    [UPDATE_COMMAND cmd...] # Source work-tree update command
    [PATCH_COMMAND cmd...] # Command to patch downloaded source
    #--Configure step-------------
    [SOURCE_DIR dir] # Source dir to be used for build
    [CONFIGURE_COMMAND cmd...] # Build tree configuration command
    [CMAKE_COMMAND /.../cmake] # Specify alternative cmake executable
    [CMAKE_GENERATOR gen] # Specify generator for native build
    [CMAKE_GENERATOR_TOOLSET t] # Generator-specific toolset name
    [CMAKE_ARGS args...] # Arguments to CMake command line
    [CMAKE_CACHE_ARGS args...] # Initial cache arguments, of the form -Dvar:string=on
    #--Build step-----------------
    [BINARY_DIR dir] # Specify build dir location
    [BUILD_COMMAND cmd...] # Command to drive the native build
    [BUILD_IN_SOURCE 1] # Use source dir for build dir
    #--Install step---------------
    [INSTALL_DIR dir] # Installation prefix
    [INSTALL_COMMAND cmd...] # Command to drive install after build
    #--Test step------------------
    [TEST_BEFORE_INSTALL 1] # Add test step executed before install step
    [TEST_AFTER_INSTALL 1] # Add test step executed after install step
    [TEST_COMMAND cmd...] # Command to drive test
    #--Output logging-------------
    [LOG_DOWNLOAD 1] # Wrap download in script to log output
    [LOG_UPDATE 1] # Wrap update in script to log output
    [LOG_CONFIGURE 1] # Wrap configure in script to log output
    [LOG_BUILD 1] # Wrap build in script to log output
    [LOG_TEST 1] # Wrap test in script to log output
```

```
[LOG_INSTALL 1] # Wrap install in script to log output
#--Custom targets-------------
[STEP_TARGETS st1 st2 ...] # Generate custom targets for these steps
)
```

The ***_DIR** options specify directories for the project, with default directories computed as follows. If the **PREFIX** option is given to **ExternalProject_Add()** or the **EP_PREFIX** directory property is set, then an external project is built and installed under the specified prefix:

```
TMP_DIR = <prefix>/tmp
STAMP_DIR = <prefix>/src/<name>-stamp
DOWNLOAD_DIR = <prefix>/src
SOURCE_DIR = <prefix>/src/<name>
BINARY_DIR = <prefix>/src/<name>-build
INSTALL_DIR = <prefix>
```

Otherwise, if the **EP_BASE** directory property is set then components of an external project are stored under the specified base:

```
TMP_DIR = <base>/tmp/<name>
STAMP_DIR = <base>/Stamp/<name>
DOWNLOAD_DIR = <base>/Download/<name>
SOURCE_DIR = <base>/Source/<name>
BINARY_DIR = <base>/Build/<name>
INSTALL_DIR = <base>/Install/<name>
```

If no **PREFIX**, **EP_PREFIX**, or **EP_BASE** is specified then the default is to set **PREFIX** to **<name>-prefix**. Relative paths are interpreted with respect to the build directory corresponding to the source directory in which **ExternalProject_Add** is invoked.

If **SOURCE_DIR** is explicitly set to an existing directory the project will be built from it. Otherwise a download step must be specified using one of the **DOWNLOAD_COMMAND**, **CVS_***, **SVN_***, or **URL** options. The **URL** option may refer locally to a directory or source tarball, or refer to a remote tarball (e.g. **http://.../src.tgz**).

The **ExternalProject_Add_Step** function adds a custom step to an external project:

```
ExternalProject_Add_Step(<name> <step> # Names of project and custom step
[COMMAND cmd...] # Command line invoked by this step
[COMMENT "text..."] # Text printed when step executes
[DEPENDEES steps...] # Steps on which this step depends
[DEPENDERS steps...] # Steps that depend on this step
[DEPENDS files...] # Files on which this step depends
[ALWAYS 1] # No stamp file, step always runs
[WORKING_DIRECTORY dir] # Working directory for command
[LOG 1] # Wrap step in script to log output
)
```

The command line, comment, and working directory of every standard and custom step is processed to replace tokens **<SOURCE_DIR>**, **<BINARY_DIR>**, **<INSTALL_DIR>**, and **<TMP_DIR>** with corresponding property values.

Any builtin step that specifies a **<step>_COMMAND cmd...**or custom step that specifies a **COMMAND cmd...**ma y specify additional command lines using the form **COMMAND cmd...**. At build time the commands will be executed in order and aborted if any one fails. For example:

```
... BUILD_COMMAND make COMMAND echo done ...
```

specifies to run **make** and then **echo done**during the build step. Whether the curren t working

directory is preserved between commands is not defined. Behavior of shell operators like **&&** is not defined.

The **ExternalProject_Get_Property** function retrieves external project target properties:

```
ExternalProject_Get_Property(<name> [prop1 [prop2 [...]]])
```

It stores property values in variables of the same name. Property names correspond to the keyword argument names of **ExternalProject_Add**.

The **ExternalProject_Add_StepTargets** function generates custom targets for the steps listed:

```
ExternalProject_Add_StepTargets(<name> [step1 [step2 [...]]])
```

If **STEP_TARGETS** is set then **ExternalProject_Add_StepTargets** is automatically called at the end of matching calls to **ExternalProject_Add_Step**. Pass **STEP_TARGETS** explicitly to individual **ExternalProject_Add** calls, or implicitly to all **ExternalProject_Add** calls by setting the directory property **EP_STEP_TARGETS**.

If **STEP_TARGETS** is not set, clients may still manually call **ExternalProject_Add_StepTargets** after calling **ExternalProject_Add** or **ExternalProject_Add_Step**.

This functionality is provided to make it easy to drive the steps independently of each other by specifying targets on build command lines. For example, you may be submitting to a sub-project based dashboard, where you want to drive the configure portion of the build, then submit to the dashboard, followed by the build portion, followed by tests. If you invoke a custom target that depends on a step halfway through the step dependency chain, then all the previous steps will also run to ensure everything is up to date.

For example, to drive configure, build and test steps independently for each **ExternalProject_Add** call in your project, write the following line prior to any **ExternalProject_Add** calls in your **CMakeLists.txt** file:

```
set_property(DIRECTORY PROPERTY EP_STEP_TARGETS configure build test)
```

FeatureSummary

Macros for generating a summary of enabled/disabled features

This module provides the macros feature_summary(), set_package_properties() and add_feature_info(). For compatibility it also still provides set_package_info(), set_feature_info(), print_enabled_features() and print_disabled_features().

These macros can be used to generate a summary of enabled and disabled packages and/or feature for a build tree:

```
-- The following OPTIONAL packages have been found:
LibXml2 (required version >= 2.4) , XML processing library. , <http://xmlsoft.org>
* Enables HTML-import in MyWordProcessor
* Enables odt-export in MyWordProcessor
PNG , A PNG image library. , <http://www.libpng.org/pub/png/>
* Enables saving screenshots
-- The following OPTIONAL packages have not been found:
Lua51 , The Lua scripting language. , <http://www.lua.org>
* Enables macros in MyWordProcessor
Foo , Foo provides cool stuff.
```

```
FEATURE_SUMMARY( [FILENAME <file>]
[APPEND]
[VAR <variable_name>]
[INCLUDE_QUIET_PACKAGES]
[FATAL_ON_MISSING_REQUIRED_PACKAGES]
[DESCRIPTION "Found packages:"]
```

```
WHAT (ALL | PACKAGES_FOUND | PACKAGES_NOT_FOUND
| ENABLED_FEATURES | DISABLED_FEATURES]
)
```

The FEATURE_SUMMARY() macro can be used to print information about enabled or disabled packages or features of a project. By default, only the names of the features/packages will be printed and their required version when one was specified. Use SET_PACKAGE_PROPERTIES() to add more useful information, like e.g. a download URL for the respective package or their purpose in the project.

The WHAT option is the only mandatory option. Here you specify what information will be printed:

```
ALL: print everything
ENABLED_FEATURES: the list of all features which are enabled
DISABLED_FEATURES: the list of all features which are disabled
PACKAGES_FOUND: the list of all packages which have been found
PACKAGES_NOT_FOUND: the list of all packages which have not been found
OPTIONAL_PACKAGES_FOUND: only those packages which have been found which have the type OPT
OPTIONAL_PACKAGES_NOT_FOUND: only those packages which have not been found which have the
RECOMMENDED_PACKAGES_FOUND: only those packages which have been found which have the type
RECOMMENDED_PACKAGES_NOT_FOUND: only those packages which have not been found which have t
REQUIRED_PACKAGES_FOUND: only those packages which have been found which have the type REC
REQUIRED_PACKAGES_NOT_FOUND: only those packages which have not been found which have the
RUNTIME_PACKAGES_FOUND: only those packages which have been found which have the type RUNT
RUNTIME_PACKAGES_NOT_FOUND: only those packages which have not been found which have the t
```

If a FILENAME is given, the information is printed into this file. If APPEND is used, it is appended to this file, otherwise the file is overwritten if it already existed. If the VAR option is used, the information is printed into the specified variable. If FILENAME is not used, the information is printed to the terminal. Using the DESCRIPTION option a description or headline can be set which will be printed above the actual content. If INCLUDE_QUIET_PACKAGES is given, packages which have been searched with find_package(... QUIET) will also be listed. By default they are skipped. If FATAL_ON_MISSING_REQUIRED_PACKAGES is given, CMake will abort if a package which is marked as REQUIRED has not been found.

Example 1, append everything to a file:

```
feature_summary(WHAT ALL
FILENAME ${CMAKE_BINARY_DIR}/all.log APPEND)
```

Example 2, print the enabled features into the variable enabledFeaturesText, including QUIET packages:

```
feature_summary(WHAT ENABLED_FEATURES
INCLUDE_QUIET_PACKAGES
DESCRIPTION "Enabled Features:"
VAR enabledFeaturesText)
message(STATUS "${enabledFeaturesText}")
```

```
SET_PACKAGE_PROPERTIES(<name> PROPERTIES [ URL <url> ]
[ DESCRIPTION <description> ]
[ TYPE (RUNTIME|OPTIONAL|RECOMMENDED|REQUIRED) ]
[ PURPOSE <purpose> ]
)
```

Use this macro to set up information about the named package, which can then be displayed via FEATURE_SUMMARY(). This can be done either directly in the Find-module or in the project which uses the module after the find_package() call. The features for which information can be set

are added automatically by the find_package() command.

URL: this should be the homepage of the package, or something similar. Ideally this is set already directly in the Find-module.

DESCRIPTION: A short description what that package is, at most one sentence. Ideally this is set already directly in the Find-module.

TYPE: What type of dependency has the using project on that package. Default is OPTIONAL. In this case it is a package which can be used by the project when available at buildtime, but it also work without. RECOMMENDED is similar to OPTIONAL, i.e. the project will build if the package is not present, but the functionality of the resulting binaries will be severly limited. If a REQUIRED package is not available at buildtime, the project may not even build. This can be combined with the FATAL_ON_MISSING_REQUIRED_PACKAGES argument for feature_summary(). Last, a RUNTIME package is a package which is actually not used at all during the build, but which is required for actually running the resulting binaries. So if such a package is missing, the project can still be built, but it may not work later on. If set_package_properties() is called multiple times for the same package with different TYPEs, the TYPE is only changed to higher TYPEs (RUNTIME < OPTIONAL < RECOMMENDED < REQUIRED), lower TYPEs are ignored. The TYPE property is project-specific, so it cannot be set by the Find-module, but must be set in the project.

PURPOSE: This describes which features this package enables in the project, i.e. it tells the user what functionality he gets in the resulting binaries. If set_package_properties() is called multiple times for a package, all PURPOSE properties are appended to a list of purposes of the package in the project. As the TYPE property, also the PURPOSE property is project-specific, so it cannot be set by the Find-module, but must be set in the project.

Example for setting the info for a package:

```
find_package(LibXml2)
set_package_properties(LibXml2 PROPERTIES DESCRIPTION "A XML processing library."
                                          URL )" -P " -- http://xmlsoft.org/

set_package_properties(LibXml2 PROPERTIES TYPE RECOMMENDED
PURPOSE "Enables HTML-import in MyWordProcessor")
...
set_package_properties(LibXml2 PROPERTIES TYPE OPTIONAL
PURPOSE "Enables odt-export in MyWordProcessor")

find_package(DBUS)
set_package_properties(DBUS PROPERTIES TYPE RUNTIME
PURPOSE "Necessary to disable the screensaver during a presentation" )

ADD_FEATURE_INFO(<name> <enabled> <description>)
```

Use this macro to add information about a feature with the given <name>. <enabled> contains whether this feature is enabled or not, <description> is a text describing the feature. The information can be displayed using feature_summary() for ENABLED_FEATURES and DISABLED_FEATURES respectively.

Example for setting the info for a feature:

```
option(WITH_FOO "Help for foo" ON)
add_feature_info(Foo WITH_FOO "The Foo feature provides very cool stuff.")
```

The following macros are provided for compatibility with previous CMake versions:

```
SET_PACKAGE_INFO(<name> <description> [<url> [<purpose>] ] )
```

Use this macro to set up information about the named package, which can then be displayed via FEATURE_SUMMARY(). This can be done either directly in the Find-module or in the project

which uses the module after the find_package() call. The features for which information can be set are added automatically by the find_package() command.

```
PRINT_ENABLED_FEATURES()
```

Does the same as FEATURE_SUMMARY(WHAT ENABLED_FEATURES DESCRIPTION Enabled features:)

```
PRINT_DISABLED_FEATURES()
```

Does the same as FEATURE_SUMMARY(WHAT DISABLED_FEATURES DESCRIPTION Disabled features:)

```
SET_FEATURE_INFO(<name> <description> [<url>] )
```

Does the same as SET_PACKAGE_INFO(<name> <description> <url>)

FindALSA

Find alsa

Find the alsa libraries (asound)

```
This module defines the following variables:
ALSA_FOUND - True if ALSA_INCLUDE_DIR & ALSA_LIBRARY are found
ALSA_LIBRARIES - Set when ALSA_LIBRARY is found
ALSA_INCLUDE_DIRS - Set when ALSA_INCLUDE_DIR is found

ALSA_INCLUDE_DIR - where to find asoundlib.h, etc.
ALSA_LIBRARY - the asound library
ALSA_VERSION_STRING - the version of alsa found (since CMake 2.8.8)
```

FindArmadillo

Find Armadillo

Find the Armadillo C++ library

Using Armadillo:

```
find_package(Armadillo REQUIRED)
include_directories(${ARMADILLO_INCLUDE_DIRS})
add_executable(foo foo.cc)
target_link_libraries(foo ${ARMADILLO_LIBRARIES})
```

This module sets the following variables:

```
ARMADILLO_FOUND - set to true if the library is found
ARMADILLO_INCLUDE_DIRS - list of required include directories
ARMADILLO_LIBRARIES - list of libraries to be linked
ARMADILLO_VERSION_MAJOR - major version number
ARMADILLO_VERSION_MINOR - minor version number
ARMADILLO_VERSION_PATCH - patch version number
ARMADILLO_VERSION_STRING - version number as a string (ex: "1.0.4")
ARMADILLO_VERSION_NAME - name of the version (ex: "Antipodean Antileech")
```

FindASPELL

Try to find ASPELL

Once done this will define

```
ASPELL_FOUND - system has ASPELL
ASPELL_EXECUTABLE - the ASPELL executable
ASPELL_INCLUDE_DIR - the ASPELL include directory
ASPELL_LIBRARIES - The libraries needed to use ASPELL
ASPELL_DEFINITIONS - Compiler switches required for using ASPELL
```

FindAVIFile

Locate AVIFILE library and include paths

AVIFILE (*http://avifile.sourceforge.net/*)*is* a set of libraries for i386 machines to use various AVI codecs. Support is limited beyond Linux. Windows provides native AVI support, and so doesnt need this library. This module defines

```
AVIFILE_INCLUDE_DIR, where to find avifile.h , etc.
AVIFILE_LIBRARIES, the libraries to link against
AVIFILE_DEFINITIONS, definitions to use when compiling
AVIFILE_FOUND, If false, don't try to use AVIFILE
```

FindBISON

Find bison executable and provides macros to generate custom build rules

The module defines the following variables:

```
BISON_EXECUTABLE - path to the bison program
BISON_VERSION - version of bison
BISON_FOUND - true if the program was found
```

The minimum required version of bison can be specified using the standard CMake syntax, e.g. find_package(BISON 2.1.3)

If bison is found, the module defines the macros:

```
BISON_TARGET(<Name> <YaccInput> <CodeOutput> [VERBOSE <file>]
[COMPILE_FLAGS <string>])
```

which will create a custom rule to generate a parser. <YaccInput> is the path to a yacc file. <CodeOutput> is the name of the source file generated by bison. A header file is also be generated, and contains the token list. If COMPILE_FLAGS option is specified, the next parameter is added in the bison command line. if VERBOSE option is specified, <file> is created and contains verbose descriptions of the grammar and parser. The macro defines a set of variables:

```
BISON_${Name}_DEFINED - true is the macro ran successfully
BISON_${Name}_INPUT - The input source file, an alias for <YaccInput>
BISON_${Name}_OUTPUT_SOURCE - The source file generated by bison
BISON_${Name}_OUTPUT_HEADER - The header file generated by bison
BISON_${Name}_OUTPUTS - The sources files generated by bison
BISON_${Name}_COMPILE_FLAGS - Options used in the bison command line

====================================================================
Example:

find_package(BISON)
BISON_TARGET(MyParser parser.y ${CMAKE_CURRENT_BINARY_DIR}/parser.cpp)
add_executable(Foo main.cpp ${BISON_MyParser_OUTPUTS})
====================================================================
```

FindBLAS

Find BLAS library

This module finds an installed fortran library that implements the BLAS linear-algebra interface (see *http://www.netlib.org/blas/*).
The list of libraries searched for is taken from the autoconf macro file, acx_blas.m4 (distributed at *http://ac-archive.sourceforge.net/ac-archive/acx_blas.html*).

This module sets the following variables:

```
BLAS_FOUND - set to true if a library implementing the BLAS interface
is found
BLAS_LINKER_FLAGS - uncached list of required linker flags (excluding -l
```

```
and -L).
BLAS_LIBRARIES - uncached list of libraries (using full path name) to
link against to use BLAS
BLAS95_LIBRARIES - uncached list of libraries (using full path name)
to link against to use BLAS95 interface
BLAS95_FOUND - set to true if a library implementing the BLAS f95 interface
is found
BLA_STATIC if set on this determines what kind of linkage we do (static)
BLA_VENDOR if set checks only the specified vendor, if not set checks
all the possibilities
BLA_F95 if set on tries to find the f95 interfaces for BLAS/LAPACK
```

######### ## List of vendors (BLA_VENDOR) valid in this module # Goto,ATLAS PhiPACK,CXML,DXML,SunPerf,SCSL,SGIMATH,IBMESSL,Intel10_32 (intel mkl v10 32 bit),Intel10_64lp (intel mkl v10 64 bit,lp thread model, lp64 model), # Intel10_64lp_seq (intel mkl v10 64 bit,sequential code, lp64 model), # Intel(older versions of mkl 32 and 64 bit), ACML,ACML_MP,ACML_GPU,Apple, NAS, Generic C/CXX should be enabled to use Intel mkl

FindBacktrace

Find provider for backtrace(3)

Checks if OS supports backtrace(3) via either libc or custom library. This module defines the following variables:

```
Backtrace_HEADER - The header file needed for backtrace(3)
Cached.
Could be forcibly set by user.
Backtrace_INCLUDE_DIRS - The include directories needed to use backtrace(3)
header.
Backtrace_LIBRARIES - The libraries (linker flags) needed to use backtrace(3),
if any.
Backtrace_FOUND - Is set if and only if backtrace(3)
support detected.
```

The following cache variables are also available to set or use:

```
Backtrace_LIBRARY - The external library providing backtrace, if any.
Backtrace_INCLUDE_DIR - The directory holding the backtrace(3)
header.
```

Typical usage is to generate of header file using configure_file() with the contents like the following:

```
#cmakedefine01 Backtrace_FOUND
#if Backtrace_FOUND
# include <${Backtrace_HEADER}>
#endif
```

And then reference that generated header file in actual source.

FindBoost

Find Boost include dirs and libraries

Use this module by invoking find_package with the form:

```
find_package(Boost
[version] [EXACT] # Minimum or EXACT version e.g. 1.36.0
[REQUIRED] # Fail with error if Boost is not found
[COMPONENTS <libs>...] # Boost libraries by their canonical name
) # e.g. "date_time" for "libboost_date_time"
```

This module finds headers and requested component libraries OR a CMake package configuration file provided by a Boost CMake build. For the latter case skip to the Boost CMake section below. For the former case results are reported in variables:

```
Boost_FOUND - True if headers and requested libraries were found
Boost_INCLUDE_DIRS - Boost include directories
Boost_LIBRARY_DIRS - Link directories for Boost libraries
Boost_LIBRARIES - Boost component libraries to be linked
Boost_<C>_FOUND - True if component <C> was found (<C> is upper-case)
Boost_<C>_LIBRARY - Libraries to link for component <C> (may include
target_link_libraries debug/optimized keywords)
Boost_VERSION - BOOST_VERSION value from boost/version.hpp
Boost_LIB_VERSION - Version string appended to library filenames
Boost_MAJOR_VERSION - Boost major version number (X in X.y.z)
Boost_MINOR_VERSION - Boost minor version number (Y in x.Y.z)
Boost_SUBMINOR_VERSION - Boost subminor version number (Z in x.y.Z)
Boost_LIB_DIAGNOSTIC_DEFINITIONS (Windows)
- Pass to add_definitions() to have diagnostic
information about Boost's automatic linking
displayed during compilation
```

This module reads hints about search locations from variables:

```
BOOST_ROOT - Preferred installation prefix
(or BOOSTROOT)
BOOST_INCLUDEDIR - Preferred include directory e.g. <prefix>/include
BOOST_LIBRARYDIR - Preferred library directory e.g. <prefix>/lib
Boost_NO_SYSTEM_PATHS - Set to ON to disable searching in locations not
specified by these hint variables. Default is OFF.
Boost_ADDITIONAL_VERSIONS
- List of Boost versions not known to this module
(Boost install locations may contain the version)
```

and saves search results persistently in CMake cache entries:

```
Boost_INCLUDE_DIR - Directory containing Boost headers
Boost_LIBRARY_DIR - Directory containing Boost libraries
Boost_<C>_LIBRARY_DEBUG - Component <C> library debug variant
Boost_<C>_LIBRARY_RELEASE - Component <C> library release variant
```

Users may set these hints or results as cache entries. Projects should not read these entries directly but instead use the above result variables. Note that some hint names start in upper-case BOOST. One may specify these as environment variables if they are not specified as CMake variables or cache entries.

This module first searches for the Boost header files using the above hint variables (excluding BOOST_LIBRARYDIR) and saves the result in Boost_INCLUDE_DIR. Then it searches for requested component libraries using the above hints (excluding BOOST_INCLUDEDIR and Boost_ADDITIONAL_VERSIONS), lib directories near Boost_INCLUDE_DIR, and the library name configuration settings below. It saves the library directory in Boost_LIBRARY_DIR and individual library locations in Boost_<C>_LIBRARY_DEBUG and Boost_<C>_LIBRARY_RELEASE. When one changes settings used by previous searches in the same build tree (excluding environment variables) this module discards previous search results affected by the changes and searches again.

Boost libraries come in many variants encoded in their file name. Users or projects may tell this module which variant to find by setting variables:

```
Boost_USE_MULTITHREADED - Set to OFF to use the non-multithreaded
```

```
libraries ('mt' tag). Default is ON.
Boost_USE_STATIC_LIBS - Set to ON to force the use of the static
libraries. Default is OFF.
Boost_USE_STATIC_RUNTIME - Set to ON or OFF to specify whether to use
libraries linked statically to the C++ runtime
('s' tag). Default is platform dependent.
Boost_USE_DEBUG_RUNTIME - Set to ON or OFF to specify whether to use
libraries linked to the MS debug C++ runtime
('g' tag). Default is ON.
Boost_USE_DEBUG_PYTHON - Set to ON to use libraries compiled with a
debug Python build ('y' tag). Default is OFF.
Boost_USE_STLPORT - Set to ON to use libraries compiled with
STLPort ('p' tag). Default is OFF.
Boost_USE_STLPORT_DEPRECATED_NATIVE_IOSTREAMS
- Set to ON to use libraries compiled with
STLPort deprecated "native iostreams"
('n' tag). Default is OFF.
Boost_COMPILER - Set to the compiler-specific library suffix
(e.g. "-gcc43"). Default is auto-computed
for the C++ compiler in use.
Boost_THREADAPI - Suffix for "thread" component library name,
such as "pthread" or "win32". Names with
and without this suffix will both be tried.
Boost_NAMESPACE - Alternate namespace used to build boost with
e.g. if set to "myboost", will search for
myboost_thread instead of boost_thread.
```

Other variables one may set to control this module are:

```
Boost_DEBUG - Set to ON to enable debug output from FindBoost.
Please enable this before filing any bug report.
Boost_DETAILED_FAILURE_MSG
- Set to ON to add detailed information to the
failure message even when the REQUIRED option
is not given to the find_package call.
Boost_REALPATH - Set to ON to resolve symlinks for discovered
libraries to assist with packaging. For example,
the "system" component library may be resolved to
"/usr/lib/libboost_system.so.1.42.0" instead of
"/usr/lib/libboost_system.so". This does not
affect linking and should not be enabled unless
the user needs this information.
```

On Visual Studio and Borland compilers Boost headers request automatic linking to corresponding libraries. This requires matching libraries to be linked explicitly or available in the link library search path. In this case setting Boost_USE_STATIC_LIBS to OFF may not achieve dynamic linking. Boost automatic linking typically requests static libraries with a few exceptions (such as Boost.Python). Use:

```
add_definitions(${Boost_LIB_DIAGNOSTIC_DEFINITIONS})
```

to ask Boost to report information about automatic linking requests.

Example to find Boost headers only:

```
find_package(Boost 1.36.0)
if(Boost_FOUND)
include_directories(${Boost_INCLUDE_DIRS})
```

```
add_executable(foo foo.cc)
endif()
```

Example to find Boost headers and some *static* libraries:

```
set(Boost_USE_STATIC_LIBS ON) # only find static libs
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.36.0 COMPONENTS date_time filesystem system ...)
if(Boost_FOUND)
include_directories(${Boost_INCLUDE_DIRS})
add_executable(foo foo.cc)
target_link_libraries(foo ${Boost_LIBRARIES})
endif()
```

Boost CMake

If Boost was built using the boost-cmake project it provides a package configuration file for use with find_packages Config mode. This module looks for the package configuration file called BoostConfig.cmake or boost-config.cmake and stores the result in cache entry Boost_DIR. If found, the package configuration file is loaded and this module returns with no further action. See documentation of the Boost CMake package configuration for details on what it provides.

Set Boost_NO_BOOST_CMAKE to ON to disable the search for boost-cmake.

FindBullet

Try to find the Bullet physics engine

```
This module defines the following variables

BULLET_FOUND - Was bullet found
BULLET_INCLUDE_DIRS - the Bullet include directories
BULLET_LIBRARIES - Link to this, by default it includes
all bullet components (Dynamics,
Collision, LinearMath, & SoftBody)

This module accepts the following variables

BULLET_ROOT - Can be set to bullet install path or Windows build path
```

FindBZip2

Try to find BZip2

Once done this will define

```
BZIP2_FOUND - system has BZip2
BZIP2_INCLUDE_DIR - the BZip2 include directory
BZIP2_LIBRARIES - Link these to use BZip2
BZIP2_NEED_PREFIX - this is set if the functions are prefixed with BZ2_
BZIP2_VERSION_STRING - the version of BZip2 found (since CMake 2.8.8)
```

FindCABLE

Find CABLE

This module finds if CABLE is installed and determines where the include files and libraries are. This code sets the following variables:

```
CABLE the path to the cable executable
CABLE_TCL_LIBRARY the path to the Tcl wrapper library
CABLE_INCLUDE_DIR the path to the include directory
```

To build Tcl wrappers, you should add shared library and link it to ${CABLE_TCL_LIBRARY}. You should also add ${CABLE_INCLUDE_DIR} as an include directory.

FindCoin3D

Find Coin3D (Open Inventor)

Coin3D is an implementation of the Open Inventor API. It provides data structures and algorithms for 3D visualization *http://www.coin3d.org/*

This module defines the following variables

```
COIN3D_FOUND - system has Coin3D - Open Inventor
COIN3D_INCLUDE_DIRS - where the Inventor include directory can be found
COIN3D_LIBRARIES - Link to this to use Coin3D
```

FindCUDA

Tools for building CUDA C files: libraries and build dependencies.

This script locates the NVIDIA CUDA C tools. It should work on linux, windows, and mac and should be reasonably up to date with CUDA C releases.

This script makes use of the standard find_package arguments of <VERSION>, REQUIRED and QUIET. CUDA_FOUND will report if an acceptable version of CUDA was found.

The script will prompt the user to specify CUDA_TOOLKIT_ROOT_DIR if the prefix cannot be determined by the location of nvcc in the system path and REQUIRED is specified to find_package(). To use a different installed version of the toolkit set the environment variable CUDA_BIN_PATH before running cmake (e.g. CUDA_BIN_PATH=/usr/local/cuda1.0 instead of the default /usr/local/cuda) or set CUDA_TOOLKIT_ROOT_DIR after configuring. If you change the value of CUDA_TOOLKIT_ROOT_DIR, various components that depend on the path will be relocated.

It might be necessary to set CUDA_TOOLKIT_ROOT_DIR manually on certain platforms, or to use a cuda runtime not installed in the default location. In newer versions of the toolkit the cuda library is included with the graphics driver- be sure that the driver version matches what is needed by the cuda runtime version.

The following variables affect the behavior of the macros in the script (in alphebetical order). Note that any of these flags can be changed multiple times in the same directory before calling CUDA_ADD_EXECUTABLE, CUDA_ADD_LIBRARY, CUDA_COMPILE, CUDA_COMPILE_PTX or CUDA_WRAP_SRCS:

```
CUDA_64_BIT_DEVICE_CODE (Default matches host bit size)
-- Set to ON to compile for 64 bit device code, OFF for 32 bit device code.
Note that making this different from the host code when generating object
or C files from CUDA code just won't work, because size_t gets defined by
nvcc in the generated source. If you compile to PTX and then load the
file yourself, you can mix bit sizes between device and host.
```

```
CUDA_ATTACH_VS_BUILD_RULE_TO_CUDA_FILE (Default ON)
-- Set to ON if you want the custom build rule to be attached to the source
file in Visual Studio. Turn OFF if you add the same cuda file to multiple
targets.
```

```
This allows the user to build the target from the CUDA file; however, bad
things can happen if the CUDA source file is added to multiple targets.
When performing parallel builds it is possible for the custom build
command to be run more than once and in parallel causing cryptic build
errors. VS runs the rules for every source file in the target, and a
source can have only one rule no matter how many projects it is added to.
When the rule is run from multiple targets race conditions can occur on
the generated file. Eventually everything will get built, but if the user
is unaware of this behavior, there may be confusion. It would be nice if
this script could detect the reuse of source files across multiple targets
```

```
and turn the option off for the user, but no good solution could be found.

CUDA_BUILD_CUBIN (Default OFF)
-- Set to ON to enable and extra compilation pass with the -cubin option in
Device mode. The output is parsed and register, shared memory usage is
printed during build.

CUDA_BUILD_EMULATION (Default OFF for device mode)
-- Set to ON for Emulation mode. -D_DEVICEEMU is defined for CUDA C files
when CUDA_BUILD_EMULATION is TRUE.

CUDA_GENERATED_OUTPUT_DIR (Default CMAKE_CURRENT_BINARY_DIR)
-- Set to the path you wish to have the generated files placed. If it is
blank output files will be placed in CMAKE_CURRENT_BINARY_DIR.
Intermediate files will always be placed in
CMAKE_CURRENT_BINARY_DIR/CMakeFiles.

CUDA_HOST_COMPILATION_CPP (Default ON)
-- Set to OFF for C compilation of host code.

CUDA_HOST_COMPILER (Default CMAKE_C_COMPILER, $(VCInstallDir)/bin for VS)
-- Set the host compiler to be used by nvcc. Ignored if -ccbin or
--compiler-bindir is already present in the CUDA_NVCC_FLAGS or
CUDA_NVCC_FLAGS_<CONFIG> variables. For Visual Studio targets
$(VCInstallDir)/bin is a special value that expands out to the path when
the command is run from withing VS.

CUDA_NVCC_FLAGS
CUDA_NVCC_FLAGS_<CONFIG>
-- Additional NVCC command line arguments. NOTE: multiple arguments must be
semi-colon delimited (e.g. --compiler-options;-Wall)

CUDA_PROPAGATE_HOST_FLAGS (Default ON)
-- Set to ON to propagate CMAKE_{C,CXX}_FLAGS and their configuration
dependent counterparts (e.g. CMAKE_C_FLAGS_DEBUG) automatically to the
host compiler through nvcc's -Xcompiler flag. This helps make the
generated host code match the rest of the system better. Sometimes
certain flags give nvcc problems, and this will help you turn the flag
propagation off. This does not affect the flags supplied directly to nvcc
via CUDA_NVCC_FLAGS or through the OPTION flags specified through
CUDA_ADD_LIBRARY, CUDA_ADD_EXECUTABLE, or CUDA_WRAP_SRCS. Flags used for
shared library compilation are not affected by this flag.

CUDA_SEPARABLE_COMPILATION (Default OFF)
-- If set this will enable separable compilation for all CUDA runtime object
files. If used outside of CUDA_ADD_EXECUTABLE and CUDA_ADD_LIBRARY
(e.g. calling CUDA_WRAP_SRCS directly),
CUDA_COMPUTE_SEPARABLE_COMPILATION_OBJECT_FILE_NAME and
CUDA_LINK_SEPARABLE_COMPILATION_OBJECTS should be called.

CUDA_VERBOSE_BUILD (Default OFF)
-- Set to ON to see all the commands used when building the CUDA file. When
using a Makefile generator the value defaults to VERBOSE (run make
VERBOSE=1 to see output), although setting CUDA_VERBOSE_BUILD to ON will
always print the output.
```

The script creates the following macros (in alphebetical order):

```
CUDA_ADD_CUFFT_TO_TARGET( cuda_target )
-- Adds the cufft library to the target (can be any target). Handles whether
```

```
you are in emulation mode or not.

CUDA_ADD_CUBLAS_TO_TARGET( cuda_target )
-- Adds the cublas library to the target (can be any target). Handles
whether you are in emulation mode or not.

CUDA_ADD_EXECUTABLE( cuda_target file0 file1 ...
[WIN32] [MACOSX_BUNDLE] [EXCLUDE_FROM_ALL] [OPTIONS ...] )
-- Creates an executable "cuda_target" which is made up of the files
specified. All of the non CUDA C files are compiled using the standard
build rules specified by CMAKE and the cuda files are compiled to object
files using nvcc and the host compiler. In addition CUDA_INCLUDE_DIRS is
added automatically to include_directories(). Some standard CMake target
calls can be used on the target after calling this macro
(e.g. set_target_properties and target_link_libraries), but setting
properties that adjust compilation flags will not affect code compiled by
nvcc. Such flags should be modified before calling CUDA_ADD_EXECUTABLE,
CUDA_ADD_LIBRARY or CUDA_WRAP_SRCS.

CUDA_ADD_LIBRARY( cuda_target file0 file1 ...
[STATIC | SHARED | MODULE] [EXCLUDE_FROM_ALL] [OPTIONS ...] )
-- Same as CUDA_ADD_EXECUTABLE except that a library is created.

CUDA_BUILD_CLEAN_TARGET()
-- Creates a convience target that deletes all the dependency files
generated. You should make clean after running this target to ensure the
dependency files get regenerated.

CUDA_COMPILE( generated_files file0 file1 ... [STATIC | SHARED | MODULE]
[OPTIONS ...] )
-- Returns a list of generated files from the input source files to be used
with ADD_LIBRARY or ADD_EXECUTABLE.

CUDA_COMPILE_PTX( generated_files file0 file1 ... [OPTIONS ...] )
-- Returns a list of PTX files generated from the input source files.

CUDA_COMPUTE_SEPARABLE_COMPILATION_OBJECT_FILE_NAME( output_file_var
cuda_target
object_files )
-- Compute the name of the intermediate link file used for separable
compilation. This file name is typically passed into
CUDA_LINK_SEPARABLE_COMPILATION_OBJECTS. output_file_var is produced
based on cuda_target the list of objects files that need separable
compilation as specified by object_files. If the object_files list is
empty, then output_file_var will be empty. This function is called
automatically for CUDA_ADD_LIBRARY and CUDA_ADD_EXECUTABLE. Note that
this is a function and not a macro.

CUDA_INCLUDE_DIRECTORIES( path0 path1 ... )
-- Sets the directories that should be passed to nvcc
(e.g. nvcc -Ipath0 -Ipath1 ... ). These paths usually contain other .cu
files.

CUDA_LINK_SEPARABLE_COMPILATION_OBJECTS( output_file_var cuda_target
nvcc_flags object_files)

-- Generates the link object required by separable compilation from the given
object files. This is called automatically for CUDA_ADD_EXECUTABLE and
CUDA_ADD_LIBRARY, but can be called manually when using CUDA_WRAP_SRCS
```

directly. When called from CUDA_ADD_LIBRARY or CUDA_ADD_EXECUTABLE the nvcc_flags passed in are the same as the flags passed in via the OPTIONS argument. The only nvcc flag added automatically is the bitness flag as specified by CUDA_64_BIT_DEVICE_CODE. Note that this is a function instead of a macro.

CUDA_WRAP_SRCS (cuda_target format generated_files file0 file1 ... [STATIC | SHARED | MODULE] [OPTIONS ...])
-- This is where all the magic happens. CUDA_ADD_EXECUTABLE, CUDA_ADD_LIBRARY, CUDA_COMPILE, and CUDA_COMPILE_PTX all call this function under the hood.

Given the list of files (file0 file1 ... fileN) this macro generates custom commands that generate either PTX or linkable objects (use "PTX" or "OBJ" for the format argument to switch). Files that don't end with .cu or have the HEADER_FILE_ONLY property are ignored.

The arguments passed in after OPTIONS are extra command line options to give to nvcc. You can also specify per configuration options by specifying the name of the configuration followed by the options. General options must preceed configuration specific options. Not all configurations need to be specified, only the ones provided will be used.

OPTIONS -DFLAG=2 "-DFLAG_OTHER=space in flag"
DEBUG -g
RELEASE --use_fast_math
RELWITHDEBINFO --use_fast_math;-g
MINSIZEREL --use_fast_math

For certain configurations (namely VS generating object files with CUDA_ATTACH_VS_BUILD_RULE_TO_CUDA_FILE set to ON), no generated file will be produced for the given cuda file. This is because when you add the cuda file to Visual Studio it knows that this file produces an object file and will link in the resulting object file automatically.

This script will also generate a separate cmake script that is used at build time to invoke nvcc. This is for several reasons.

1. nvcc can return negative numbers as return values which confuses Visual Studio into thinking that the command succeeded. The script now checks the error codes and produces errors when there was a problem.

2. nvcc has been known to not delete incomplete results when it encounters problems. This confuses build systems into thinking the target was generated when in fact an unusable file exists. The script now deletes the output files if there was an error.

3. By putting all the options that affect the build into a file and then make the build rule dependent on the file, the output files will be regenerated when the options change.

This script also looks at optional arguments STATIC, SHARED, or MODULE to determine when to target the object compilation for a shared library. BUILD_SHARED_LIBS is ignored in CUDA_WRAP_SRCS, but it is respected in CUDA_ADD_LIBRARY. On some systems special flags are added for building objects intended for shared libraries. A preprocessor macro, <target_name>_EXPORTS is defined when a shared library compilation is detected.

Flags passed into add_definitions with -D or /D are passed along to nvcc.

The script defines the following variables:

```
CUDA_VERSION_MAJOR -- The major version of cuda as reported by nvcc.
CUDA_VERSION_MINOR -- The minor version.
CUDA_VERSION
CUDA_VERSION_STRING -- CUDA_VERSION_MAJOR.CUDA_VERSION_MINOR

CUDA_TOOLKIT_ROOT_DIR -- Path to the CUDA Toolkit (defined if not set).
CUDA_SDK_ROOT_DIR -- Path to the CUDA SDK. Use this to find files in the
SDK. This script will not directly support finding
specific libraries or headers, as that isn't
supported by NVIDIA. If you want to change
libraries when the path changes see the
FindCUDA.cmake script for an example of how to clear
these variables. There are also examples of how to
use the CUDA_SDK_ROOT_DIR to locate headers or
libraries, if you so choose (at your own risk).
CUDA_INCLUDE_DIRS -- Include directory for cuda headers. Added automatically
for CUDA_ADD_EXECUTABLE and CUDA_ADD_LIBRARY.
CUDA_LIBRARIES -- Cuda RT library.
CUDA_CUFFT_LIBRARIES -- Device or emulation library for the Cuda FFT
implementation (alternative to:
CUDA_ADD_CUFFT_TO_TARGET macro)
CUDA_CUBLAS_LIBRARIES -- Device or emulation library for the Cuda BLAS
implementation (alterative to:
CUDA_ADD_CUBLAS_TO_TARGET macro).
CUDA_cupti_LIBRARY -- CUDA Profiling Tools Interface library.
Only available for CUDA version 4.0+.
CUDA_curand_LIBRARY -- CUDA Random Number Generation library.
Only available for CUDA version 3.2+.
CUDA_cusparse_LIBRARY -- CUDA Sparse Matrix library.
Only available for CUDA version 3.2+.
CUDA_npp_LIBRARY -- NVIDIA Performance Primitives library.
Only available for CUDA version 4.0+.
CUDA_nppc_LIBRARY -- NVIDIA Performance Primitives library (core).
Only available for CUDA version 5.5+.
CUDA_nppi_LIBRARY -- NVIDIA Performance Primitives library (image processing).
Only available for CUDA version 5.5+.
CUDA_npps_LIBRARY -- NVIDIA Performance Primitives library (signal processing).
Only available for CUDA version 5.5+.
CUDA_nvcuvenc_LIBRARY -- CUDA Video Encoder library.
Only available for CUDA version 3.2+.
Windows only.
CUDA_nvcuvid_LIBRARY -- CUDA Video Decoder library.
Only available for CUDA version 3.2+.
Windows only.
```

FindCups

Try to find the Cups printing system

Once done this will define

```
CUPS_FOUND - system has Cups
CUPS_INCLUDE_DIR - the Cups include directory
CUPS_LIBRARIES - Libraries needed to use Cups
CUPS_VERSION_STRING - version of Cups found (since CMake 2.8.8)
Set CUPS_REQUIRE_IPP_DELETE_ATTRIBUTE to TRUE if you need a version which
```

```
features this function (i.e. at least 1.1.19)
```

FindCURL

Find curl

Find the native CURL headers and libraries.

```
CURL_INCLUDE_DIRS - where to find curl/curl.h, etc.
CURL_LIBRARIES - List of libraries when using curl.
CURL_FOUND - True if curl found.
CURL_VERSION_STRING - the version of curl found (since CMake 2.8.8)
```

FindCurses

Find the curses include file and library

```
CURSES_FOUND - system has Curses
CURSES_INCLUDE_DIR - the Curses include directory
CURSES_LIBRARIES - The libraries needed to use Curses
CURSES_HAVE_CURSES_H - true if curses.h is available
CURSES_HAVE_NCURSES_H - true if ncurses.h is available
CURSES_HAVE_NCURSES_NCURSES_H - true if ncurses/ncurses.h is available
CURSES_HAVE_NCURSES_CURSES_H - true if ncurses/curses.h is available
CURSES_LIBRARY - set for backwards compatibility with 2.4 CMake
```

Set CURSES_NEED_NCURSES to TRUE before the find_package() command if NCurses functionality is required.

FindCVS

The module defines the following variables:

```
CVS_EXECUTABLE - path to cvs command line client
CVS_FOUND - true if the command line client was found
```

Example usage:

```
find_package(CVS)
if(CVS_FOUND)
message("CVS found: ${CVS_EXECUTABLE}")
endif()
```

FindCxxTest

Find CxxTest

Find the CxxTest suite and declare a helper macro for creating unit tests and integrating them with CTest. For more details on CxxTest see *http://cxxtest.tigris.org*

INPUT Variables

```
CXXTEST_USE_PYTHON [deprecated since 1.3]
Only used in the case both Python & Perl
are detected on the system to control
which CxxTest code generator is used.
Valid only for CxxTest version 3.
```

```
NOTE: In older versions of this Find Module,
this variable controlled if the Python test
generator was used instead of the Perl one,
regardless of which scripting language the
user had installed.
```

```
CXXTEST_TESTGEN_ARGS (since CMake 2.8.3)
Specify a list of options to pass to the CxxTest code
generator. If not defined, --error-printer is
```

```
passed.
```

OUTPUT Variables

```
CXXTEST_FOUND
True if the CxxTest framework was found
CXXTEST_INCLUDE_DIRS
Where to find the CxxTest include directory
CXXTEST_PERL_TESTGEN_EXECUTABLE
The perl-based test generator
CXXTEST_PYTHON_TESTGEN_EXECUTABLE
The python-based test generator
CXXTEST_TESTGEN_EXECUTABLE (since CMake 2.8.3)
The test generator that is actually used (chosen using user preferences
and interpreters found in the system)
CXXTEST_TESTGEN_INTERPRETER (since CMake 2.8.3)
The full path to the Perl or Python executable on the system
```

MACROS for optional use by CMake users:

```
CXXTEST_ADD_TEST(<test_name> <gen_source_file> <input_files_to_testgen...>)
Creates a CxxTest runner and adds it to the CTest testing suite
Parameters:
test_name The name of the test
gen_source_file The generated source filename to be
generated by CxxTest
input_files_to_testgen The list of header files containing the
CxxTest::TestSuite's to be included in
this runner

#==============
Example Usage:

find_package(CxxTest)
if(CXXTEST_FOUND)
include_directories(${CXXTEST_INCLUDE_DIR})
enable_testing()

CXXTEST_ADD_TEST(unittest_foo foo_test.cc
${CMAKE_CURRENT_SOURCE_DIR}/foo_test.h)
target_link_libraries(unittest_foo foo) # as needed
endif()

This will (if CxxTest is found):
1. Invoke the testgen executable to autogenerate foo_test.cc in the
binary tree from "foo_test.h" in the current source directory.
2. Create an executable and test called unittest_foo.

#=============
Example foo_test.h:

#include <cxxtest/TestSuite.h>

class MyTestSuite : public CxxTest::TestSuite
{
public:
void testAddition( void )
{
TS_ASSERT( 1 + 1 > 1 );
TS_ASSERT_EQUALS( 1 + 1, 2 );
```

```
}
};
```

FindCygwin
this module looks for Cygwin

FindDart
Find DART

This module looks for the dart testing software and sets DART_ROOT to point to where it found it.

FindDCMTK
find DCMTK libraries and applications

FindDevIL
This module locates the developers image library. *http://openil.sourceforge.net/*

This module sets:

```
IL_LIBRARIES - the name of the IL library. These include the full path to
the core DevIL library. This one has to be linked into the
application.
ILU_LIBRARIES - the name of the ILU library. Again, the full path. This
library is for filters and effects, not actual loading. It
doesn't have to be linked if the functionality it provides
is not used.
ILUT_LIBRARIES - the name of the ILUT library. Full path. This part of the
library interfaces with OpenGL. It is not strictly needed
in applications.
IL_INCLUDE_DIR - where to find the il.h, ilu.h and ilut.h files.
IL_FOUND - this is set to TRUE if all the above variables were set.
This will be set to false if ILU or ILUT are not found,
even if they are not needed. In most systems, if one
library is found all the others are as well. That's the
way the DevIL developers release it.
```

FindDoxygen
This module looks for Doxygen and the path to Graphvizs dot

Doxygen is a documentation generation tool. Please see *http://www.doxygen.org*

This module accepts the following optional variables:

```
DOXYGEN_SKIP_DOT = If true this module will skip trying to find Dot
(an optional component often used by Doxygen)
```

This modules defines the following variables:

```
DOXYGEN_EXECUTABLE = The path to the doxygen command.
DOXYGEN_FOUND = Was Doxygen found or not?
DOXYGEN_VERSION = The version reported by doxygen --version
```

```
DOXYGEN_DOT_EXECUTABLE = The path to the dot program used by doxygen.
DOXYGEN_DOT_FOUND = Was Dot found or not?
DOXYGEN_DOT_PATH = The path to dot not including the executable
```

FindEXPAT
Find expat

Find the native EXPAT headers and libraries.

```
EXPAT_INCLUDE_DIRS - where to find expat.h, etc.
EXPAT_LIBRARIES - List of libraries when using expat.
```

```
EXPAT_FOUND - True if expat found.
```

FindFLEX

Find flex executable and provides a macro to generate custom build rules

The module defines the following variables:

```
FLEX_FOUND - true is flex executable is found
FLEX_EXECUTABLE - the path to the flex executable
FLEX_VERSION - the version of flex
FLEX_LIBRARIES - The flex libraries
FLEX_INCLUDE_DIRS - The path to the flex headers
```

The minimum required version of flex can be specified using the standard syntax, e.g. find_package(FLEX 2.5.13)

If flex is found on the system, the module provides the macro:

```
FLEX_TARGET(Name FlexInput FlexOutput [COMPILE_FLAGS <string>])
```

which creates a custom command to generate the <FlexOutput> file from the <FlexInput> file. If COMPILE_FLAGS option is specified, the next parameter is added to the flex command line. Name is an alias used to get details of this custom command. Indeed the macro defines the following variables:

```
FLEX_${Name}_DEFINED - true is the macro ran successfully
FLEX_${Name}_OUTPUTS - the source file generated by the custom rule, an
alias for FlexOutput
FLEX_${Name}_INPUT - the flex source file, an alias for ${FlexInput}
```

Flex scanners oftenly use tokens defined by Bison: the code generated by Flex depends of the header generated by Bison. This module also defines a macro:

```
ADD_FLEX_BISON_DEPENDENCY(FlexTarget BisonTarget)
```

which adds the required dependency between a scanner and a parser where <FlexTarget> and <BisonTarget> are the first parameters of respectively FLEX_TARGET and BISON_TARGET macros.

```
=====================================================================
Example:

find_package(BISON)
find_package(FLEX)

BISON_TARGET(MyParser parser.y ${CMAKE_CURRENT_BINARY_DIR}/parser.cpp)
FLEX_TARGET(MyScanner lexer.l ${CMAKE_CURRENT_BINARY_DIR}/lexer.cpp)
ADD_FLEX_BISON_DEPENDENCY(MyScanner MyParser)

include_directories(${CMAKE_CURRENT_BINARY_DIR})
add_executable(Foo
Foo.cc
${BISON_MyParser_OUTPUTS}
${FLEX_MyScanner_OUTPUTS}
)
=====================================================================
```

FindFLTK2

Find the native FLTK2 includes and library

The following settings are defined

```
FLTK2_FLUID_EXECUTABLE, where to find the Fluid tool
FLTK2_WRAP_UI, This enables the FLTK2_WRAP_UI command
```

```
FLTK2_INCLUDE_DIR, where to find include files
FLTK2_LIBRARIES, list of fltk2 libraries
FLTK2_FOUND, Don't use FLTK2 if false.
```

The following settings should not be used in general.

```
FLTK2_BASE_LIBRARY = the full path to fltk2.lib
FLTK2_GL_LIBRARY = the full path to fltk2_gl.lib
FLTK2_IMAGES_LIBRARY = the full path to fltk2_images.lib
```

FindFLTK

Find the native FLTK includes and library

By default FindFLTK.cmake will search for all of the FLTK components and add them to the FLTK_LIBRARIES variable.

```
You can limit the components which get placed in FLTK_LIBRARIES by
defining one or more of the following three options:
```

```
FLTK_SKIP_OPENGL, set to true to disable searching for opengl and
the FLTK GL library
FLTK_SKIP_FORMS, set to true to disable searching for fltk_forms
FLTK_SKIP_IMAGES, set to true to disable searching for fltk_images
```

```
FLTK_SKIP_FLUID, set to true if the fluid binary need not be present
at build time
```

The following variables will be defined:

```
FLTK_FOUND, True if all components not skipped were found
FLTK_INCLUDE_DIR, where to find include files
FLTK_LIBRARIES, list of fltk libraries you should link against
FLTK_FLUID_EXECUTABLE, where to find the Fluid tool
FLTK_WRAP_UI, This enables the FLTK_WRAP_UI command
```

The following cache variables are assigned but should not be used. See the FLTK_LIBRARIES variable instead.

```
FLTK_BASE_LIBRARY = the full path to fltk.lib
FLTK_GL_LIBRARY = the full path to fltk_gl.lib
FLTK_FORMS_LIBRARY = the full path to fltk_forms.lib
FLTK_IMAGES_LIBRARY = the full path to fltk_images.lib
```

FindFreetype

Locate FreeType library

This module defines

```
FREETYPE_LIBRARIES, the library to link against
FREETYPE_FOUND, if false, do not try to link to FREETYPE
FREETYPE_INCLUDE_DIRS, where to find headers.
FREETYPE_VERSION_STRING, the version of freetype found (since CMake 2.8.8)
This is the concatenation of the paths:
FREETYPE_INCLUDE_DIR_ft2build
FREETYPE_INCLUDE_DIR_freetype2
```

$FREETYPE_DIR is an environment variable that would correspond to the

FindGCCXML

Find the GCC-XML front-end executable.

This module will define the following variables:

```
GCCXML - the GCC-XML front-end executable.
```

FindGDAL

Locate gdal

This module accepts the following environment variables:

```
GDAL_DIR or GDAL_ROOT - Specify the location of GDAL
```

This module defines the following CMake variables:

```
GDAL_FOUND - True if libgdal is found
GDAL_LIBRARY - A variable pointing to the GDAL library
GDAL_INCLUDE_DIR - Where to find the headers
```

FindGettext

Find GNU gettext tools

This module looks for the GNU gettext tools. This module defines the following values:

```
GETTEXT_MSGMERGE_EXECUTABLE: the full path to the msgmerge tool.
GETTEXT_MSGFMT_EXECUTABLE: the full path to the msgfmt tool.
GETTEXT_FOUND: True if gettext has been found.
GETTEXT_VERSION_STRING: the version of gettext found (since CMake 2.8.8)
```

Additionally it provides the following macros: GETTEXT_CREATE_TRANSLATIONS (outputFile [ALL] file1 ... fileN)

```
This will create a target "translations" which will convert the
given input po files into the binary output mo file. If the
ALL option is used, the translations will also be created when
building the default target.
```

GETTEXT_PROCESS_POT(<potfile> [ALL] [INSTALL_DESTINATION <destdir>] LANGUAGES <lang1> <lang2> ...)

```
Process the given pot file to mo files.
If INSTALL_DESTINATION is given then automatically install rules will be created,
the language subdirectory will be taken into account (by default use share/locale/).
If ALL is specified, the pot file is processed when building the all traget.
It creates a custom target "potfile".
```

GETTEXT_PROCESS_PO_FILES(<lang> [ALL] [INSTALL_DESTINATION <dir>] PO_FILES <po1> <po2> ...)

```
Process the given po files to mo files for the given language.
If INSTALL_DESTINATION is given then automatically install rules will be created,
the language subdirectory will be taken into account (by default use share/locale/).
If ALL is specified, the po files are processed when building the all traget.
It creates a custom target "pofiles".
```

FindGIF

This module searches giflib and defines GIF_LIBRARIES - libraries to link to in order to use GIF GIF_FOUND, if false, do not try to link GIF_INCLUDE_DIR, where to find the headers GIF_VERSION, reports either version 4 or 3 (for everything before version 4)

The minimum required version of giflib can be specified using the standard syntax, e.g. find_package(GIF 4)

$GIF_DIR is an environment variable that would correspond to the

FindGit

The module defines the following variables:

```
GIT_EXECUTABLE - path to git command line client
GIT_FOUND - true if the command line client was found
```

```
GIT_VERSION_STRING - the version of git found (since CMake 2.8.8)
```

Example usage:

```
find_package(Git)
if(GIT_FOUND)
message("git found: ${GIT_EXECUTABLE}")
endif()
```

FindGLEW

Find the OpenGL Extension Wrangler Library (GLEW)

This module defines the following variables:

```
GLEW_INCLUDE_DIRS - include directories for GLEW
GLEW_LIBRARIES - libraries to link against GLEW
GLEW_FOUND - true if GLEW has been found and can be used
```

FindGLUT

try to find glut library and include files

```
GLUT_INCLUDE_DIR, where to find GL/glut.h, etc.
GLUT_LIBRARIES, the libraries to link against
GLUT_FOUND, If false, do not try to use GLUT.
```

Also defined, but not for general use are:

```
GLUT_glut_LIBRARY = the full path to the glut library.
GLUT_Xmu_LIBRARY = the full path to the Xmu library.
GLUT_Xi_LIBRARY = the full path to the Xi Library.
```

FindGnuplot

this module looks for gnuplot

Once done this will define

```
GNUPLOT_FOUND - system has Gnuplot
GNUPLOT_EXECUTABLE - the Gnuplot executable
GNUPLOT_VERSION_STRING - the version of Gnuplot found (since CMake 2.8.8)
```

GNUPLOT_VERSION_STRING will not work for old versions like 3.7.1.

FindGnuTLS

Try to find the GNU Transport Layer Security library (gnutls)

Once done this will define

```
GNUTLS_FOUND - System has gnutls
GNUTLS_INCLUDE_DIR - The gnutls include directory
GNUTLS_LIBRARIES - The libraries needed to use gnutls
GNUTLS_DEFINITIONS - Compiler switches required for using gnutls
```

FindGTest

Locate the Google C++ Testing Framework.

Defines the following variables:

```
GTEST_FOUND - Found the Google Testing framework
GTEST_INCLUDE_DIRS - Include directories
```

Also defines the library variables below as normal variables. These contain debug/optimized keywords when a debugging library is found.

```
GTEST_BOTH_LIBRARIES - Both libgtest & libgtest-main
GTEST_LIBRARIES - libgtest
```

```
GTEST_MAIN_LIBRARIES - libgtest-main
```

Accepts the following variables as input:

```
GTEST_ROOT - (as a CMake or environment variable)
The root directory of the gtest install prefix

GTEST_MSVC_SEARCH - If compiling with MSVC, this variable can be set to
"MD" or "MT" to enable searching a GTest build tree
(defaults: "MD")
```

Example Usage:

```
enable_testing()
find_package(GTest REQUIRED)
include_directories(${GTEST_INCLUDE_DIRS})

add_executable(foo foo.cc)
target_link_libraries(foo ${GTEST_BOTH_LIBRARIES})

add_test(AllTestsInFoo foo)
```

If you would like each Google test to show up in CTest as a test you may use the following macro. NOTE: It will slow down your tests by running an executable for each test and test fixture. You will also have to rerun CMake after adding or removing tests or test fixtures.

GTEST_ADD_TESTS(executable extra_args ARGN)

```
executable = The path to the test executable
extra_args = Pass a list of extra arguments to be passed to
executable enclosed in quotes (or "" for none)
ARGN = A list of source files to search for tests & test
fixtures.

Example:
set(FooTestArgs --foo 1 --bar 2)
add_executable(FooTest FooUnitTest.cc)
GTEST_ADD_TESTS(FooTest "${FooTestArgs}" FooUnitTest.cc)
```

FindGTK2

FindGTK2.cmake

This module can find the GTK2 widget libraries and several of its other optional components like gtkmm, glade, and glademm.

NOTE: If you intend to use version checking, CMake 2.6.2 or later is

```
required.
```

Specify one or more of the following components as you call this find module. See example below.

```
gtk
gtkmm
glade
glademm
```

The following variables will be defined for your use

```
GTK2_FOUND - Were all of your specified components found?
GTK2_INCLUDE_DIRS - All include directories
GTK2_LIBRARIES - All libraries
GTK2_DEFINITIONS - Additional compiler flags

GTK2_VERSION - The version of GTK2 found (x.y.z)
GTK2_MAJOR_VERSION - The major version of GTK2
```

```
GTK2_MINOR_VERSION - The minor version of GTK2
GTK2_PATCH_VERSION - The patch version of GTK2
```

Optional variables you can define prior to calling this module:

```
GTK2_DEBUG - Enables verbose debugging of the module
GTK2_ADDITIONAL_SUFFIXES - Allows defining additional directories to
search for include files
```

========== Example Usage:

```
Call find_package() once, here are some examples to pick from:

Require GTK 2.6 or later
find_package(GTK2 2.6 REQUIRED gtk)

Require GTK 2.10 or later and Glade
find_package(GTK2 2.10 REQUIRED gtk glade)

Search for GTK/GTKMM 2.8 or later
find_package(GTK2 2.8 COMPONENTS gtk gtkmm)

if(GTK2_FOUND)
include_directories(${GTK2_INCLUDE_DIRS})
add_executable(mygui mygui.cc)
target_link_libraries(mygui ${GTK2_LIBRARIES})
endif()
```

FindGTK

try to find GTK (and glib) and GTKGLArea

```
GTK_INCLUDE_DIR - Directories to include to use GTK
GTK_LIBRARIES - Files to link against to use GTK
GTK_FOUND - GTK was found
GTK_GL_FOUND - GTK's GL features were found
```

FindHDF5

Find HDF5, a library for reading and writing self describing array data.

This module invokes the HDF5 wrapper compiler that should be installed alongside HDF5. Depending upon the HDF5 Configuration, the wrapper compiler is called either h5cc or h5pcc. If this succeeds, the module will then call the compiler with the -show argument to see what flags are used when compiling an HDF5 client application.

The module will optionally accept the COMPONENTS argument. If no COMPONENTS are specified, then the find module will default to finding only the HDF5 C library. If one or more COMPONENTS are specified, the module will attempt to find the language bindings for the specified components. The only valid components are C, CXX, Fortran, HL, and Fortran_HL. If the COMPONENTS argument is not given, the module will attempt to find only the C bindings.

On UNIX systems, this module will read the variable HDF5_USE_STATIC_LIBRARIES to determine whether or not to prefer a static link to a dynamic link for HDF5 and all of its dependencies. To use this feature, make sure that the HDF5_USE_STATIC_LIBRARIES variable is set before the call to find_package.

To provide the module with a hint about where to find your HDF5 installation, you can set the environment variable HDF5_ROOT. The Find module will then look in this path when searching for HDF5 executables, paths, and libraries.

In addition to finding the includes and libraries required to compile an HDF5 client application, this module also makes an effort to find tools that come with the HDF5 distribution that may be useful for regression testing.

This module will define the following variables:

```
HDF5_INCLUDE_DIRS - Location of the hdf5 includes
HDF5_INCLUDE_DIR - Location of the hdf5 includes (deprecated)
HDF5_DEFINITIONS - Required compiler definitions for HDF5
HDF5_C_LIBRARIES - Required libraries for the HDF5 C bindings.
HDF5_CXX_LIBRARIES - Required libraries for the HDF5 C++ bindings
HDF5_Fortran_LIBRARIES - Required libraries for the HDF5 Fortran bindings
HDF5_HL_LIBRARIES - Required libraries for the HDF5 high level API
HDF5_Fortran_HL_LIBRARIES - Required libraries for the high level Fortran
bindings.
HDF5_LIBRARIES - Required libraries for all requested bindings
HDF5_FOUND - true if HDF5 was found on the system
HDF5_LIBRARY_DIRS - the full set of library directories
HDF5_IS_PARALLEL - Whether or not HDF5 was found with parallel IO support
HDF5_C_COMPILER_EXECUTABLE - the path to the HDF5 C wrapper compiler
HDF5_CXX_COMPILER_EXECUTABLE - the path to the HDF5 C++ wrapper compiler
HDF5_Fortran_COMPILER_EXECUTABLE - the path to the HDF5 Fortran wrapper compiler
HDF5_DIFF_EXECUTABLE - the path to the HDF5 dataset comparison tool
```

FindHg

The module defines the following variables:

```
HG_EXECUTABLE - path to mercurial command line client (hg)
HG_FOUND - true if the command line client was found
HG_VERSION_STRING - the version of mercurial found
```

Example usage:

```
find_package(Hg)
if(HG_FOUND)
message("hg found: ${HG_EXECUTABLE}")
endif()
```

FindHSPELL

Try to find Hspell

Once done this will define

```
HSPELL_FOUND - system has Hspell
HSPELL_INCLUDE_DIR - the Hspell include directory
HSPELL_LIBRARIES - The libraries needed to use Hspell
HSPELL_DEFINITIONS - Compiler switches required for using Hspell

HSPELL_VERSION_STRING - The version of Hspell found (x.y)
HSPELL_MAJOR_VERSION - the major version of Hspell
HSPELL_MINOR_VERSION - The minor version of Hspell
```

FindHTMLHelp

This module looks for Microsoft HTML Help Compiler

It defines:

```
HTML_HELP_COMPILER : full path to the Compiler (hhc.exe)
HTML_HELP_INCLUDE_PATH : include path to the API (htmlhelp.h)
HTML_HELP_LIBRARY : full path to the library (htmlhelp.lib)
```

FindIcotool

Find icotool

This module looks for icotool. This module defines the following values:

```
ICOTOOL_EXECUTABLE: the full path to the icotool tool.
ICOTOOL_FOUND: True if icotool has been found.
ICOTOOL_VERSION_STRING: the version of icotool found.
```

FindImageMagick

Find the ImageMagick binary suite.

This module will search for a set of ImageMagick tools specified as components in the FIND_PACKAGE call. Typical components include, but are not limited to (future versions of ImageMagick might have additional components not listed here):

```
animate
compare
composite
conjure
convert
display
identify
import
mogrify
montage
stream
```

If no component is specified in the FIND_PACKAGE call, then it only searches for the ImageMagick executable directory. This code defines the following variables:

```
ImageMagick_FOUND - TRUE if all components are found.
ImageMagick_EXECUTABLE_DIR - Full path to executables directory.
ImageMagick_<component>_FOUND - TRUE if <component> is found.
ImageMagick_<component>_EXECUTABLE - Full path to <component> executable.
ImageMagick_VERSION_STRING - the version of ImageMagick found
(since CMake 2.8.8)
```

ImageMagick_VERSION_STRING will not work for old versions like 5.2.3.

There are also components for the following ImageMagick APIs:

```
Magick++
MagickWand
MagickCore
```

For these components the following variables are set:

```
ImageMagick_FOUND - TRUE if all components are found.
ImageMagick_INCLUDE_DIRS - Full paths to all include dirs.
ImageMagick_LIBRARIES - Full paths to all libraries.
ImageMagick_<component>_FOUND - TRUE if <component> is found.
ImageMagick_<component>_INCLUDE_DIRS - Full path to <component> include dirs.
ImageMagick_<component>_LIBRARIES - Full path to <component> libraries.
```

Example Usages:

```
find_package(ImageMagick)
find_package(ImageMagick COMPONENTS convert)
find_package(ImageMagick COMPONENTS convert mogrify display)
find_package(ImageMagick COMPONENTS Magick++)
find_package(ImageMagick COMPONENTS Magick++ convert)
```

Note that the standard FIND_PACKAGE features are supported (i.e., QUIET, REQUIRED, etc.).

FindITK
Find an ITK installation or build tree.

FindJasper
Try to find the Jasper JPEG2000 library

Once done this will define

```
JASPER_FOUND - system has Jasper
JASPER_INCLUDE_DIR - the Jasper include directory
JASPER_LIBRARIES - the libraries needed to use Jasper
JASPER_VERSION_STRING - the version of Jasper found (since CMake 2.8.8)
```

FindJava
Find Java

This module finds if Java is installed and determines where the include files and libraries are. The caller may set variable JAVA_HOME to specify a Java installation prefix explicitly.

This module sets the following result variables:

```
Java_JAVA_EXECUTABLE = the full path to the Java runtime
Java_JAVAC_EXECUTABLE = the full path to the Java compiler
Java_JAVAH_EXECUTABLE = the full path to the Java header generator
Java_JAVADOC_EXECUTABLE = the full path to the Java documention generator
Java_JAR_EXECUTABLE = the full path to the Java archiver
Java_VERSION_STRING = Version of the package found (java version), eg. 1.6.0_12
Java_VERSION_MAJOR = The major version of the package found.
Java_VERSION_MINOR = The minor version of the package found.
Java_VERSION_PATCH = The patch version of the package found.
Java_VERSION_TWEAK = The tweak version of the package found (after '_')
Java_VERSION = This is set to: $major.$minor.$patch(.$tweak)
```

The minimum required version of Java can be specified using the standard CMake syntax, e.g. find_package(Java 1.5)

NOTE: ${Java_VERSION} and ${Java_VERSION_STRING} are not guaranteed to be identical. For example some java version may return: Java_VERSION_STRING = 1.5.0_17 and Java_VERSION = 1.5.0.17

another example is the Java OEM, with: Java_VERSION_STRING = 1.6.0-oem and Java_VERSION = 1.6.0

For these components the following variables are set:

```
Java_FOUND - TRUE if all components are found.
Java_INCLUDE_DIRS - Full paths to all include dirs.
Java_LIBRARIES - Full paths to all libraries.
Java_<component>_FOUND - TRUE if <component> is found.
```

Example Usages:

```
find_package(Java)
find_package(Java COMPONENTS Runtime)
find_package(Java COMPONENTS Development)
```

FindJNI
Find JNI java libraries.

This module finds if Java is installed and determines where the include files and libraries are. It also determines what the name of the library is. The caller may set variable JAVA_HOME to specify a Java installation prefix explicitly.

This module sets the following result variables:

```
JNI_INCLUDE_DIRS = the include dirs to use
JNI_LIBRARIES = the libraries to use
JNI_FOUND = TRUE if JNI headers and libraries were found.
JAVA_AWT_LIBRARY = the path to the jawt library
JAVA_JVM_LIBRARY = the path to the jvm library
JAVA_INCLUDE_PATH = the include path to jni.h
JAVA_INCLUDE_PATH2 = the include path to jni_md.h
JAVA_AWT_INCLUDE_PATH = the include path to jawt.h
```

FindJPEG

Find JPEG

Find the native JPEG includes and library This module defines

```
JPEG_INCLUDE_DIR, where to find jpeglib.h, etc.
JPEG_LIBRARIES, the libraries needed to use JPEG.
JPEG_FOUND, If false, do not try to use JPEG.
```

also defined, but not for general use are

```
JPEG_LIBRARY, where to find the JPEG library.
```

FindKDE3

Find the KDE3 include and library dirs, KDE preprocessors and define a some macros

This module defines the following variables:

```
KDE3_DEFINITIONS - compiler definitions required for compiling KDE software
KDE3_INCLUDE_DIR - the KDE include directory
KDE3_INCLUDE_DIRS - the KDE and the Qt include directory, for use with include_directories
KDE3_LIB_DIR - the directory where the KDE libraries are installed, for use with link_dire
QT_AND_KDECORE_LIBS - this contains both the Qt and the kdecore library
KDE3_DCOPIDL_EXECUTABLE - the dcopidl executable
KDE3_DCOPIDL2CPP_EXECUTABLE - the dcopidl2cpp executable
KDE3_KCFGC_EXECUTABLE - the kconfig_compiler executable
KDE3_FOUND - set to TRUE if all of the above has been found
```

The following user adjustable options are provided:

```
KDE3_BUILD_TESTS - enable this to build KDE testcases
```

It also adds the following macros (from KDE3Macros.cmake) SRCS_VAR is always the variable which contains the list of source files for your application or library.

KDE3_AUTOMOC(file1 ... fileN)

```
Call this if you want to have automatic moc file handling.
This means if you include "foo.moc" in the source file foo.cpp
a moc file for the header foo.h will be created automatically.
You can set the property SKIP_AUTOMAKE using set_source_files_properties()
to exclude some files in the list from being processed.
```

KDE3_ADD_MOC_FILES(SRCS_VAR file1 ... fileN)

```
If you don't use the KDE3_AUTOMOC() macro, for the files
listed here moc files will be created (named "foo.moc.cpp")
```

KDE3_ADD_DCOP_SKELS(SRCS_VAR header1.h ... headerN.h)

```
Use this to generate DCOP skeletions from the listed headers.
```

KDE3_ADD_DCOP_STUBS(SRCS_VAR header1.h ... headerN.h)

```
Use this to generate DCOP stubs from the listed headers.
```

KDE3_ADD_UI_FILES(SRCS_VAR file1.ui ... fileN.ui)

```
Use this to add the Qt designer ui files to your application/library.
```

KDE3_ADD_KCFG_FILES(SRCS_VAR file1.kcfgc ... fileN.kcfgc)

```
Use this to add KDE kconfig compiler files to your application/library.
```

KDE3_INSTALL_LIBTOOL_FILE(target)

```
This will create and install a simple libtool file for the given target.
```

KDE3_ADD_EXECUTABLE(name file1 ... fileN)

```
Currently identical to add_executable(), may provide some advanced features in the future.
```

KDE3_ADD_KPART(name [WITH_PREFIX] file1 ... fileN)

```
Create a KDE plugin (KPart, kioslave, etc.) from the given source files.
If WITH_PREFIX is given, the resulting plugin will have the prefix "lib", otherwise it wor
It creates and installs an appropriate libtool la-file.
```

KDE3_ADD_KDEINIT_EXECUTABLE(name file1 ... fileN)

```
Create a KDE application in the form of a module loadable via kdeinit.
A library named kdeinit_<name> will be created and a small executable which links to it.
```

The option KDE3_ENABLE_FINAL to enable all-in-one compilation is no longer supported.

Author: Alexander Neundorf <*neundorf@kde.org*>

FindKDE4

Find KDE4 and provide all necessary variables and macros to compile software for it. It looks for KDE 4 in the following directories in the given order:

```
CMAKE_INSTALL_PREFIX
KDEDIRS
/opt/kde4
```

Please look in FindKDE4Internal.cmake and KDE4Macros.cmake for more information. They are installed with the KDE 4 libraries in $KDEDIRS/share/apps/cmake/modules/.

Author: Alexander Neundorf <*neundorf@kde.org*>

FindLAPACK

Find LAPACK library

This module finds an installed fortran library that implements the LAPACK linear-algebra interface (see *http://www.netlib.org/lapack/*).

The approach follows that taken for the autoconf macro file, acx_lapack.m4 (distributed at *http://ac-archive.sourceforge.net/ac-archive/acx_lapack.html*).

This module sets the following variables:

```
LAPACK_FOUND - set to true if a library implementing the LAPACK interface
is found
LAPACK_LINKER_FLAGS - uncached list of required linker flags (excluding -l
and -L).
LAPACK_LIBRARIES - uncached list of libraries (using full path name) to
link against to use LAPACK
LAPACK95_LIBRARIES - uncached list of libraries (using full path name) to
link against to use LAPACK95
LAPACK95_FOUND - set to true if a library implementing the LAPACK f95
```

```
interface is found
BLA_STATIC if set on this determines what kind of linkage we do (static)
BLA_VENDOR if set checks only the specified vendor, if not set checks
all the possibilities
BLA_F95 if set on tries to find the f95 interfaces for BLAS/LAPACK
```

List of vendors (BLA_VENDOR) valid in this module # Intel(mkl), ACML,Apple, NAS, Generic

FindLATEX

Find Latex

This module finds if Latex is installed and determines where the executables are. This code sets the following variables:

```
LATEX_COMPILER: path to the LaTeX compiler
PDFLATEX_COMPILER: path to the PdfLaTeX compiler
BIBTEX_COMPILER: path to the BibTeX compiler
MAKEINDEX_COMPILER: path to the MakeIndex compiler
DVIPS_CONVERTER: path to the DVIPS converter
PS2PDF_CONVERTER: path to the PS2PDF converter
LATEX2HTML_CONVERTER: path to the LaTeX2Html converter
```

FindLibArchive

Find libarchive library and headers

The module defines the following variables:

```
LibArchive_FOUND - true if libarchive was found
LibArchive_INCLUDE_DIRS - include search path
LibArchive_LIBRARIES - libraries to link
LibArchive_VERSION - libarchive 3-component version number
```

FindLibLZMA

Find LibLZMA

Find LibLZMA headers and library

```
LIBLZMA_FOUND - True if liblzma is found.
LIBLZMA_INCLUDE_DIRS - Directory where liblzma headers are located.
LIBLZMA_LIBRARIES - Lzma libraries to link against.
LIBLZMA_HAS_AUTO_DECODER - True if lzma_auto_decoder() is found (required).
LIBLZMA_HAS_EASY_ENCODER - True if lzma_easy_encoder() is found (required).
LIBLZMA_HAS_LZMA_PRESET - True if lzma_lzma_preset() is found (required).
LIBLZMA_VERSION_MAJOR - The major version of lzma
LIBLZMA_VERSION_MINOR - The minor version of lzma
LIBLZMA_VERSION_PATCH - The patch version of lzma
LIBLZMA_VERSION_STRING - version number as a string (ex: "5.0.3")
```

FindLibXml2

Try to find the LibXml2 xml processing library

Once done this will define

```
LIBXML2_FOUND - System has LibXml2
LIBXML2_INCLUDE_DIR - The LibXml2 include directory
LIBXML2_LIBRARIES - The libraries needed to use LibXml2
LIBXML2_DEFINITIONS - Compiler switches required for using LibXml2
LIBXML2_XMLLINT_EXECUTABLE - The XML checking tool xmllint coming with LibXml2
LIBXML2_VERSION_STRING - the version of LibXml2 found (since CMake 2.8.8)
```

FindLibXslt

Try to find the LibXslt library

Once done this will define

```
LIBXSLT_FOUND - system has LibXslt
LIBXSLT_INCLUDE_DIR - the LibXslt include directory
LIBXSLT_LIBRARIES - Link these to LibXslt
LIBXSLT_DEFINITIONS - Compiler switches required for using LibXslt
LIBXSLT_VERSION_STRING - version of LibXslt found (since CMake 2.8.8)
```

Additionally, the following two variables are set (but not required for using xslt):

```
LIBXSLT_EXSLT_LIBRARIES - Link to these if you need to link against the exslt library
LIBXSLT_XSLTPROC_EXECUTABLE - Contains the full path to the xsltproc executable if found
```

FindLua50

Locate Lua library This module defines

```
LUA50_FOUND, if false, do not try to link to Lua
LUA_LIBRARIES, both lua and lualib
LUA_INCLUDE_DIR, where to find lua.h and lualib.h (and probably lauxlib.h)
```

Note that the expected include convention is

```
#include "lua.h"
```

and not

```
#include <lua/lua.h>
```

This is because, the lua location is not standardized and may exist in locations other than lua/

FindLua51

Locate Lua library This module defines

```
LUA51_FOUND, if false, do not try to link to Lua
LUA_LIBRARIES
LUA_INCLUDE_DIR, where to find lua.h
LUA_VERSION_STRING, the version of Lua found (since CMake 2.8.8)
```

Note that the expected include convention is

```
#include "lua.h"
```

and not

```
#include <lua/lua.h>
```

This is because, the lua location is not standardized and may exist in locations other than lua/

FindLua

Locate Lua library This module defines

```
LUA_FOUND - if false, do not try to link to Lua
LUA_LIBRARIES - both lua and lualib
LUA_INCLUDE_DIR - where to find lua.h
LUA_VERSION_STRING - the version of Lua found
LUA_VERSION_MAJOR - the major version of Lua
LUA_VERSION_MINOR - the minor version of Lua
LUA_VERSION_PATCH - the patch version of Lua
```

Note that the expected include convention is

```
#include "lua.h"
```

and not

```
#include <lua/lua.h>
```

This is because, the lua location is not standardized and may exist in locations other than lua/

FindMatlab
this module looks for Matlab

Defines:

```
MATLAB_INCLUDE_DIR: include path for mex.h, engine.h
MATLAB_LIBRARIES: required libraries: libmex, etc
MATLAB_MEX_LIBRARY: path to libmex.lib
MATLAB_MX_LIBRARY: path to libmx.lib
MATLAB_ENG_LIBRARY: path to libeng.lib
```

FindMFC
Find MFC on Windows

Find the native MFC - i.e. decide if an application can link to the MFC libraries.

```
MFC_FOUND - Was MFC support found
```

You dont need to include anything or link anything to use it.

FindMotif
Try to find Motif (or lesstif)

Once done this will define:

```
MOTIF_FOUND - system has MOTIF
MOTIF_INCLUDE_DIR - include paths to use Motif
MOTIF_LIBRARIES - Link these to use Motif
```

FindMPEG2
Find the native MPEG2 includes and library

This module defines

```
MPEG2_INCLUDE_DIR, path to mpeg2dec/mpeg2.h, etc.
MPEG2_LIBRARIES, the libraries required to use MPEG2.
MPEG2_FOUND, If false, do not try to use MPEG2.
```

also defined, but not for general use are

```
MPEG2_mpeg2_LIBRARY, where to find the MPEG2 library.
MPEG2_vo_LIBRARY, where to find the vo library.
```

FindMPEG
Find the native MPEG includes and library

This module defines

```
MPEG_INCLUDE_DIR, where to find MPEG.h, etc.
MPEG_LIBRARIES, the libraries required to use MPEG.
MPEG_FOUND, If false, do not try to use MPEG.
```

also defined, but not for general use are

```
MPEG_mpeg2_LIBRARY, where to find the MPEG library.
MPEG_vo_LIBRARY, where to find the vo library.
```

FindMPI
Find a Message Passing Interface (MPI) implementation

The Message Passing Interface (MPI) is a library used to write high-performance distributed-

memory parallel applications, and is typically deployed on a cluster. MPI is a standard interface (defined by the MPI forum) for which many implementations are available. All of them have somewhat different include paths, libraries to link against, etc., and this module tries to smooth out those differences.

=== Variables ===

This module will set the following variables per language in your project, where <lang> is one of C, CXX, or Fortran:

```
MPI_<lang>_FOUND TRUE if FindMPI found MPI flags for <lang>
MPI_<lang>_COMPILER MPI Compiler wrapper for <lang>
MPI_<lang>_COMPILE_FLAGS Compilation flags for MPI programs
MPI_<lang>_INCLUDE_PATH Include path(s) for MPI header
MPI_<lang>_LINK_FLAGS Linking flags for MPI programs
MPI_<lang>_LIBRARIES All libraries to link MPI programs against
```

Additionally, FindMPI sets the following variables for running MPI programs from the command line:

```
MPIEXEC Executable for running MPI programs
MPIEXEC_NUMPROC_FLAG Flag to pass to MPIEXEC before giving
it the number of processors to run on
MPIEXEC_PREFLAGS Flags to pass to MPIEXEC directly
before the executable to run.
MPIEXEC_POSTFLAGS Flags to pass to MPIEXEC after other flags
```

=== Usage ===

To use this module, simply call FindMPI from a CMakeLists.txt file, or run find_package(MPI), then run CMake. If you are happy with the auto- detected configuration for your language, then youre done. If not, you have two options:

```
1. Set MPI_<lang>_COMPILER to the MPI wrapper (mpicc, etc.) of your
choice and reconfigure. FindMPI will attempt to determine all the
necessary variables using THAT compiler's compile and link flags.
2. If this fails, or if your MPI implementation does not come with
a compiler wrapper, then set both MPI_<lang>_LIBRARIES and
MPI_<lang>_INCLUDE_PATH. You may also set any other variables
listed above, but these two are required. This will circumvent
autodetection entirely.
```

When configuration is successful, MPI_<lang>_COMPILER will be set to the compiler wrapper for <lang>, if it was found. MPI_<lang>_FOUND and other variables above will be set if any MPI implementation was found for <lang>, regardless of whether a compiler was found.

When using MPIEXEC to execute MPI applications, you should typically use all of the MPIEXEC flags as follows:

```
${MPIEXEC} ${MPIEXEC_NUMPROC_FLAG} PROCS
${MPIEXEC_PREFLAGS} EXECUTABLE ${MPIEXEC_POSTFLAGS} ARGS
```

where PROCS is the number of processors on which to execute the program, EXECUTABLE is the MPI program, and ARGS are the arguments to pass to the MPI program.

=== Backward Compatibility ===

For backward compatibility with older versions of FindMPI, these variables are set, but deprecated:

```
MPI_FOUND MPI_COMPILER MPI_LIBRARY
MPI_COMPILE_FLAGS MPI_INCLUDE_PATH MPI_EXTRA_LIBRARY
```

```
MPI_LINK_FLAGS MPI_LIBRARIES
```

In new projects, please use the MPI_<lang>_XXX equivalents.

FindOpenAL

Locate OpenAL This module defines OPENAL_LIBRARY OPENAL_FOUND, if false, do not try to link to OpenAL OPENAL_INCLUDE_DIR, where to find the headers

$OPENALDIR is an environment variable that would correspond to the

Created by Eric Wing. This was influenced by the FindSDL.cmake module.

FindOpenGL

Try to find OpenGL

Once done this will define

```
OPENGL_FOUND - system has OpenGL
OPENGL_XMESA_FOUND - system has XMESA
OPENGL_GLU_FOUND - system has GLU
OPENGL_INCLUDE_DIR - the GL include directory
OPENGL_LIBRARIES - Link these to use OpenGL and GLU
```

If you want to use just GL you can use these values

```
OPENGL_gl_LIBRARY - Path to OpenGL Library
OPENGL_glu_LIBRARY - Path to GLU Library
```

On OSX default to using the framework version of opengl People will have to change the cache values of OPENGL_glu_LIBRARY and OPENGL_gl_LIBRARY to use OpenGL with X11 on OSX

FindOpenMP

Finds OpenMP support

This module can be used to detect OpenMP support in a compiler. If the compiler supports OpenMP, the flags required to compile with OpenMP support are returned in variables for the different languages. The variables may be empty if the compiler does not need a special flag to support OpenMP.

The following variables are set:

```
OpenMP_C_FLAGS - flags to add to the C compiler for OpenMP support
OpenMP_CXX_FLAGS - flags to add to the CXX compiler for OpenMP support
OPENMP_FOUND - true if openmp is detected
```

Supported compilers can be found at *http://openmp.org/wp/openmp-compilers/*

FindOpenSceneGraph

Find OpenSceneGraph

This module searches for the OpenSceneGraph core osg library as well as OpenThreads, and whatever additional COMPONENTS (nodekits) that you specify.

```
See http://www.openscenegraph.org
```

NOTE: To use this module effectively you must either require CMake >= 2.6.3 with cmake_minimum_required(VERSION 2.6.3) or download and place FindOpenThreads.cmake, Findosg_functions.cmake, Findosg.cmake, and Find<etc>.cmake files into your CMAKE_MODULE_PATH.

This module accepts the following variables (note mixed case)

```
OpenSceneGraph_DEBUG - Enable debugging output
```

```
OpenSceneGraph_MARK_AS_ADVANCED - Mark cache variables as advanced
automatically
```

The following environment variables are also respected for finding the OSG and its various components. CMAKE_PREFIX_PATH can also be used for this (see find_library() CMake documentation).

```
<MODULE>_DIR (where MODULE is of the form "OSGVOLUME" and there is a FindosgVolume.cmake f
OSG_DIR
OSGDIR
OSG_ROOT
```

[CMake 2.8.10]: The CMake variable OSG_DIR can now be used as well to influence detection, instead of needing to specify an environment variable.

This module defines the following output variables:

```
OPENSCENEGRAPH_FOUND - Was the OSG and all of the specified components found?

OPENSCENEGRAPH_VERSION - The version of the OSG which was found

OPENSCENEGRAPH_INCLUDE_DIRS - Where to find the headers

OPENSCENEGRAPH_LIBRARIES - The OSG libraries
```

========== Example Usage:

```
find_package(OpenSceneGraph 2.0.0 REQUIRED osgDB osgUtil)
# libOpenThreads & libosg automatically searched
include_directories(${OPENSCENEGRAPH_INCLUDE_DIRS})

add_executable(foo foo.cc)
target_link_libraries(foo ${OPENSCENEGRAPH_LIBRARIES})
```

FindOpenSSL

Try to find the OpenSSL encryption library

Once done this will define

```
OPENSSL_ROOT_DIR - Set this variable to the root installation of OpenSSL
```

Read-Only variables:

```
OPENSSL_FOUND - system has the OpenSSL library
OPENSSL_INCLUDE_DIR - the OpenSSL include directory
OPENSSL_LIBRARIES - The libraries needed to use OpenSSL
OPENSSL_VERSION - This is set to $major.$minor.$revision$path (eg. 0.9.8s)
```

FindOpenThreads

OpenThreads is a C++ based threading library. Its largest userbase seems to OpenSceneGraph so you might notice I accept OSGDIR as an environment path. I consider this part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module.

Locate OpenThreads This module defines OPENTHREADS_LIBRARY OPENTHREADS_FOUND, if false, do not try to link to OpenThreads OPENTHREADS_INCLUDE_DIR, where to find the headers

$OPENTHREADS_DIR is an environment variable that would correspond to the ./configure --prefix=$OPENTHREADS_DIR used in building osg.

[CMake 2.8.10]: The CMake variables OPENTHREADS_DIR or OSG_DIR can now be used as well to influence detection, instead of needing to specify an environment variable.

Created by Eric Wing.

FindosgAnimation

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgAnimation This module defines

OSGANIMATION_FOUND - Was osgAnimation found? OSGANIMATION_INCLUDE_DIR - Where to find the headers OSGANIMATION_LIBRARIES - The libraries to link against for the OSG (use this)

OSGANIMATION_LIBRARY - The OSG library OSGANIMATION_LIBRARY_DEBUG - The OSG debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgDB

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgDB This module defines

OSGDB_FOUND - Was osgDB found? OSGDB_INCLUDE_DIR - Where to find the headers OSGDB_LIBRARIES - The libraries to link against for the osgDB (use this)

OSGDB_LIBRARY - The osgDB library OSGDB_LIBRARY_DEBUG - The osgDB debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

Findosg_functions

This CMake file contains two macros to assist with searching for OSG libraries and nodekits. Please see FindOpenSceneGraph.cmake for full documentation.

FindosgFX

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgFX This module defines

OSGFX_FOUND - Was osgFX found? OSGFX_INCLUDE_DIR - Where to find the headers OSGFX_LIBRARIES - The libraries to link against for the osgFX (use this)

OSGFX_LIBRARY - The osgFX library OSGFX_LIBRARY_DEBUG - The osgFX debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgGA

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgGA This module defines

OSGGA_FOUND - Was osgGA found? OSGGA_INCLUDE_DIR - Where to find the headers OSGGA_LIBRARIES - The libraries to link against for the osgGA (use this)

OSGGA_LIBRARY - The osgGA library OSGGA_LIBRARY_DEBUG - The osgGA debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgIntrospection

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgINTROSPECTION This module defines

OSGINTROSPECTION_FOUND - Was osgIntrospection found? OSGINTROSPECTION_INCLUDE_DIR - Where to find the headers OSGINTROSPECTION_LIBRARIES - The libraries to link for osgIntrospection (use this)

OSGINTROSPECTION_LIBRARY - The osgIntrospection library OSGINTROSPECTION_LIBRARY_DEBUG - The osgIntrospection debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgManipulator

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgManipulator This module defines

OSGMANIPULATOR_FOUND - Was osgManipulator found? OSGMANIPULATOR_INCLUDE_DIR - Where to find the headers OSGMANIPULATOR_LIBRARIES - The libraries to link for osgManipulator (use this)

OSGMANIPULATOR_LIBRARY - The osgManipulator library OSGMANIPULATOR_LIBRARY_DEBUG - The osgManipulator debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgParticle

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgParticle This module defines

OSGPARTICLE_FOUND - Was osgParticle found? OSGPARTICLE_INCLUDE_DIR - Where to find the headers OSGPARTICLE_LIBRARIES - The libraries to link for osgParticle (use this)

OSGPARTICLE_LIBRARY - The osgParticle library OSGPARTICLE_LIBRARY_DEBUG - The osgParticle debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgPresentation

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgPresentation This module defines

OSGPRESENTATION_FOUND - Was osgPresentation found? OSGPRESENTATION_INCLUDE_DIR - Where to find the headers OSGPRESENTATION_LIBRARIES - The libraries to link for osgPresentation (use this)

OSGPRESENTATION_LIBRARY - The osgPresentation library OSGPRESENTATION_LIBRARY_DEBUG - The osgPresentation debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing. Modified to work with osgPresentation by Robert Osfield, January 2012.

FindosgProducer

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgProducer This module defines

OSGPRODUCER_FOUND - Was osgProducer found? OSGPRODUCER_INCLUDE_DIR - Where to find the headers OSGPRODUCER_LIBRARIES - The libraries to link for osgProducer (use this)

OSGPRODUCER_LIBRARY - The osgProducer library OSGPRODUCER_LIBRARY_DEBUG -

The osgProducer debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgQt

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgQt This module defines

OSGQT_FOUND - Was osgQt found? OSGQT_INCLUDE_DIR - Where to find the headers OSGQT_LIBRARIES - The libraries to link for osgQt (use this)

OSGQT_LIBRARY - The osgQt library OSGQT_LIBRARY_DEBUG - The osgQt debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing. Modified to work with osgQt by Robert Osfield, January 2012.

Findosg

NOTE: It is highly recommended that you use the new FindOpenSceneGraph.cmake introduced in CMake 2.6.3 and not use this Find module directly.

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osg This module defines

OSG_FOUND - Was the Osg found? OSG_INCLUDE_DIR - Where to find the headers OSG_LIBRARIES - The libraries to link against for the OSG (use this)

OSG_LIBRARY - The OSG library OSG_LIBRARY_DEBUG - The OSG debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgShadow

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgShadow This module defines

OSGSHADOW_FOUND - Was osgShadow found? OSGSHADOW_INCLUDE_DIR - Where to find the headers OSGSHADOW_LIBRARIES - The libraries to link for osgShadow (use this)

OSGSHADOW_LIBRARY - The osgShadow library OSGSHADOW_LIBRARY_DEBUG - The

osgShadow debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgSim
This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgSim This module defines

OSGSIM_FOUND - Was osgSim found? OSGSIM_INCLUDE_DIR - Where to find the headers OSGSIM_LIBRARIES - The libraries to link for osgSim (use this)

OSGSIM_LIBRARY - The osgSim library OSGSIM_LIBRARY_DEBUG - The osgSim debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgTerrain
This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgTerrain This module defines

OSGTERRAIN_FOUND - Was osgTerrain found? OSGTERRAIN_INCLUDE_DIR - Where to find the headers OSGTERRAIN_LIBRARIES - The libraries to link for osgTerrain (use this)

OSGTERRAIN_LIBRARY - The osgTerrain library OSGTERRAIN_LIBRARY_DEBUG - The osgTerrain debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgText
This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgText This module defines

OSGTEXT_FOUND - Was osgText found? OSGTEXT_INCLUDE_DIR - Where to find the headers OSGTEXT_LIBRARIES - The libraries to link for osgText (use this)

OSGTEXT_LIBRARY - The osgText library OSGTEXT_LIBRARY_DEBUG - The osgText debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgUtil

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgUtil This module defines

OSGUTIL_FOUND - Was osgUtil found? OSGUTIL_INCLUDE_DIR - Where to find the headers OSGUTIL_LIBRARIES - The libraries to link for osgUtil (use this)

OSGUTIL_LIBRARY - The osgUtil library OSGUTIL_LIBRARY_DEBUG - The osgUtil debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgViewer

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgViewer This module defines

OSGVIEWER_FOUND - Was osgViewer found? OSGVIEWER_INCLUDE_DIR - Where to find the headers OSGVIEWER_LIBRARIES - The libraries to link for osgViewer (use this)

OSGVIEWER_LIBRARY - The osgViewer library OSGVIEWER_LIBRARY_DEBUG - The osgViewer debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgVolume

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgVolume This module defines

OSGVOLUME_FOUND - Was osgVolume found? OSGVOLUME_INCLUDE_DIR - Where to find the headers OSGVOLUME_LIBRARIES - The libraries to link for osgVolume (use this)

OSGVOLUME_LIBRARY - The osgVolume library OSGVOLUME_LIBRARY_DEBUG - The osgVolume debug library

$OSGDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindosgWidget

This is part of the Findosg* suite used to find OpenSceneGraph components. Each component is separate and you must opt in to each module. You must also opt into OpenGL and OpenThreads (and Producer if needed) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate osgWidget This module defines

OSGWIDGET_FOUND - Was osgWidget found? OSGWIDGET_INCLUDE_DIR - Where to find the headers OSGWIDGET_LIBRARIES - The libraries to link for osgWidget (use this)

OSGWIDGET_LIBRARY - The osgWidget library OSGWIDGET_LIBRARY_DEBUG - The osgWidget debug library

$OSGDIR is an environment variable that would correspond to the

FindosgWidget.cmake tweaked from Findosg* suite as created by Eric Wing.

FindPackageHandleStandardArgs

FIND_PACKAGE_HANDLE_STANDARD_ARGS(<name> ...)

This function is intended to be used in FindXXX.cmake modules files. It handles the REQUIRED, QUIET and version-related arguments to find_package(). It also sets the <packagename>_FOUND variable. The package is considered found if all variables <var1>... listed contain valid results, e.g. valid filepaths.

There are two modes of this function. The first argument in both modes is the name of the Findmodule where it is called (in original casing).

The first simple mode looks like this:

```
FIND_PACKAGE_HANDLE_STANDARD_ARGS(<name> (DEFAULT_MSG|"Custom failure message") <var1>...<
```

If the variables <var1> to <varN> are all valid, then <UPPERCASED_NAME>_FOUND will be set to TRUE. If DEFAULT_MSG is given as second argument, then the function will generate itself useful success and error messages. You can also supply a custom error message for the failure case. This is not recommended.

The second mode is more powerful and also supports version checking:

```
FIND_PACKAGE_HANDLE_STANDARD_ARGS(NAME [FOUND_VAR <resultVar>]
[REQUIRED_VARS <var1>...<varN>]
[VERSION_VAR <versionvar>]
[HANDLE_COMPONENTS]
[CONFIG_MODE]
[FAIL_MESSAGE "Custom failure message"] )
```

In this mode, the name of the result-variable can be set either to either <UPPERCASED_NAME>_FOUND or <OriginalCase_Name>_FOUND using the FOUND_VAR option. Other names for the result-variable are not allowed. So for a Find-module named FindFooBar.cmake, the two possible names are FooBar_FOUND and FOOBAR_FOUND. It is recommended to use the original case version. If the FOUND_VAR option is not used, the default is <UPPERCASED_NAME>_FOUND.

As in the simple mode, if <var1> through <varN> are all valid, <packagename>_FOUND will be set to TRUE. After REQUIRED_VARS the variables which are required for this package are listed. Following VERSION_VAR the name of the variable can be specified which holds the version of the package which has been found. If this is done, this version will be checked against the

(potentially) specified required version used in the find_package() call. The EXACT keyword is also handled. The default messages include information about the required version and the version which has been actually found, both if the version is ok or not. If the package supports components, use the HANDLE_COMPONENTS option to enable handling them. In this case, find_package_handle_standard_args() will report which components have been found and which are missing, and the <packagename>_FOUND variable will be set to FALSE if any of the required components (i.e. not the ones listed after OPTIONAL_COMPONENTS) are missing. Use the option CONFIG_MODE if your FindXXX.cmake module is a wrapper for a find_package(... NO_MODULE) call. In this case VERSION_VAR will be set to <NAME>_VERSION and the macro will automatically check whether the Config module was found. Via FAIL_MESSAGE a custom failure message can be specified, if this is not used, the default message will be displayed.

Example for mode 1:

```
find_package_handle_standard_args(LibXml2 DEFAULT_MSG LIBXML2_LIBRARY LIBXML2_INCLUDE_DIR)
```

LibXml2 is considered to be found, if both LIBXML2_LIBRARY and LIBXML2_INCLUDE_DIR are valid. Then also LIBXML2_FOUND is set to TRUE. If it is not found and REQUIRED was used, it fails with FATAL_ERROR, independent whether QUIET was used or not. If it is found, success will be reported, including the content of <var1>. On repeated Cmake runs, the same message wont be printed again.

Example for mode 2:

```
find_package_handle_standard_args(LibXslt FOUND_VAR LibXslt_FOUND
REQUIRED_VARS LibXslt_LIBRARIES LibXslt_INCLUDE_DIRS
VERSION_VAR LibXslt_VERSION_STRING)
```

In this case, LibXslt is considered to be found if the variable(s) listed after REQUIRED_VAR are all valid, i.e. LibXslt_LIBRARIES and LibXslt_INCLUDE_DIRS in this case. The result will then be stored in LibXslt_FOUND . Also the version of LibXslt will be checked by using the version contained in LibXslt_VERSION_STRING. Since no FAIL_MESSAGE is given, the default messages will be printed.

Another example for mode 2:

```
find_package(Automoc4 QUIET NO_MODULE HINTS /opt/automoc4)
find_package_handle_standard_args(Automoc4 CONFIG_MODE)
```

In this case, FindAutmoc4.cmake wraps a call to find_package(Automoc4 NO_MODULE) and adds an additional search directory for automoc4. Here the result will be stored in AUTOMOC4_FOUND. The following FIND_PACKAGE_HANDLE_STANDARD_ARGS() call produces a proper success/error message.

FindPackageMessage

FIND_PACKAGE_MESSAGE(<name> message for user find result details)

This macro is intended to be used in FindXXX.cmake modules files. It will print a message once for each unique find result. This is useful for telling the user where a package was found. The first argument specifies the name (XXX) of the package. The second argument specifies the message to display. The third argument lists details about the find result so that if they change the message will be displayed again. The macro also obeys the QUIET argument to the find_package command.

Example:

```
if(X11_FOUND)
FIND_PACKAGE_MESSAGE(X11 "Found X11: ${X11_X11_LIB}"
"[${X11_X11_LIB}][${X11_INCLUDE_DIR}]")
else()
endif()
```

FindPerlLibs
Find Perl libraries

This module finds if PERL is installed and determines where the include files and libraries are. It also determines what the name of the library is. This code sets the following variables:

```
PERLLIBS_FOUND = True if perl.h & libperl were found
PERL_INCLUDE_PATH = path to where perl.h is found
PERL_LIBRARY = path to libperl
PERL_EXECUTABLE = full path to the perl binary
```

The minimum required version of Perl can be specified using the standard syntax, e.g. find_package(PerlLibs 6.0)

```
The following variables are also available if needed
(introduced after CMake 2.6.4)
```

```
PERL_SITESEARCH = path to the sitesearch install dir
PERL_SITELIB = path to the sitelib install directory
PERL_VENDORARCH = path to the vendor arch install directory
PERL_VENDORLIB = path to the vendor lib install directory
PERL_ARCHLIB = path to the arch lib install directory
PERL_PRIVLIB = path to the priv lib install directory
PERL_EXTRA_C_FLAGS = Compilation flags used to build perl
```

FindPerl
Find perl

this module looks for Perl

```
PERL_EXECUTABLE - the full path to perl
PERL_FOUND - If false, don't attempt to use perl.
PERL_VERSION_STRING - version of perl found (since CMake 2.8.8)
```

FindPHP4
Find PHP4

This module finds if PHP4 is installed and determines where the include files and libraries are. It also determines what the name of the library is. This code sets the following variables:

```
PHP4_INCLUDE_PATH = path to where php.h can be found
PHP4_EXECUTABLE = full path to the php4 binary
```

FindPhysFS
Locate PhysFS library This module defines PHYSFS_LIBRARY, the name of the library to link against PHYSFS_FOUND, if false, do not try to link to PHYSFS PHYSFS_INCLUDE_DIR, where to find physfs.h

$PHYSFSDIR is an environment variable that would correspond to the

Created by Eric Wing.

FindPike
Find Pike

This module finds if PIKE is installed and determines where the include files and libraries are. It also determines what the name of the library is. This code sets the following variables:

```
PIKE_INCLUDE_PATH = path to where program.h is found
PIKE_EXECUTABLE = full path to the pike binary
```

FindPkgConfig
a pkg-config module for CMake

Usage:

```
pkg_check_modules(<PREFIX> [REQUIRED] [QUIET] <MODULE> [<MODULE>]*)
checks for all the given modules

pkg_search_module(<PREFIX> [REQUIRED] [QUIET] <MODULE> [<MODULE>]*)
checks for given modules and uses the first working one
```

When the REQUIRED argument was set, macros will fail with an error when module(s) could not be found

When the QUIET argument is set, no status messages will be printed.

It sets the following variables:

```
PKG_CONFIG_FOUND ... if pkg-config executable was found
PKG_CONFIG_EXECUTABLE ... pathname of the pkg-config program
PKG_CONFIG_VERSION_STRING ... the version of the pkg-config program found
(since CMake 2.8.8)
```

For the following variables two sets of values exist; first one is the common one and has the given PREFIX. The second set contains flags which are given out when pkgconfig was called with the --static option.

```
<XPREFIX>_FOUND ... set to 1 if module(s) exist
<XPREFIX>_LIBRARIES ... only the libraries (w/o the '-l')
<XPREFIX>_LIBRARY_DIRS ... the paths of the libraries (w/o the '-L')
<XPREFIX>_LDFLAGS ... all required linker flags
<XPREFIX>_LDFLAGS_OTHER ... all other linker flags
<XPREFIX>_INCLUDE_DIRS ... the '-I' preprocessor flags (w/o the '-I')
<XPREFIX>_CFLAGS ... all required cflags
<XPREFIX>_CFLAGS_OTHER ... the other compiler flags

<XPREFIX> = <PREFIX> for common case
<XPREFIX> = <PREFIX>_STATIC for static linking
```

There are some special variables whose prefix depends on the count of given modules. When there is only one module, <PREFIX> stays unchanged. When there are multiple modules, the prefix will be changed to <PREFIX>_<MODNAME>:

```
<XPREFIX>_VERSION ... version of the module
<XPREFIX>_PREFIX ... prefix-directory of the module
<XPREFIX>_INCLUDEDIR ... include-dir of the module
<XPREFIX>_LIBDIR ... lib-dir of the module

<XPREFIX> = <PREFIX> when |MODULES| == 1, else
<XPREFIX> = <PREFIX>_<MODNAME>
```

A <MODULE> parameter can have the following formats:

```
{MODNAME} ... matches any version
{MODNAME}>={VERSION} ... at least version <VERSION> is required
{MODNAME}={VERSION} ... exactly version <VERSION> is required
{MODNAME}<={VERSION} ... modules must not be newer than <VERSION>
```

Examples

```
pkg_check_modules (GLIB2 glib-2.0)

pkg_check_modules (GLIB2 glib-2.0>=2.10)
requires at least version 2.10 of glib2 and defines e.g.
GLIB2_VERSION=2.10.3
```

```
pkg_check_modules (FOO glib-2.0>=2.10 gtk+-2.0)
requires both glib2 and gtk2, and defines e.g.
FOO_glib-2.0_VERSION=2.10.3
FOO_gtk+-2.0_VERSION=2.8.20

pkg_check_modules (XRENDER REQUIRED xrender)
defines e.g.:
XRENDER_LIBRARIES=Xrender;X11
XRENDER_STATIC_LIBRARIES=Xrender;X11;pthread;Xau;Xdmcp

pkg_search_module (BAR libxml-2.0 libxml2 libxml>=2)
```

FindPNG

Find the native PNG includes and library

This module searches libpng, the library for working with PNG images.

It defines the following variables

```
PNG_INCLUDE_DIRS, where to find png.h, etc.
PNG_LIBRARIES, the libraries to link against to use PNG.
PNG_DEFINITIONS - You should add_definitons(${PNG_DEFINITIONS}) before compiling code that
PNG_FOUND, If false, do not try to use PNG.
PNG_VERSION_STRING - the version of the PNG library found (since CMake 2.8.8)
```

Also defined, but not for general use are

```
PNG_LIBRARY, where to find the PNG library.
```

For backward compatiblity the variable PNG_INCLUDE_DIR is also set. It has the same value as PNG_INCLUDE_DIRS.

Since PNG depends on the ZLib compression library, none of the above will be defined unless ZLib can be found.

FindPostgreSQL

Find the PostgreSQL installation.

In Windows, we make the assumption that, if the PostgreSQL files are installed, the default directory will be C:Program FilesPostgreSQL.

This module defines

```
PostgreSQL_LIBRARIES - the PostgreSQL libraries needed for linking
PostgreSQL_INCLUDE_DIRS - the directories of the PostgreSQL headers
PostgreSQL_VERSION_STRING - the version of PostgreSQL found (since CMake 2.8.8)
```

FindProducer

Though Producer isnt directly part of OpenSceneGraph, its primary user is OSG so I consider this part of the Findosg* suite used to find OpenSceneGraph components. Youll notice that I accept OSGDIR as an environment path.

Each component is separate and you must opt in to each module. You must also opt into OpenGL (and OpenThreads?) as these modules wont do it for you. This is to allow you control over your own system piece by piece in case you need to opt out of certain components or change the Find behavior for a particular module (perhaps because the default FindOpenGL.cmake module doesnt work with your system as an example). If you want to use a more convenient module that includes everything, use the FindOpenSceneGraph.cmake instead of the Findosg*.cmake modules.

Locate Producer This module defines PRODUCER_LIBRARY PRODUCER_FOUND, if false, do not try to link to Producer PRODUCER_INCLUDE_DIR, where to find the headers

$PRODUCER_DIR is an environment variable that would correspond to the

Created by Eric Wing.

FindProtobuf

Locate and configure the Google Protocol Buffers library.

The following variables can be set and are optional:

```
PROTOBUF_SRC_ROOT_FOLDER - When compiling with MSVC, if this cache variable is set
the protobuf-default VS project build locations
(vsprojects/Debug & vsprojects/Release) will be searched
for libraries and binaries.

PROTOBUF_IMPORT_DIRS - List of additional directories to be searched for
imported .proto files. (New in CMake 2.8.8)
```

Defines the following variables:

```
PROTOBUF_FOUND - Found the Google Protocol Buffers library (libprotobuf & header files)
PROTOBUF_INCLUDE_DIRS - Include directories for Google Protocol Buffers
PROTOBUF_LIBRARIES - The protobuf libraries
```

[New in CMake 2.8.5]

```
PROTOBUF_PROTOC_LIBRARIES - The protoc libraries
PROTOBUF_LITE_LIBRARIES - The protobuf-lite libraries
```

The following cache variables are also available to set or use:

```
PROTOBUF_LIBRARY - The protobuf library
PROTOBUF_PROTOC_LIBRARY - The protoc library
PROTOBUF_INCLUDE_DIR - The include directory for protocol buffers
PROTOBUF_PROTOC_EXECUTABLE - The protoc compiler
```

[New in CMake 2.8.5]

```
PROTOBUF_LIBRARY_DEBUG - The protobuf library (debug)
PROTOBUF_PROTOC_LIBRARY_DEBUG - The protoc library (debug)
PROTOBUF_LITE_LIBRARY - The protobuf lite library
PROTOBUF_LITE_LIBRARY_DEBUG - The protobuf lite library (debug)

=============================================================================
Example:

find_package(Protobuf REQUIRED)
include_directories(${PROTOBUF_INCLUDE_DIRS})

include_directories(${CMAKE_CURRENT_BINARY_DIR})
PROTOBUF_GENERATE_CPP(PROTO_SRCS PROTO_HDRS foo.proto)
add_executable(bar bar.cc ${PROTO_SRCS} ${PROTO_HDRS})
target_link_libraries(bar ${PROTOBUF_LIBRARIES})
```

NOTE: You may need to link against pthreads, depending

```
on the platform.
```

NOTE: The PROTOBUF_GENERATE_CPP macro & add_executable() or add_library()

```
calls only work properly within the same directory.

=============================================================================
```

PROTOBUF_GENERATE_CPP (public function)

```
SRCS = Variable to define with autogenerated
source files
HDRS = Variable to define with autogenerated
```

```
header files
ARGN = proto files

=============================================================
```

FindPythonInterp
Find python interpreter

This module finds if Python interpreter is installed and determines where the executables are. This code sets the following variables:

```
PYTHONINTERP_FOUND - Was the Python executable found
PYTHON_EXECUTABLE - path to the Python interpreter

PYTHON_VERSION_STRING - Python version found e.g. 2.5.2
PYTHON_VERSION_MAJOR - Python major version found e.g. 2
PYTHON_VERSION_MINOR - Python minor version found e.g. 5
PYTHON_VERSION_PATCH - Python patch version found e.g. 2
```

The Python_ADDITIONAL_VERSIONS variable can be used to specify a list of version numbers that should be taken into account when searching for Python. You need to set this variable before calling find_package(PythonInterp).

FindPythonLibs
Find python libraries

This module finds if Python is installed and determines where the include files and libraries are. It also determines what the name of the library is. This code sets the following variables:

```
PYTHONLIBS_FOUND - have the Python libs been found
PYTHON_LIBRARIES - path to the python library
PYTHON_INCLUDE_PATH - path to where Python.h is found (deprecated)
PYTHON_INCLUDE_DIRS - path to where Python.h is found
PYTHON_DEBUG_LIBRARIES - path to the debug library (deprecated)
PYTHONLIBS_VERSION_STRING - version of the Python libs found (since CMake 2.8.8)
```

The Python_ADDITIONAL_VERSIONS variable can be used to specify a list of version numbers that should be taken into account when searching for Python. You need to set this variable before calling find_package(PythonLibs).

If youd like to specify the installation of Python to use, you should modify the following cache variables:

```
PYTHON_LIBRARY - path to the python library
PYTHON_INCLUDE_DIR - path to where Python.h is found
```

FindQt3
Locate Qt include paths and libraries

This module defines:

```
QT_INCLUDE_DIR - where to find qt.h, etc.
QT_LIBRARIES - the libraries to link against to use Qt.
QT_DEFINITIONS - definitions to use when
compiling code that uses Qt.
QT_FOUND - If false, don't try to use Qt.
QT_VERSION_STRING - the version of Qt found
```

If you need the multithreaded version of Qt, set QT_MT_REQUIRED to TRUE

Also defined, but not for general use are:

```
QT_MOC_EXECUTABLE, where to find the moc tool.
QT_UIC_EXECUTABLE, where to find the uic tool.
```

```
QT_QT_LIBRARY, where to find the Qt library.
QT_QTMAIN_LIBRARY, where to find the qtmain
library. This is only required by Qt3 on Windows.
```

FindQt4

Finding and Using Qt4

This module can be used to find Qt4. The most important issue is that the Qt4 qmake is available via the system path. This qmake is then used to detect basically everything else. This module defines a number of **IMPORTED** targets, macros and variables.

Typical usage could be something like:

```
set(CMAKE_AUTOMOC ON)
set(CMAKE_INCLUDE_CURRENT_DIR ON)
find_package(Qt4 4.4.3 REQUIRED QtGui QtXml)
add_executable(myexe main.cpp)
target_link_libraries(myexe Qt4::QtGui Qt4::QtXml)
```

NOTE:

When using **IMPORTED** targets, the qtmain.lib static library is automatically linked on Windows for **WIN32** executables. To disable that globally, set the **QT4_NO_LINK_QTMAIN** variable before finding Qt4. To disable that for a particular executable, set the **QT4_NO_LINK_QTMAIN** target property to **TRUE** on the executable.

Qt Build Tools

Qt relies on some bundled tools for code generation, such as **moc** for meta-object code generation,``uic`` for widget layout and population, and **rcc** for virtual filesystem content generation. These tools may be automatically invoked by **cmake(1)** if the appropriate conditions are met. See **cmake-qt(7)** for more.

Qt Macros

In some cases it can be necessary or useful to invoke the Qt build tools in a more-manual way. Several macros are available to add targets for such uses.

```
macro QT4_WRAP_CPP(outfiles inputfile ... [TARGET tgt] OPTIONS ...)
create moc code from a list of files containing Qt class with
the Q_OBJECT declaration. Per-directory preprocessor definitions
are also added. If the <tgt> is specified, the
INTERFACE_INCLUDE_DIRECTORIES and INTERFACE_COMPILE_DEFINITIONS from
the <tgt> are passed to moc. Options may be given to moc, such as
those found when executing "moc -help".
```

```
macro QT4_WRAP_UI(outfiles inputfile ... OPTIONS ...)
create code from a list of Qt designer ui files.
Options may be given to uic, such as those found
when executing "uic -help"
```

```
macro QT4_ADD_RESOURCES(outfiles inputfile ... OPTIONS ...)
create code from a list of Qt resource files.
Options may be given to rcc, such as those found
when executing "rcc -help"
```

```
macro QT4_GENERATE_MOC(inputfile outputfile [TARGET tgt])
creates a rule to run moc on infile and create outfile.
Use this if for some reason QT4_WRAP_CPP() isn't appropriate, e.g.
because you need a custom filename for the moc file or something
similar. If the <tgt> is specified, the
INTERFACE_INCLUDE_DIRECTORIES and INTERFACE_COMPILE_DEFINITIONS from
the <tgt> are passed to moc.
```

```
macro QT4_ADD_DBUS_INTERFACE(outfiles interface basename)
Create the interface header and implementation files with the
given basename from the given interface xml file and add it to
the list of sources.

You can pass additional parameters to the qdbusxml2cpp call by setting
properties on the input file:

INCLUDE the given file will be included in the generate interface header

CLASSNAME the generated class is named accordingly

NO_NAMESPACE the generated class is not wrapped in a namespace

macro QT4_ADD_DBUS_INTERFACES(outfiles inputfile ... )
Create the interface header and implementation files
for all listed interface xml files.
The basename will be automatically determined from the name of the xml file.

The source file properties described for QT4_ADD_DBUS_INTERFACE also apply here.

macro QT4_ADD_DBUS_ADAPTOR(outfiles xmlfile parentheader parentclassname [basename] [class
create a dbus adaptor (header and implementation file) from the xml file
describing the interface, and add it to the list of sources. The adaptor
forwards the calls to a parent class, defined in parentheader and named
parentclassname. The name of the generated files will be
<basename>adaptor.{cpp,h} where basename defaults to the basename of the xml file.
If <classname> is provided, then it will be used as the classname of the
adaptor itself.

macro QT4_GENERATE_DBUS_INTERFACE( header [interfacename] OPTIONS ...)
generate the xml interface file from the given header.
If the optional argument interfacename is omitted, the name of the
interface file is constructed from the basename of the header with
the suffix .xml appended.
Options may be given to qdbuscpp2xml, such as those found when executing "qdbuscpp2xml --h

macro QT4_CREATE_TRANSLATION( qm_files directories ... sources ...
ts_files ... OPTIONS ...)
out: qm_files
in: directories sources ts_files
options: flags to pass to lupdate, such as -extensions to specify
extensions for a directory scan.
generates commands to create .ts (vie lupdate) and .qm
(via lrelease) - files from directories and/or sources. The ts files are
created and/or updated in the source tree (unless given with full paths).
The qm files are generated in the build tree.
Updating the translations can be done by adding the qm_files
to the source list of your library/executable, so they are
always updated, or by adding a custom target to control when
they get updated/generated.

macro QT4_ADD_TRANSLATION( qm_files ts_files ... )
out: qm_files
in: ts_files
generates commands to create .qm from .ts - files. The generated
filenames can be found in qm_files. The ts_files
must exist and are not updated in any way.

macro QT4_AUTOMOC(sourcefile1 sourcefile2 ... [TARGET tgt])
```

```
The qt4_automoc macro is obsolete. Use the CMAKE_AUTOMOC feature instead.
This macro is still experimental.
It can be used to have moc automatically handled.
So if you have the files foo.h and foo.cpp, and in foo.h a
a class uses the Q_OBJECT macro, moc has to run on it. If you don't
want to use QT4_WRAP_CPP() (which is reliable and mature), you can insert
#include "foo.moc"
in foo.cpp and then give foo.cpp as argument to QT4_AUTOMOC(). This will the
scan all listed files at cmake-time for such included moc files and if it finds
them cause a rule to be generated to run moc at build time on the
accompanying header file foo.h.
If a source file has the SKIP_AUTOMOC property set it will be ignored by this macro.
If the <tgt> is specified, the INTERFACE_INCLUDE_DIRECTORIES and
INTERFACE_COMPILE_DEFINITIONS from the <tgt> are passed to moc.

function QT4_USE_MODULES( target [link_type] modules...)
This function is obsolete. Use target_link_libraries with IMPORTED targets instead.
Make <target> use the <modules> from Qt. Using a Qt module means
to link to the library, add the relevant include directories for the module,
and add the relevant compiler defines for using the module.
Modules are roughly equivalent to components of Qt4, so usage would be
something like:
qt4_use_modules(myexe Core Gui Declarative)
to use QtCore, QtGui and QtDeclarative. The optional <link_type> argument can
be specified as either LINK_PUBLIC or LINK_PRIVATE to specify the same argument
to the target_link_libraries call.
```

IMPORTED Targets

A particular Qt library may be used by using the corresponding **IMPORTED** target with the **target_link_libraries()** command:

```
target_link_libraries(myexe Qt4::QtGui Qt4::QtXml)
```

Using a target in this way causes :cmake(1)` to use the appropriate include directories and compile definitions for the target when compiling **myexe**.

Targets are aware of their dependencies, so for example it is not necessary to list **Qt4::QtCore** if another Qt library is listed, and it is not necessary to list **Qt4::QtGui** if **Qt4::QtDeclarative** is listed. Targets may be tested for existence in the usual way with the **if(TARGET)** command.

The Qt toolkit may contain both debug and release libraries. **cmake(1)** will choose the appropriate version based on the build configuration.

Qt4::QtCore
: The QtCore target

Qt4::QtGui
: The QtGui target

Qt4::Qt3Support
: The Qt3Support target

Qt4::QtAssistant
: The QtAssistant target

Qt4::QtAssistantClient
: The QtAssistantClient target

Qt4::QAxContainer
: The QAxContainer target (Windows only)

Qt4::QAxServer
The QAxServer target (Windows only)

Qt4::QtDBus
The QtDBus target

Qt4::QtDesigner
The QtDesigner target

Qt4::QtDesignerComponents
The QtDesignerComponents target

Qt4::QtHelp
The QtHelp target

Qt4::QtMotif
The QtMotif target

Qt4::QtMultimedia
The QtMultimedia target

Qt4::QtNetwork
The QtNetwork target

Qt4::QtNsPLugin
The QtNsPLugin target

Qt4::QtOpenGL
The QtOpenGL target

Qt4::QtScript
The QtScript target

Qt4::QtScriptTools
The QtScriptTools target

Qt4::QtSql
The QtSql target

Qt4::QtSvg
The QtSvg target

Qt4::QtTest
The QtTest target

Qt4::QtUiTools
The QtUiTools target

Qt4::QtWebKit
The QtWebKit target

Qt4::QtXml
The QtXml target

Qt4::QtXmlPatterns
The QtXmlPatterns target

Qt4::phonon
The phonon target

Result Variables
Below is a detailed list of variables that FindQt4.cmake sets.

Qt4_FOUND
If false, dont try to use Qt 4.

QT_FOUND
If false, dont try to use Qt. This variable is for compatibility only.

QT4_FOUND
If false, dont try to use Qt 4. This variable is for compatibility only.

QT_VERSION_MAJOR
The major version of Qt found.

QT_VERSION_MINOR
The minor version of Qt found.

QT_VERSION_PATCH
The patch version of Qt found.

FindQt
Searches for all installed versions of Qt.

This should only be used if your project can work with multiple versions of Qt. If not, you should just directly use FindQt4 or FindQt3. If multiple versions of Qt are found on the machine, then The user must set the option DESIRED_QT_VERSION to the version they want to use. If only one version of qt is found on the machine, then the DESIRED_QT_VERSION is set to that version and the matching FindQt3 or FindQt4 module is included. Once the user sets DESIRED_QT_VERSION, then the FindQt3 or FindQt4 module is included.

```
QT_REQUIRED if this is set to TRUE then if CMake can
not find Qt4 or Qt3 an error is raised
and a message is sent to the user.

DESIRED_QT_VERSION OPTION is created
QT4_INSTALLED is set to TRUE if qt4 is found.
QT3_INSTALLED is set to TRUE if qt3 is found.
```

FindQuickTime
Locate QuickTime This module defines QUICKTIME_LIBRARY QUICKTIME_FOUND, if false, do not try to link to gdal QUICKTIME_INCLUDE_DIR, where to find the headers

$QUICKTIME_DIR is an environment variable that would correspond to the

Created by Eric Wing.

FindRTI
Try to find M&S HLA RTI libraries

This module finds if any HLA RTI is installed and locates the standard RTI include files and libraries.

RTI is a simulation infrastructure standardized by IEEE and SISO. It has a well defined C++ API that assures that simulation applications are independent on a particular RTI implementation.

```
http://en.wikipedia.org/wiki/Run-Time_Infrastructure_(simulation)
```

This code sets the following variables:

```
RTI_INCLUDE_DIR = the directory where RTI includes file are found
RTI_LIBRARIES = The libraries to link against to use RTI
RTI_DEFINITIONS = -DRTI_USES_STD_FSTREAM
RTI_FOUND = Set to FALSE if any HLA RTI was not found
```

Report problems to <*certi-devel@nongnu.org*>

FindRuby
Find Ruby

This module finds if Ruby is installed and determines where the include files and libraries are. Ruby 1.8, 1.9, 2.0 and 2.1 are supported.

The minimum required version of Ruby can be specified using the standard syntax, e.g. find_package(Ruby 1.8)

It also determines what the name of the library is. This code sets the following variables:

```
RUBY_EXECUTABLE = full path to the ruby binary
RUBY_INCLUDE_DIRS = include dirs to be used when using the ruby library
RUBY_LIBRARY = full path to the ruby library
RUBY_VERSION = the version of ruby which was found, e.g. "1.8.7"
RUBY_FOUND = set to true if ruby ws found successfully

RUBY_INCLUDE_PATH = same as RUBY_INCLUDE_DIRS, only provided for compatibility reasons, do
```

FindSDL_image

Locate SDL_image library

This module defines:

```
SDL_IMAGE_LIBRARIES, the name of the library to link against
SDL_IMAGE_INCLUDE_DIRS, where to find the headers
SDL_IMAGE_FOUND, if false, do not try to link against
SDL_IMAGE_VERSION_STRING - human-readable string containing the version of SDL_image
```

For backward compatiblity the following variables are also set:

```
SDLIMAGE_LIBRARY (same value as SDL_IMAGE_LIBRARIES)
SDLIMAGE_INCLUDE_DIR (same value as SDL_IMAGE_INCLUDE_DIRS)
SDLIMAGE_FOUND (same value as SDL_IMAGE_FOUND)
```

$SDLDIR is an environment variable that would correspond to the

Created by Eric Wing. This was influenced by the FindSDL.cmake module, but with modifications to recognize OS X frameworks and additional Unix paths (FreeBSD, etc).

FindSDL_mixer

Locate SDL_mixer library

This module defines:

```
SDL_MIXER_LIBRARIES, the name of the library to link against
SDL_MIXER_INCLUDE_DIRS, where to find the headers
SDL_MIXER_FOUND, if false, do not try to link against
SDL_MIXER_VERSION_STRING - human-readable string containing the version of SDL_mixer
```

For backward compatiblity the following variables are also set:

```
SDLMIXER_LIBRARY (same value as SDL_MIXER_LIBRARIES)
SDLMIXER_INCLUDE_DIR (same value as SDL_MIXER_INCLUDE_DIRS)
SDLMIXER_FOUND (same value as SDL_MIXER_FOUND)
```

$SDLDIR is an environment variable that would correspond to the

Created by Eric Wing. This was influenced by the FindSDL.cmake module, but with modifications to recognize OS X frameworks and additional Unix paths (FreeBSD, etc).

FindSDL_net

Locate SDL_net library

This module defines:

```
SDL_NET_LIBRARIES, the name of the library to link against
SDL_NET_INCLUDE_DIRS, where to find the headers
```

```
SDL_NET_FOUND, if false, do not try to link against
SDL_NET_VERSION_STRING - human-readable string containing the version of SDL_net
```

For backward compatiblity the following variables are also set:

```
SDLNET_LIBRARY (same value as SDL_NET_LIBRARIES)
SDLNET_INCLUDE_DIR (same value as SDL_NET_INCLUDE_DIRS)
SDLNET_FOUND (same value as SDL_NET_FOUND)
```

$SDLDIR is an environment variable that would correspond to the

Created by Eric Wing. This was influenced by the FindSDL.cmake module, but with modifications to recognize OS X frameworks and additional Unix paths (FreeBSD, etc).

FindSDL

Locate SDL library

This module defines

```
SDL_LIBRARY, the name of the library to link against
SDL_FOUND, if false, do not try to link to SDL
SDL_INCLUDE_DIR, where to find SDL.h
SDL_VERSION_STRING, human-readable string containing the version of SDL
```

This module responds to the flag:

```
SDL_BUILDING_LIBRARY
If this is defined, then no SDL_main will be linked in because
only applications need main().
Otherwise, it is assumed you are building an application and this
module will attempt to locate and set the proper link flags
as part of the returned SDL_LIBRARY variable.
```

Dont forget to include SDLmain.h and SDLmain.m your project for the OS X framework based version. (Other versions link to -lSDLmain which this module will try to find on your behalf.) Also for OS X, this module will automatically add the -framework Cocoa on your behalf.

Additional Note: If you see an empty SDL_LIBRARY_TEMP in your configuration and no SDL_LIBRARY, it means CMake did not find your SDL library (SDL.dll, libsdl.so, SDL.framework, etc). Set SDL_LIBRARY_TEMP to point to your SDL library, and configure again. Similarly, if you see an empty SDLMAIN_LIBRARY, you should set this value as appropriate. These values are used to generate the final SDL_LIBRARY variable, but when these values are unset, SDL_LIBRARY does not get created.

$SDLDIR is an environment variable that would correspond to the

Modified by Eric Wing. Added code to assist with automated building by using environmental variables and providing a more controlled/consistent search behavior. Added new modifications to recognize OS X frameworks and additional Unix paths (FreeBSD, etc). Also corrected the header search path to follow proper SDL guidelines. Added a search for SDLmain which is needed by some platforms. Added a search for threads which is needed by some platforms. Added needed compile switches for MinGW.

On OSX, this will prefer the Framework version (if found) over others. People will have to manually change the cache values of SDL_LIBRARY to override this selection or set the CMake environment CMAKE_INCLUDE_PATH to modify the search paths.

Note that the header path has changed from SDL/SDL.h to just SDL.h This needed to change because proper SDL convention is #include SDL.h, not <SDL/SDL.h>. This is done for portability reasons because not all systems place things in SDL/ (see FreeBSD).

FindSDL_sound

Locates the SDL_sound library

This module depends on SDL being found and must be called AFTER FindSDL.cmake is called.

This module defines

```
SDL_SOUND_INCLUDE_DIR, where to find SDL_sound.h
SDL_SOUND_FOUND, if false, do not try to link to SDL_sound
SDL_SOUND_LIBRARIES, this contains the list of libraries that you need
to link against. This is a read-only variable and is marked INTERNAL.
SDL_SOUND_EXTRAS, this is an optional variable for you to add your own
flags to SDL_SOUND_LIBRARIES. This is prepended to SDL_SOUND_LIBRARIES.
This is available mostly for cases this module failed to anticipate for
and you must add additional flags. This is marked as ADVANCED.
SDL_SOUND_VERSION_STRING, human-readable string containing the version of SDL_sound
```

This module also defines (but you shouldnt need to use directly)

```
SDL_SOUND_LIBRARY, the name of just the SDL_sound library you would link
against. Use SDL_SOUND_LIBRARIES for you link instructions and not this one.
```

And might define the following as needed

```
MIKMOD_LIBRARY
MODPLUG_LIBRARY
OGG_LIBRARY
VORBIS_LIBRARY
SMPEG_LIBRARY
FLAC_LIBRARY
SPEEX_LIBRARY
```

Typically, you should not use these variables directly, and you should use SDL_SOUND_LIBRARIES which contains SDL_SOUND_LIBRARY and the other audio libraries (if needed) to successfully compile on your system.

Created by Eric Wing. This module is a bit more complicated than the other FindSDL* family modules. The reason is that SDL_sound can be compiled in a large variety of different ways which are independent of platform. SDL_sound may dynamically link against other 3rd party libraries to get additional codec support, such as Ogg Vorbis, SMPEG, ModPlug, MikMod, FLAC, Speex, and potentially others. Under some circumstances which I dont fully understand, there seems to be a requirement that dependent libraries of libraries you use must also be explicitly linked against in order to successfully compile. SDL_sound does not currently have any system in place to know how it was compiled. So this CMake module does the hard work in trying to discover which 3rd party libraries are required for building (if any). This module uses a brute force approach to create a test program that uses SDL_sound, and then tries to build it. If the build fails, it parses the error output for known symbol names to figure out which libraries are needed.

Responds to the $SDLDIR and $SDLSOUNDDIR environmental variable that would correspond to the ./configure --prefix=$SDLDIR used in building SDL.

On OSX, this will prefer the Framework version (if found) over others. People will have to manually change the cache values of SDL_LIBRARY to override this selectionor set the CMake environment CMAKE_INCLUDE_PATH to modify the search paths.

FindSDL_ttf

Locate SDL_ttf library

This module defines:

```
SDL_TTF_LIBRARIES, the name of the library to link against
SDL_TTF_INCLUDE_DIRS, where to find the headers
```

```
SDL_TTF_FOUND, if false, do not try to link against
SDL_TTF_VERSION_STRING - human-readable string containing the version of SDL_ttf
```

For backward compatiblity the following variables are also set:

```
SDLTTF_LIBRARY (same value as SDL_TTF_LIBRARIES)
SDLTTF_INCLUDE_DIR (same value as SDL_TTF_INCLUDE_DIRS)
SDLTTF_FOUND (same value as SDL_TTF_FOUND)
```

$SDLDIR is an environment variable that would correspond to the

Created by Eric Wing. This was influenced by the FindSDL.cmake module, but with modifications to recognize OS X frameworks and additional Unix paths (FreeBSD, etc).

FindSelfPackers

Find upx

This module looks for some executable packers (i.e. software that compress executables or shared libs into on-the-fly self-extracting executables or shared libs. Examples:

```
UPX: http://wildsau.idv.uni-linz.ac.at/mfx/upx.html
```

FindSquish

-- Typical Use

This module can be used to find Squish. Currently Squish versions 3 and 4 are supported.

```
SQUISH_FOUND If false, don't try to use Squish
SQUISH_VERSION The full version of Squish found
SQUISH_VERSION_MAJOR The major version of Squish found
SQUISH_VERSION_MINOR The minor version of Squish found
SQUISH_VERSION_PATCH The patch version of Squish found
```

```
SQUISH_INSTALL_DIR The Squish installation directory (containing bin, lib, etc)
SQUISH_SERVER_EXECUTABLE The squishserver executable
SQUISH_CLIENT_EXECUTABLE The squishrunner executable
```

```
SQUISH_INSTALL_DIR_FOUND Was the install directory found?
SQUISH_SERVER_EXECUTABLE_FOUND Was the server executable found?
SQUISH_CLIENT_EXECUTABLE_FOUND Was the client executable found?
```

It provides the function squish_v4_add_test() for adding a squish test to cmake using Squish 4.x:

```
squish_v4_add_test(cmakeTestName AUT targetName SUITE suiteName TEST squishTestName
[SETTINGSGROUP group] [PRE_COMMAND command] [POST_COMMAND command] )
```

The arguments have the following meaning:

```
cmakeTestName: this will be used as the first argument for add_test()
AUT targetName: the name of the cmake target which will be used as AUT, i.e. the
executable which will be tested.
SUITE suiteName: this is either the full path to the squish suite, or just the
last directory of the suite, i.e. the suite name. In this case
the CMakeLists.txt which calls squish_add_test() must be located
in the parent directory of the suite directory.
TEST squishTestName: the name of the squish test, i.e. the name of the subdirectory
of the test inside the suite directory.
SETTINGSGROUP group: if specified, the given settings group will be used for executing the
If not specified, the groupname will be "CTest_<username>"
PRE_COMMAND command: if specified, the given command will be executed before starting the
POST_COMMAND command: same as PRE_COMMAND, but after the squish test has been executed.
```

```
enable_testing()
```

```
find_package(Squish 4.0)
if (SQUISH_FOUND)
squish_v4_add_test(myTestName AUT myApp SUITE ${CMAKE_SOURCE_DIR}/tests/mySuite TEST someS
endif ()
```

For users of Squish version 3.x the macro squish_v3_add_test() is provided:

```
squish_v3_add_test(testName applicationUnderTest testCase envVars testWrapper)
Use this macro to add a test using Squish 3.x.
```

```
enable_testing()
find_package(Squish)
if (SQUISH_FOUND)
squish_v3_add_test(myTestName myApplication testCase envVars testWrapper)
endif ()
```

macro SQUISH_ADD_TEST(testName applicationUnderTest testCase envVars testWrapper)

```
This is deprecated. Use SQUISH_V3_ADD_TEST() if you are using Squish 3.x instead.
```

FindSubversion

Extract information from a subversion working copy

The module defines the following variables:

```
Subversion_SVN_EXECUTABLE - path to svn command line client
Subversion_VERSION_SVN - version of svn command line client
Subversion_FOUND - true if the command line client was found
SUBVERSION_FOUND - same as Subversion_FOUND, set for compatiblity reasons
```

The minimum required version of Subversion can be specified using the standard syntax, e.g. find_package(Subversion 1.4)

If the command line client executable is found two macros are defined:

```
Subversion_WC_INFO(<dir> <var-prefix>)
Subversion_WC_LOG(<dir> <var-prefix>)
```

Subversion_WC_INFO extracts information of a subversion working copy at a given location. This macro defines the following variables:

```
<var-prefix>_WC_URL - url of the repository (at <dir>)
<var-prefix>_WC_ROOT - root url of the repository
<var-prefix>_WC_REVISION - current revision
<var-prefix>_WC_LAST_CHANGED_AUTHOR - author of last commit
<var-prefix>_WC_LAST_CHANGED_DATE - date of last commit
<var-prefix>_WC_LAST_CHANGED_REV - revision of last commit
<var-prefix>_WC_INFO - output of command `svn info <dir>'
```

Subversion_WC_LOG retrieves the log message of the base revision of a subversion working copy at a given location. This macro defines the variable:

```
<var-prefix>_LAST_CHANGED_LOG - last log of base revision
```

Example usage:

```
find_package(Subversion)
if(SUBVERSION_FOUND)
Subversion_WC_INFO(${PROJECT_SOURCE_DIR} Project)
message("Current revision is ${Project_WC_REVISION}")
Subversion_WC_LOG(${PROJECT_SOURCE_DIR} Project)
message("Last changed log is ${Project_LAST_CHANGED_LOG}")
endif()
```

FindSWIG

Find SWIG

This module finds an installed SWIG. It sets the following variables:

```
SWIG_FOUND - set to true if SWIG is found
SWIG_DIR - the directory where swig is installed
SWIG_EXECUTABLE - the path to the swig executable
SWIG_VERSION - the version number of the swig executable
```

The minimum required version of SWIG can be specified using the standard syntax, e.g. find_package(SWIG 1.1)

All information is collected from the SWIG_EXECUTABLE so the version to be found can be changed from the command line by means of setting SWIG_EXECUTABLE

FindTCL

TK_INTERNAL_PATH was removed.

This module finds if Tcl is installed and determines where the include files and libraries are. It also determines what the name of the library is. This code sets the following variables:

```
TCL_FOUND = Tcl was found
TK_FOUND = Tk was found
TCLTK_FOUND = Tcl and Tk were found
TCL_LIBRARY = path to Tcl library (tcl tcl80)
TCL_INCLUDE_PATH = path to where tcl.h can be found
TCL_TCLSH = path to tclsh binary (tcl tcl80)
TK_LIBRARY = path to Tk library (tk tk80 etc)
TK_INCLUDE_PATH = path to where tk.h can be found
TK_WISH = full path to the wish executable
```

In an effort to remove some clutter and clear up some issues for people who are not necessarily Tcl/Tk gurus/developpers, some variables were moved or removed. Changes compared to CMake 2.4 are:

```
=> they were only useful for people writing Tcl/Tk extensions.
=> these libs are not packaged by default with Tcl/Tk distributions.
Even when Tcl/Tk is built from source, several flavors of debug libs
are created and there is no real reason to pick a single one
specifically (say, amongst tcl84g, tcl84gs, or tcl84sgx).
Let's leave that choice to the user by allowing him to assign
TCL_LIBRARY to any Tcl library, debug or not.
=> this ended up being only a Win32 variable, and there is a lot of
confusion regarding the location of this file in an installed Tcl/Tk
tree anyway (see 8.5 for example). If you need the internal path at
this point it is safer you ask directly where the *source* tree is
and dig from there.
```

FindTclsh

Find tclsh

This module finds if TCL is installed and determines where the include files and libraries are. It also determines what the name of the library is. This code sets the following variables:

```
TCLSH_FOUND = TRUE if tclsh has been found
TCL_TCLSH = the path to the tclsh executable
```

In cygwin, look for the cygwin version first. Dont look for it later to avoid finding the cygwin version on a Win32 build.

FindTclStub

TCL_STUB_LIBRARY_DEBUG and TK_STUB_LIBRARY_DEBUG were removed.

This module finds Tcl stub libraries. It first finds Tcl include files and libraries by calling FindTCL.cmake. How to Use the Tcl Stubs Library:

http://tcl.activestate.com/doc/howto/stubs.html

Using Stub Libraries:

http://safari.oreilly.com/0130385603/ch48lev1sec3

This code sets the following variables:

```
TCL_STUB_LIBRARY = path to Tcl stub library
TK_STUB_LIBRARY = path to Tk stub library
TTK_STUB_LIBRARY = path to ttk stub library
```

In an effort to remove some clutter and clear up some issues for people who are not necessarily Tcl/Tk gurus/developpers, some variables were moved or removed. Changes compared to CMake 2.4 are:

```
=> these libs are not packaged by default with Tcl/Tk distributions.
Even when Tcl/Tk is built from source, several flavors of debug libs
are created and there is no real reason to pick a single one
specifically (say, amongst tclstub84g, tclstub84gs, or tclstub84sgx).
Let's leave that choice to the user by allowing him to assign
TCL_STUB_LIBRARY to any Tcl library, debug or not.
```

FindThreads

This module determines the thread library of the system.

The following variables are set

```
CMAKE_THREAD_LIBS_INIT - the thread library
CMAKE_USE_SPROC_INIT - are we using sproc?
CMAKE_USE_WIN32_THREADS_INIT - using WIN32 threads?
CMAKE_USE_PTHREADS_INIT - are we using pthreads
CMAKE_HP_PTHREADS_INIT - are we using hp pthreads
```

For systems with multiple thread libraries, caller can set

```
CMAKE_THREAD_PREFER_PTHREAD
```

FindTIFF

Find TIFF library

Find the native TIFF includes and library This module defines

```
TIFF_INCLUDE_DIR, where to find tiff.h, etc.
TIFF_LIBRARIES, libraries to link against to use TIFF.
TIFF_FOUND, If false, do not try to use TIFF.
```

also defined, but not for general use are

```
TIFF_LIBRARY, where to find the TIFF library.
```

FindUnixCommands

Find unix commands from cygwin

This module looks for some usual Unix commands.

FindVTK

Find a VTK installation or build tree.

The following variables are set if VTK is found. If VTK is not found, VTK_FOUND is set to

false.

```
VTK_FOUND - Set to true when VTK is found.
VTK_USE_FILE - CMake file to use VTK.
VTK_MAJOR_VERSION - The VTK major version number.
VTK_MINOR_VERSION - The VTK minor version number
(odd non-release).
VTK_BUILD_VERSION - The VTK patch level
(meaningless for odd minor).
VTK_INCLUDE_DIRS - Include directories for VTK
VTK_LIBRARY_DIRS - Link directories for VTK libraries
VTK_KITS - List of VTK kits, in CAPS
(COMMON,IO,) etc.
VTK_LANGUAGES - List of wrapped languages, in CAPS
(TCL, PYHTON,) etc.
```

The following cache entries must be set by the user to locate VTK:

```
VTK_DIR - The directory containing VTKConfig.cmake.
This is either the root of the build tree,
or the lib/vtk directory. This is the
only cache entry.
```

The following variables are set for backward compatibility and should not be used in new code:

```
USE_VTK_FILE - The full path to the UseVTK.cmake file.
This is provided for backward
compatibility. Use VTK_USE_FILE
instead.
```

FindWget

Find wget

This module looks for wget. This module defines the following values:

```
WGET_EXECUTABLE: the full path to the wget tool.
WGET_FOUND: True if wget has been found.
```

FindWish

Find wish installation

This module finds if TCL is installed and determines where the include files and libraries are. It also determines what the name of the library is. This code sets the following variables:

```
TK_WISH = the path to the wish executable
```

if UNIX is defined, then it will look for the cygwin version first

FindwxWidgets

Find a wxWidgets (a.k.a., wxWindows) installation.

This module finds if wxWidgets is installed and selects a default configuration to use. wxWidgets is a modular library. To specify the modules that you will use, you need to name them as components to the package:

find_package(wxWidgets COMPONENTS core base ...)

There are two search branches: a windows style and a unix style. For windows, the following variables are searched for and set to defaults in case of multiple choices. Change them if the defaults are not desired (i.e., these are the only variables you should change to select a configuration):

```
wxWidgets_ROOT_DIR - Base wxWidgets directory
(e.g., C:/wxWidgets-2.6.3).
```

```
wxWidgets_LIB_DIR - Path to wxWidgets libraries
(e.g., C:/wxWidgets-2.6.3/lib/vc_lib).
wxWidgets_CONFIGURATION - Configuration to use
(e.g., msw, mswd, mswu, mswunivud, etc.)
wxWidgets_EXCLUDE_COMMON_LIBRARIES
- Set to TRUE to exclude linking of
commonly required libs (e.g., png tiff
jpeg zlib regex expat).
```

For unix style it uses the wx-config utility. You can select between debug/release, unicode/ansi, universal/non-universal, and static/shared in the QtDialog or ccmake interfaces by turning ON/OFF the following variables:

```
wxWidgets_USE_DEBUG
wxWidgets_USE_UNICODE
wxWidgets_USE_UNIVERSAL
wxWidgets_USE_STATIC
```

There is also a wxWidgets_CONFIG_OPTIONS variable for all other options that need to be passed to the wx-config utility. For example, to use the base toolkit found in the /usr/local path, set the variable (before calling the FIND_PACKAGE command) as such:

```
set(wxWidgets_CONFIG_OPTIONS --toolkit=base --prefix=/usr)
```

The following are set after the configuration is done for both windows and unix style:

```
wxWidgets_FOUND - Set to TRUE if wxWidgets was found.
wxWidgets_INCLUDE_DIRS - Include directories for WIN32
i.e., where to find "wx/wx.h" and
"wx/setup.h"; possibly empty for unices.
wxWidgets_LIBRARIES - Path to the wxWidgets libraries.
wxWidgets_LIBRARY_DIRS - compile time link dirs, useful for
rpath on UNIX. Typically an empty string
in WIN32 environment.
wxWidgets_DEFINITIONS - Contains defines required to compile/link
against WX, e.g. WXUSINGDLL
wxWidgets_DEFINITIONS_DEBUG- Contains defines required to compile/link
against WX debug builds, e.g. __WXDEBUG__
wxWidgets_CXX_FLAGS - Include dirs and compiler flags for
unices, empty on WIN32. Essentially
"`wx-config --cxxflags`".
wxWidgets_USE_FILE - Convenience include file.
```

Sample usage:

```
# Note that for MinGW users the order of libs is important!
find_package(wxWidgets COMPONENTS net gl core base)
if(wxWidgets_FOUND)
include(${wxWidgets_USE_FILE})
# and for each of your dependent executable/library targets:
target_link_libraries(<YourTarget> ${wxWidgets_LIBRARIES})
endif()
```

If wxWidgets is required (i.e., not an optional part):

```
find_package(wxWidgets REQUIRED net gl core base)
include(${wxWidgets_USE_FILE})
# and for each of your dependent executable/library targets:
target_link_libraries(<YourTarget> ${wxWidgets_LIBRARIES})
```

FindwxWindows

Find wxWindows (wxWidgets) installation

This module finds if wxWindows/wxWidgets is installed and determines where the include files and libraries are. It also determines what the name of the library is. Please note this file is DEPRECATED and replaced by FindwxWidgets.cmake. This code sets the following variables:

```
WXWINDOWS_FOUND = system has WxWindows
WXWINDOWS_LIBRARIES = path to the wxWindows libraries
on Unix/Linux with additional
linker flags from
"wx-config --libs"
CMAKE_WXWINDOWS_CXX_FLAGS = Compiler flags for wxWindows,
essentially "`wx-config --cxxflags`"
on Linux
WXWINDOWS_INCLUDE_DIR = where to find "wx/wx.h" and "wx/setup.h"
WXWINDOWS_LINK_DIRECTORIES = link directories, useful for rpath on
Unix
WXWINDOWS_DEFINITIONS = extra defines
```

OPTIONS If you need OpenGL support please

```
set(WXWINDOWS_USE_GL 1)
```

in your CMakeLists.txt *before* you include this file.

```
HAVE_ISYSTEM - true required to replace -I by -isystem on g++
```

For convenience include Use_wxWindows.cmake in your projects CMakeLists.txt using include(${CMAKE_CURRENT_LIST_DIR}/Use_wxWindows.cmake).

USAGE

```
set(WXWINDOWS_USE_GL 1)
find_package(wxWindows)
```

NOTES wxWidgets 2.6.x is supported for monolithic builds e.g. compiled in wx/build/msw dir as:

```
nmake -f makefile.vc BUILD=debug SHARED=0 USE_OPENGL=1 MONOLITHIC=1
```

DEPRECATED

```
CMAKE_WX_CAN_COMPILE
WXWINDOWS_LIBRARY
CMAKE_WX_CXX_FLAGS
WXWINDOWS_INCLUDE_PATH
```

AUTHOR Jan Woetzel <*http://www.mip.informatik.uni-kiel.de/~jw*> (07/2003-01/2006)

FindX11

Find X11 installation

Try to find X11 on UNIX systems. The following values are defined

```
X11_FOUND - True if X11 is available
X11_INCLUDE_DIR - include directories to use X11
X11_LIBRARIES - link against these to use X11
```

and also the following more fine grained variables: Include paths: X11_ICE_INCLUDE_PATH, X11_ICE_LIB, X11_ICE_FOUND

```
X11_SM_INCLUDE_PATH, X11_SM_LIB, X11_SM_FOUND
X11_X11_INCLUDE_PATH, X11_X11_LIB
```

```
X11_Xaccessrules_INCLUDE_PATH, X11_Xaccess_FOUND
X11_Xaccessstr_INCLUDE_PATH, X11_Xaccess_FOUND
X11_Xau_INCLUDE_PATH, X11_Xau_LIB, X11_Xau_FOUND
X11_Xcomposite_INCLUDE_PATH, X11_Xcomposite_LIB, X11_Xcomposite_FOUND
X11_Xcursor_INCLUDE_PATH, X11_Xcursor_LIB, X11_Xcursor_FOUND
X11_Xdamage_INCLUDE_PATH, X11_Xdamage_LIB, X11_Xdamage_FOUND
X11_Xdmcp_INCLUDE_PATH, X11_Xdmcp_LIB, X11_Xdmcp_FOUND
X11_Xext_LIB, X11_Xext_FOUND
X11_dpms_INCLUDE_PATH, (in X11_Xext_LIB), X11_dpms_FOUND
X11_XShm_INCLUDE_PATH, (in X11_Xext_LIB), X11_XShm_FOUND
X11_Xshape_INCLUDE_PATH, (in X11_Xext_LIB), X11_Xshape_FOUND
X11_xf86misc_INCLUDE_PATH, X11_Xxf86misc_LIB, X11_xf86misc_FOUND
X11_xf86vmode_INCLUDE_PATH, X11_Xxf86vm_LIB X11_xf86vmode_FOUND
X11_Xfixes_INCLUDE_PATH, X11_Xfixes_LIB, X11_Xfixes_FOUND
X11_Xft_INCLUDE_PATH, X11_Xft_LIB, X11_Xft_FOUND
X11_Xi_INCLUDE_PATH, X11_Xi_LIB, X11_Xi_FOUND
X11_Xinerama_INCLUDE_PATH, X11_Xinerama_LIB, X11_Xinerama_FOUND
X11_Xinput_INCLUDE_PATH, X11_Xinput_LIB, X11_Xinput_FOUND
X11_Xkb_INCLUDE_PATH, X11_Xkb_FOUND
X11_Xkblib_INCLUDE_PATH, X11_Xkb_FOUND
X11_Xkbfile_INCLUDE_PATH, X11_Xkbfile_LIB, X11_Xkbfile_FOUND
X11_Xmu_INCLUDE_PATH, X11_Xmu_LIB, X11_Xmu_FOUND
X11_Xpm_INCLUDE_PATH, X11_Xpm_LIB, X11_Xpm_FOUND
X11_XTest_INCLUDE_PATH, X11_XTest_LIB, X11_XTest_FOUND
X11_Xrandr_INCLUDE_PATH, X11_Xrandr_LIB, X11_Xrandr_FOUND
X11_Xrender_INCLUDE_PATH, X11_Xrender_LIB, X11_Xrender_FOUND
X11_Xscreensaver_INCLUDE_PATH, X11_Xscreensaver_LIB, X11_Xscreensaver_FOUND
X11_Xt_INCLUDE_PATH, X11_Xt_LIB, X11_Xt_FOUND
X11_Xutil_INCLUDE_PATH, X11_Xutil_FOUND
X11_Xv_INCLUDE_PATH, X11_Xv_LIB, X11_Xv_FOUND
X11_XSync_INCLUDE_PATH, (in X11_Xext_LIB), X11_XSync_FOUND
```

FindXMLRPC

Find xmlrpc

Find the native XMLRPC headers and libraries.

```
XMLRPC_INCLUDE_DIRS - where to find xmlrpc.h, etc.
XMLRPC_LIBRARIES - List of libraries when using xmlrpc.
XMLRPC_FOUND - True if xmlrpc found.
```

XMLRPC modules may be specified as components for this find module. Modules may be listed by running xmlrpc-c-config. Modules include:

```
c++ C++ wrapper code
libwww-client libwww-based client
cgi-server CGI-based server
abyss-server ABYSS-based server
```

Typical usage:

```
find_package(XMLRPC REQUIRED libwww-client)
```

FindZLIB

Find zlib

Find the native ZLIB includes and library. Once done this will define

```
ZLIB_INCLUDE_DIRS - where to find zlib.h, etc.
ZLIB_LIBRARIES - List of libraries when using zlib.
```

```
ZLIB_FOUND - True if zlib found.

ZLIB_VERSION_STRING - The version of zlib found (x.y.z)
ZLIB_VERSION_MAJOR - The major version of zlib
ZLIB_VERSION_MINOR - The minor version of zlib
ZLIB_VERSION_PATCH - The patch version of zlib
ZLIB_VERSION_TWEAK - The tweak version of zlib
```

The following variable are provided for backward compatibility

```
ZLIB_MAJOR_VERSION - The major version of zlib
ZLIB_MINOR_VERSION - The minor version of zlib
ZLIB_PATCH_VERSION - The patch version of zlib
```

An includer may set ZLIB_ROOT to a zlib installation root to tell this module where to look.

FortranCInterface

Fortran/C Interface Detection

This module automatically detects the API by which C and Fortran languages interact. Variables indicate if the mangling is found:

```
FortranCInterface_GLOBAL_FOUND = Global subroutines and functions
FortranCInterface_MODULE_FOUND = Module subroutines and functions
(declared by "MODULE PROCEDURE")
```

A function is provided to generate a C header file containing macros to mangle symbol names:

```
FortranCInterface_HEADER(<file>
[MACRO_NAMESPACE <macro-ns>]
[SYMBOL_NAMESPACE <ns>]
[SYMBOLS [<module>:]<function> ...])
```

It generates in <file> definitions of the following macros:

```
#define FortranCInterface_GLOBAL (name,NAME) ...
#define FortranCInterface_GLOBAL_(name,NAME) ...
#define FortranCInterface_MODULE (mod,name, MOD,NAME) ...
#define FortranCInterface_MODULE_(mod,name, MOD,NAME) ...
```

These macros mangle four categories of Fortran symbols, respectively:

```
- Global symbols without '_': call mysub()
- Global symbols with '_' : call my_sub()
- Module symbols without '_': use mymod; call mysub()
- Module symbols with '_' : use mymod; call my_sub()
```

If mangling for a category is not known, its macro is left undefined. All macros require raw names in both lower case and upper case. The MACRO_NAMESPACE option replaces the default *FortranCInterface* prefix with a given namespace <macro-ns>.

The SYMBOLS option lists symbols to mangle automatically with C preprocessor definitions:

```
<function> ==> #define <ns><function> ...
<module>:<function> ==> #define <ns><module>_<function> ...
```

If the mangling for some symbol is not known then no preprocessor definition is created, and a warning is displayed. The SYMBOL_NAMESPACE option prefixes all preprocessor definitions generated by the SYMBOLS option with a given namespace <ns>.

Example usage:

```
include(FortranCInterface)
FortranCInterface_HEADER(FC.h MACRO_NAMESPACE "FC_")
```

This creates a FC.h header that defines mangling macros FC_GLOBAL(), FC_GLOBAL_(), FC_MODULE(), and FC_MODULE_().

Example usage:

```
include(FortranCInterface)
FortranCInterface_HEADER(FCMangle.h
MACRO_NAMESPACE "FC_"
SYMBOL_NAMESPACE "FC_"
SYMBOLS mysub mymod:my_sub)
```

This creates a FCMangle.h header that defines the same FC_*() mangling macros as the previous example plus preprocessor symbols FC_mysub and FC_mymod_my_sub.

Another function is provided to verify that the Fortran and C/C++ compilers work together:

```
FortranCInterface_VERIFY([CXX] [QUIET])
```

It tests whether a simple test executable using Fortran and C (and C++ when the CXX option is given) compiles and links successfully. The result is stored in the cache entry FortranCInterface_VERIFIED_C (or FortranCInterface_VERIFIED_CXX if CXX is given) as a boolean. If the check fails and QUIET is not given the function terminates with a FATAL_ERROR message describing the problem. The purpose of this check is to stop a build early for incompatible compiler combinations. The test is built in the Release configuration.

FortranCInterface is aware of possible GLOBAL and MODULE manglings for many Fortran compilers, but it also provides an interface to specify new possible manglings. Set the variables

```
FortranCInterface_GLOBAL_SYMBOLS
FortranCInterface_MODULE_SYMBOLS
```

before including FortranCInterface to specify manglings of the symbols MySub, My_Sub, MyModule:MySub, and My_Module:My_Sub. For example, the code:

```
set(FortranCInterface_GLOBAL_SYMBOLS mysub_ my_sub__ MYSUB_)
#
set(FortranCInterface_MODULE_SYMBOLS
__mymodule_MOD_mysub __my_module_MOD_my_sub)
#
include(FortranCInterface)
```

tells FortranCInterface to try given GLOBAL and MODULE manglings. (The carets point at raw symbol names for clarity in this example but are not needed.)

GenerateExportHeader

Function for generation of export macros for libraries

This module provides the function GENERATE_EXPORT_HEADER().

The **GENERATE_EXPORT_HEADER** function can be used to generate a file suitable for preprocessor inclusion which contains EXPORT macros to be used in library classes:

```
GENERATE_EXPORT_HEADER( LIBRARY_TARGET
[BASE_NAME <base_name>]
[EXPORT_MACRO_NAME <export_macro_name>]
[EXPORT_FILE_NAME <export_file_name>]
[DEPRECATED_MACRO_NAME <deprecated_macro_name>]
[NO_EXPORT_MACRO_NAME <no_export_macro_name>]
[STATIC_DEFINE <static_define>]
[NO_DEPRECATED_MACRO_NAME <no_deprecated_macro_name>]
[DEFINE_NO_DEPRECATED]
[PREFIX_NAME <prefix_name>]
```

```
)
```

The target properties **CXX_VISIBILITY_PRESET** and **VISIBILITY_INLINES_HIDDEN** can be used to add the appropriate compile flags for targets. See the documentation of those target properties, and the convenience variables **CMAKE_CXX_VISIBILITY_PRESET** and **CMAKE_VISIBILITY_INLINES_HIDDEN**.

By default **GENERATE_EXPORT_HEADER()** generates macro names in a file name determined by the name of the library. This means that in the simplest case, users of **GenerateExportHeader** will be equivalent to:

```
set(CMAKE_CXX_VISIBILITY_PRESET hidden)
set(CMAKE_VISIBILITY_INLINES_HIDDEN 1)
add_library(somelib someclass.cpp)
generate_export_header(somelib)
install(TARGETS somelib DESTINATION ${LIBRARY_INSTALL_DIR})
install(FILES
someclass.h
${PROJECT_BINARY_DIR}/somelib_export.h DESTINATION ${INCLUDE_INSTALL_DIR}
)
```

And in the ABI header files:

```
#include "somelib_export.h"
class SOMELIB_EXPORT SomeClass {
};
```

The CMake fragment will generate a file in the **${CMAKE_CURRENT_BINARY_DIR}** called **somelib_export.h** containing the macros **SOMELIB_EXPORT**, **SOMELIB_NO_EXPORT**, **SOMELIB_DEPRECATED**, **SOMELIB_DEPRECATED_EXPORT** and **SOMELIB_DEPRECATED_NO_EXPORT**. The resulting file should be installed with other headers in the library.

The **BASE_NAME** argument can be used to override the file name and the names used for the macros:

```
add_library(somelib someclass.cpp)
generate_export_header(somelib
BASE_NAME other_name
)
```

Generates a file called **other_name_export.h** containing the macros **OTHER_NAME_EXPORT**, **OTHER_NAME_NO_EXPORT** and **OTHER_NAME_DEPRECATED** etc.

The **BASE_NAME** may be overridden by specifiying other options in the function. For example:

```
add_library(somelib someclass.cpp)
generate_export_header(somelib
EXPORT_MACRO_NAME OTHER_NAME_EXPORT
)
```

creates the macro **OTHER_NAME_EXPORT** instead of **SOMELIB_EXPORT**, but other macros and the generated file name is as default:

```
add_library(somelib someclass.cpp)
generate_export_header(somelib
DEPRECATED_MACRO_NAME KDE_DEPRECATED
)
```

creates the macro **KDE_DEPRECATED** instead of **SOMELIB_DEPRECATED**.

If **LIBRARY_TARGET** is a static library, macros are defined without values.

If the same sources are used to create both a shared and a static library, the uppercased symbol **${BASE_NAME}_STATIC_DEFINE** should be used when building the static library:

```
add_library(shared_variant SHARED ${lib_SRCS})
add_library(static_variant ${lib_SRCS})
generate_export_header(shared_variant BASE_NAME libshared_and_static)
set_target_properties(static_variant PROPERTIES
COMPILE_FLAGS -DLIBSHARED_AND_STATIC_STATIC_DEFINE)
```

This will cause the export macros to expand to nothing when building the static library.

If **DEFINE_NO_DEPRECATED** is specified, then a macro **${BASE_NAME}_NO_DEPRECATED** will be defined This macro can be used to remove deprecated code from preprocessor output:

```
option(EXCLUDE_DEPRECATED "Exclude deprecated parts of the library" FALSE)
if (EXCLUDE_DEPRECATED)
set(NO_BUILD_DEPRECATED DEFINE_NO_DEPRECATED)
endif()
generate_export_header(somelib ${NO_BUILD_DEPRECATED})
```

And then in somelib:

```
class SOMELIB_EXPORT SomeClass
{
public:
#ifndef SOMELIB_NO_DEPRECATED
SOMELIB_DEPRECATED void oldMethod();
#endif
};
```

```
#ifndef SOMELIB_NO_DEPRECATED
void SomeClass::oldMethod() { }
#endif
```

If **PREFIX_NAME** is specified, the argument will be used as a prefix to all generated macros.

For example:

```
generate_export_header(somelib PREFIX_NAME VTK_)
```

Generates the macros **VTK_SOMELIB_EXPORT** etc.

```
ADD_COMPILER_EXPORT_FLAGS( [<output_variable>] )
```

The **ADD_COMPILER_EXPORT_FLAGS** function adds **-fvisibility=hidden** to **CMAKE_CXX_FLAGS** if supported, and is a no-op on Windows which does not need extra compiler flags for exporting support. You may optionally pass a single argument to **ADD_COMPILER_EXPORT_FLAGS** that will be populated with the **CXX_FLAGS** required to enable visibility support for the compiler/architecture in use.

This function is deprecated. Set the target properties **CXX_VISIBILITY_PRESET** and **VISIBILITY_INLINES_HIDDEN** instead.

GetPrerequisites

Functions to analyze and list executable file prerequisites.

This module provides functions to list the .dll, .dylib or .so files that an executable or shared library file depends on. (Its prerequisites.)

It uses various tools to obtain the list of required shared library files:

```
dumpbin (Windows)
objdump (MinGW on Windows)
ldd (Linux/Unix)
otool (Mac OSX)
```

The following functions are provided by this module:

```
get_prerequisites
list_prerequisites
list_prerequisites_by_glob
gp_append_unique
is_file_executable
gp_item_default_embedded_path
(projects can override with gp_item_default_embedded_path_override)
gp_resolve_item
(projects can override with gp_resolve_item_override)
gp_resolved_file_type
(projects can override with gp_resolved_file_type_override)
gp_file_type
```

Requires CMake 2.6 or greater because it uses function, break, return and PARENT_SCOPE.

```
GET_PREREQUISITES(<target> <prerequisites_var> <exclude_system> <recurse>
<exepath> <dirs>)
```

Get the list of shared library files required by <target>. The list in the variable named <prerequisites_var> should be empty on first entry to this function. On exit, <prerequisites_var> will contain the list of required shared library files.

<target> is the full path to an executable file. <prerequisites_var> is the name of a CMake variable to contain the results. <exclude_system> m ust be 0 or 1 indicating whether to include or exclude system prerequisites. If <recurse> is set to 1 all prerequisites will be found recursively, if set to 0 only direct prerequisites are listed. <exepath> is the path to the top level executable used for @executable_path replacment on the Mac. <dirs> is a list of paths where libraries might be found: these paths are searched first when a target without any path info is given. Then standard system locations are also searched: PATH, Framework locations, /usr/lib...

```
LIST_PREREQUISITES(<target> [<recurse> [<exclude_system> [<verbose>]]])
```

Print a message listing the prerequisites of <target>.

<target> is the name of a shared library or executable target or the full path to a shared library or executable file. If <recurse> is set to 1 all prerequisites will be found recursively, if set to 0 only direct prerequisites are listed. <exclude_system> must be 0 or 1 indicating whether to include or exclude system prerequisites. With <verbose> set to 0 only the full path names of the prerequisites are printed, set to 1 extra informatin will be displayed.

```
LIST_PREREQUISITES_BY_GLOB(<glob_arg> <glob_exp>)
```

Print the prerequisites of shared library and executable files matching a globbing pattern. <glob_arg> is GLOB or GLOB_RECURSE and <glob_exp> is a globbing expression used with file(GLOB or file(GLOB_RECURSE to retrieve a list of matching files. If a matching file is executable, its prerequisites are listed.

Any additional (optional) arguments provided are passed along as the optional arguments to the list_prerequisites calls.

```
GP_APPEND_UNIQUE(<list_var> <value>)
```

Append <value> to the list variable <list_var> only if the value is not already in the list.

```
IS_FILE_EXECUTABLE(<file> <result_var>)
```

Return 1 in <result_var> if <file> is a binary executable, 0 otherwise.

```
GP_ITEM_DEFAULT_EMBEDDED_PATH(<item> <default_embedded_path_var>)
```

Return the path that others should refer to the item by when the item is embedded inside a bundle.

Override on a per-project basis by providing a project-specific gp_item_default_embedded_path_override function.

```
GP_RESOLVE_ITEM(<context> <item> <exepath> <dirs> <resolved_item_var>)
```

Resolve an item into an existing full path file.

Override on a per-project basis by providing a project-specific gp_resolve_item_override function.

```
GP_RESOLVED_FILE_TYPE(<original_file> <file> <exepath> <dirs> <type_var>)
```

Return the type of <file> with respect to <original_file>. String describing type of prerequisite is returned in variable named <type_var>.

Use <exepath> and <dirs> if necessary to resolve non-absolute <file> values -- but only for non-embedded items.

Possible types are:

```
system
local
embedded
other
```

Override on a per-project basis by providing a project-specific gp_resolved_file_type_override function.

```
GP_FILE_TYPE(<original_file> <file> <type_var>)
```

Return the type of <file> with respect to <original_file>. String describing type of prerequisite is returned in variable named <type_var>.

Possible types are:

```
system
local
embedded
other
```

GNUInstallDirs

Define GNU standard installation directories

Provides install directory variables as defined for GNU software:

http://www.gnu.org/prep/standards/html_node/Directory-Variables.html

Inclusion of this module defines the following variables:

```
CMAKE_INSTALL_<dir> - destination for files of a given type
CMAKE_INSTALL_FULL_<dir> - corresponding absolute path
```

where <dir> is one of:

```
BINDIR - user executables (bin)
SBINDIR - system admin executables (sbin)
LIBEXECDIR - program executables (libexec)
SYSCONFDIR - read-only single-machine data (etc)
SHAREDSTATEDIR - modifiable architecture-independent data (com)
LOCALSTATEDIR - modifiable single-machine data (var)
```

```
LIBDIR - object code libraries (lib or lib64 or lib/<multiarch-tuple> on Debian)
INCLUDEDIR - C header files (include)
OLDINCLUDEDIR - C header files for non-gcc (/usr/include)
DATAROOTDIR - read-only architecture-independent data root (share)
DATADIR - read-only architecture-independent data (DATAROOTDIR)
INFODIR - info documentation (DATAROOTDIR/info)
LOCALEDIR - locale-dependent data (DATAROOTDIR/locale)
MANDIR - man documentation (DATAROOTDIR/man)
DOCDIR - documentation root (DATAROOTDIR/doc/PROJECT_NAME)
```

Each CMAKE_INSTALL_<dir> value may be passed to the DESTINATION options of install() commands for the corresponding file type. If the includer does not define a value the above-shown default will be used and the value will appear in the cache for editing by the user. Each CMAKE_INSTALL_FULL_<dir> value contains an absolute path constructed from the corresponding destination by prepending (if necessary) the value of CMAKE_INSTALL_PREFIX.

InstallRequiredSystemLibraries

By including this file, all library files listed in the variable CMAKE_INSTALL_SYSTEM_RUNTIME_LIBS will be installed with install(PROGRAMS ...) into bin for WIN32 and lib for non-WIN32. If CMAKE_INSTALL_SYSTEM_RUNTIME_LIBS_SKIP is set to TRUE before including this file, then the INSTALL command is not called. The user can use the variable CMAKE_INSTALL_SYSTEM_RUNTIME_LIBS to use a custom install command and install them however they want. If it is the MSVC compiler, then the microsoft run time libraries will be found and automatically added to the CMAKE_INSTALL_SYSTEM_RUNTIME_LIBS, and installed. If CMAKE_INSTALL_DEBUG_LIBRARIES is set and it is the MSVC compiler, then the debug libraries are installed when available. If CMAKE_INSTALL_DEBUG_LIBRARIES_ONLY is set then only the debug libraries are installed when both debug and release are available. If CMAKE_INSTALL_MFC_LIBRARIES is set then the MFC run time libraries are installed as well as the CRT run time libraries. If CMAKE_INSTALL_SYSTEM_RUNTIME_DESTINATION is set then the libraries are installed to that directory rather than the default. If CMAKE_INSTALL_SYSTEM_RUNTIME_LIBS_NO_WARNINGS is NOT set, then this file warns about required files that do not exist. You can set this variable to ON before including this file to avoid the warning. For example, the Visual Studio Express editions do not include the redistributable files, so if you include this file on a machine with only VS Express installed, youll get the warning.

MacroAddFileDependencies

MACRO_ADD_FILE_DEPENDENCIES(<_file> depend_files...)

Using the macro MACRO_ADD_FILE_DEPENDENCIES() is discouraged. There are usually better ways to specify the correct dependencies.

MACRO_ADD_FILE_DEPENDENCIES(<_file> depend_files...) is just a convenience wrapper around the OBJECT_DEPENDS source file property. You can just use set_property(SOURCE <file> APPEND PROPERTY OBJECT_DEPENDS depend_files) instead.

ProcessorCount

ProcessorCount(var)

Determine the number of processors/cores and save value in ${var}

Sets the variable named ${var} to the number of physical cores available on the machine if the information can be determined. Otherwise it is set to 0. Currently this functionality is implemented for AIX, cygwin, FreeBSD, HPUX, IRIX, Linux, Mac OS X, QNX, Sun and Windows.

This function is guaranteed to return a positive integer (>=1) if it succeeds. It returns 0 if theres a problem determining the processor count.

Example use, in a ctest -S dashboard script:

```
include(ProcessorCount)
ProcessorCount(N)
if(NOT N EQUAL 0)
set(CTEST_BUILD_FLAGS -j${N})
set(ctest_test_args ${ctest_test_args} PARALLEL_LEVEL ${N})
endif()
```

This function is intended to offer an approximation of the value of the number of compute cores available on the current machine, such that you may use that value for parallel building and parallel testing. It is meant to help utilize as much of the machine as seems reasonable. Of course, knowledge of what else might be running on the machine simultaneously should be used when deciding whether to request a machines full capacity all for yourself.

SelectLibraryConfigurations

select_library_configurations(basename)

This macro takes a library base name as an argument, and will choose good values for basename_LIBRARY, basename_LIBRARIES, basename_LIBRARY_DEBUG, and basename_LIBRARY_RELEASE depending on what has been found and set. If only basename_LIBRARY_RELEASE is defined, basename_LIBRARY will be set to the release value, and basename_LIBRARY_DEBUG will be set to basename_LIBRARY_DEBUG-NOTFOUND. If only basename_LIBRARY_DEBUG is defined, then basename_LIBRARY will take the debug value, and basename_LIBRARY_RELEASE will be set to basename_LIBRARY_RELEASE-NOTFOUND.

If the generator supports configuration types, then basename_LIBRARY and basename_LIBRARIES will be set with debug and optimized flags specifying the library to be used for the given configuration. If no build type has been set or the generator in use does not support configuration types, then basename_LIBRARY and basename_LIBRARIES will take only the release value, or the debug value if the release one is not set.

SquishTestScript

This script launches a GUI test using Squish. You should not call the script directly; instead, you should access it via the SQUISH_ADD_TEST macro that is defined in FindSquish.cmake.

This script starts the Squish server, launches the test on the client, and finally stops the squish server. If any of these steps fail (including if the tests do not pass) then a fatal error is raised.

TestBigEndian

Define macro to determine endian type

Check if the system is big endian or little endian

```
TEST_BIG_ENDIAN(VARIABLE)
VARIABLE - variable to store the result to
```

TestCXXAcceptsFlag

Deprecated. See **CheckCXXCompilerFlag**.

Check if the CXX compiler accepts a flag.

```
CHECK_CXX_ACCEPTS_FLAG(<flags> <variable>)
```

<flags>
: the flags to try

<variable>
: variable to store the result

TestForANSIForScope

Check for ANSI for scope support

Check if the compiler restricts the scope of variables declared in a for-init-statement to the loop

body.

```
CMAKE_NO_ANSI_FOR_SCOPE - holds result
```

TestForANSIStreamHeaders
Test for compiler support of ANSI stream headers iostream, etc.

check if the compiler supports the standard ANSI iostream header (without the .h)

```
CMAKE_NO_ANSI_STREAM_HEADERS - defined by the results
```

TestForSSTREAM
Test for compiler support of ANSI sstream header

check if the compiler supports the standard ANSI sstream header

```
CMAKE_NO_ANSI_STRING_STREAM - defined by the results
```

TestForSTDNamespace
Test for std:: namespace support

check if the compiler supports std:: on stl classes

```
CMAKE_NO_STD_NAMESPACE - defined by the results
```

UseEcos
This module defines variables and macros required to build eCos application.

This file contains the following macros: ECOS_ADD_INCLUDE_DIRECTORIES() - add the eCos include dirs ECOS_ADD_EXECUTABLE(name source1 ... sourceN) - create an eCos executable ECOS_ADJUST_DIRECTORY(VAR source1 ... sourceN) - adjusts the path of the source files and puts the result into VAR

Macros for selecting the toolchain: ECOS_USE_ARM_ELF_TOOLS() - enable the ARM ELF toolchain for the directory where it is called ECOS_USE_I386_ELF_TOOLS() - enable the i386 ELF toolchain for the directory where it is called ECOS_USE_PPC_EABI_TOOLS() - enable the PowerPC toolchain for the directory where it is called

It contains the following variables: ECOS_DEFINITIONS ECOSCONFIG_EXECUTABLE ECOS_CONFIG_FILE - defaults to ecos.ecc, if your eCos configuration file has a different name, adjust this variable for internal use only:

```
ECOS_ADD_TARGET_LIB
```

UseJavaClassFilelist
This script create a list of compiled Java class files to be added to a jar file. This avoids including cmake files which get created in the binary directory.

UseJava
Use Module for Java

This file provides functions for Java. It is assumed that FindJava.cmake has already been loaded. See FindJava.cmake for information on how to load Java into your CMake project.

```
add_jar(target_name
[SOURCES] source1 [source2 ...] [resource1 ...]
[INCLUDE_JARS jar1 [jar2 ...]]
[ENTRY_POINT entry]
[VERSION version]
[OUTPUT_NAME name]
[OUTPUT_DIR dir]
)
```

This command creates a <target_name>.jar. It compiles the given source files (source) and adds the given resource files (resource) to the jar file. If only resource files are given then just a jar file

is created. The list of include jars are added to the classpath when compiling the java sources and also to the dependencies of the target. INCLUDE_JARS also accepts other target names created by add_jar. For backwards compatibility, jar files listed as sources are ignored (as they have been since the first version of this module).

The default OUTPUT_DIR can also be changed by setting the variable CMAKE_JAVA_TARGET_OUTPUT_DIR.

Additional instructions:

```
To add compile flags to the target you can set these flags with
the following variable:

set(CMAKE_JAVA_COMPILE_FLAGS -nowarn)

To add a path or a jar file to the class path you can do this
with the CMAKE_JAVA_INCLUDE_PATH variable.

set(CMAKE_JAVA_INCLUDE_PATH /usr/share/java/shibboleet.jar)

To use a different output name for the target you can set it with:

add_jar(foobar foobar.java OUTPUT_NAME shibboleet.jar)

To use a different output directory than CMAKE_CURRENT_BINARY_DIR
you can set it with:

add_jar(foobar foobar.java OUTPUT_DIR ${PROJECT_BINARY_DIR}/bin)

To define an entry point in your jar you can set it with the ENTRY_POINT
named argument:

add_jar(example ENTRY_POINT com/examples/MyProject/Main)

To define a custom manifest for the jar, you can set it with the manifest
named argument:

add_jar(example MANIFEST /path/to/manifest)

To add a VERSION to the target output name you can set it using
the VERSION named argument to add_jar. This will create a jar file with the
name shibboleet-1.0.0.jar and will create a symlink shibboleet.jar
pointing to the jar with the version information.

add_jar(shibboleet shibbotleet.java VERSION 1.2.0)

If the target is a JNI library, utilize the following commands to
create a JNI symbolic link:

set(CMAKE_JNI_TARGET TRUE)
add_jar(shibboleet shibbotleet.java VERSION 1.2.0)
install_jar(shibboleet ${LIB_INSTALL_DIR}/shibboleet)
install_jni_symlink(shibboleet ${JAVA_LIB_INSTALL_DIR})

If a single target needs to produce more than one jar from its
java source code, to prevent the accumulation of duplicate class
files in subsequent jars, set/reset CMAKE_JAR_CLASSES_PREFIX prior
to calling the add_jar() function:

set(CMAKE_JAR_CLASSES_PREFIX com/redhat/foo)
add_jar(foo foo.java)

set(CMAKE_JAR_CLASSES_PREFIX com/redhat/bar)
add_jar(bar bar.java)
```

Target Properties:

```
The add_jar() functions sets some target properties. You can get these
properties with the
get_property(TARGET <target_name> PROPERTY <propery_name>)
command.

INSTALL_FILES The files which should be installed. This is used by
install_jar().
JNI_SYMLINK The JNI symlink which should be installed.
This is used by install_jni_symlink().
JAR_FILE The location of the jar file so that you can include
it.
CLASS_DIR The directory where the class files can be found. For
example to use them with javah.

find_jar(<VAR>
name | NAMES name1 [name2 ...]
[PATHS path1 [path2 ... ENV var]]
[VERSIONS version1 [version2]]
[DOC "cache documentation string"]
)
```

This command is used to find a full path to the named jar. A cache entry named by <VAR> is created to stor the result of this command. If the full path to a jar is found the result is stored in the variable and the search will not repeated unless the variable is cleared. If nothing is found, the result will be <VAR>-NOTFOUND, and the search will be attempted again next time find_jar is invoked with the same variable. The name of the full path to a file that is searched for is specified by the names listed after NAMES argument. Additional search locations can be specified after the PATHS argument. If you require special a version of a jar file you can specify it with the VERSIONS argument. The argument after DOC will be used for the documentation string in the cache.

```
install_jar(TARGET_NAME DESTINATION)
```

This command installs the TARGET_NAME files to the given DESTINATION. It should be called in the same scope as add_jar() or it will fail.

```
install_jni_symlink(TARGET_NAME DESTINATION)
```

This command installs the TARGET_NAME JNI symlinks to the given DESTINATION. It should be called in the same scope as add_jar() or it will fail.

```
create_javadoc(<VAR>
PACKAGES pkg1 [pkg2 ...]
[SOURCEPATH <sourcepath>]
[CLASSPATH <classpath>]
[INSTALLPATH <install path>]
[DOCTITLE "the documentation title"]
[WINDOWTITLE "the title of the document"]
[AUTHOR TRUE|FALSE]
[USE TRUE|FALSE]
[VERSION TRUE|FALSE]
)
```

Create java documentation based on files or packages. For more details please read the javadoc manpage.

There are two main signatures for create_javadoc. The first signature works with package names on a path with source files:

```
Example:
create_javadoc(my_example_doc
PACKAGES com.exmaple.foo com.example.bar
SOURCEPATH "${CMAKE_CURRENT_SOURCE_DIR}"
CLASSPATH ${CMAKE_JAVA_INCLUDE_PATH}
WINDOWTITLE "My example"
DOCTITLE "<h1>My example</h1>"
AUTHOR TRUE
USE TRUE
VERSION TRUE
)
```

The second signature for create_javadoc works on a given list of files.

```
create_javadoc(<VAR>
FILES file1 [file2 ...]
[CLASSPATH <classpath>]
[INSTALLPATH <install path>]
[DOCTITLE "the documentation title"]
[WINDOWTITLE "the title of the document"]
[AUTHOR TRUE|FALSE]
[USE TRUE|FALSE]
[VERSION TRUE|FALSE]
)
```

Example:

```
create_javadoc(my_example_doc
FILES ${example_SRCS}
CLASSPATH ${CMAKE_JAVA_INCLUDE_PATH}
WINDOWTITLE "My example"
DOCTITLE "<h1>My example</h1>"
AUTHOR TRUE
USE TRUE
VERSION TRUE
)
```

Both signatures share most of the options. These options are the same as what you can find in the javadoc manpage. Please look at the manpage for CLASSPATH, DOCTITLE, WINDOWTITLE, AUTHOR, USE and VERSION.

The documentation will be by default installed to

```
${CMAKE_INSTALL_PREFIX}/share/javadoc/<VAR>
```

if you dont set the INSTALLPATH.

UseJavaSymlinks
Helper script for UseJava.cmake

UsePkgConfig
Obsolete pkg-config module for CMake, use FindPkgConfig instead.

This module defines the following macro:

PKGCONFIG(package includedir libdir linkflags cflags)

Calling PKGCONFIG will fill the desired information into the 4 given arguments, e.g. PKGCONFIG(libart-2.0 LIBART_INCLUDE_DIR LIBART_LINK_DIR LIBART_LINK_FLAGS LIBART_CFLAGS) if pkg-config was NOT found or the specified software package doesnt exist, the variable will be empty when the function returns, otherwise they will contain the respective

information

UseSWIG

SWIG module for CMake

Defines the following macros:

```
SWIG_ADD_MODULE(name language [ files ])
- Define swig module with given name and specified language
SWIG_LINK_LIBRARIES(name [ libraries ])
- Link libraries to swig module
```

All other macros are for internal use only. To get the actual name of the swig module, use: ${SWIG_MODULE_${name}_REAL_NAME}. Set Source files properties such as CPLUSPLUS and SWIG_FLAGS to specify special behavior of SWIG. Also global CMAKE_SWIG_FLAGS can be used to add special flags to all swig calls. Another special variable is CMAKE_SWIG_OUTDIR, it allows one to specify where to write all the swig generated module (swig -outdir option) The name-specific variable SWIG_MODULE_<name>_EXTRA_DEPS may be used to specify extra dependencies for the generated modules. If the source file generated by swig need some special flag you can use:

```
set_source_files_properties( ${swig_generated_file_fullname}
PROPERTIES COMPILE_FLAGS "-bla")
```

UsewxWidgets

Convenience include for using wxWidgets library.

Determines if wxWidgets was FOUND and sets the appropriate libs, incdirs, flags, etc. INCLUDE_DIRECTORIES and LINK_DIRECTORIES are called.

USAGE

```
# Note that for MinGW users the order of libs is important!
find_package(wxWidgets REQUIRED net gl core base)
include(${wxWidgets_USE_FILE})
# and for each of your dependent executable/library targets:
target_link_libraries(<YourTarget> ${wxWidgets_LIBRARIES})
```

DEPRECATED

```
LINK_LIBRARIES is not called in favor of adding dependencies per target.
```

AUTHOR

```
Jan Woetzel <jw -at- mip.informatik.uni-kiel.de>
```

Use_wxWindows

This convenience include finds if wxWindows is installed and set the appropriate libs, incdirs, flags etc. author Jan Woetzel <jw -at- mip.informatik.uni-kiel.de> (07/2003)

USAGE:

```
just include Use_wxWindows.cmake
in your projects CMakeLists.txt
```

include(${CMAKE_MODULE_PATH}/Use_wxWindows.cmake)

```
if you are sure you need GL then
```

set(WXWINDOWS_USE_GL 1)

```
*before* you include this file.
```

WriteBasicConfigVersionFile

`WRITE_BASIC_CONFIG_VERSION_FILE( filename [VERSION major.minor.patch] COMPATIBILITY (AnyNe`

Deprecated, see WRITE_BASIC_PACKAGE_VERSION_FILE(), it is identical.

COPYRIGHT