ecBuild

Release 3.6.2

unknown

May 07, 2021

CONTENTS

CHAPTER

ONE

ECBUILD MACROS

1.1 ecbuild_add_c_flags

Add C compiler flags to CMAKE_C_FLAGS only if supported by the compiler.

```
ecbuild_add_c_flags( <flag1> [ <flag2> ... ]
                     [ BUILD <build> ]
                     [ NAME <name> ]
                     [ NO_FAIL ] )
```

1.1.1 Options

BUILD [optional] add flags to `CMAKE_C_FLAGS_<build>` instead of `CMAKE_C_FLAGS`

NAME [optional] name of the check (if omitted, checks are enumerated)

NO_FAIL [optional] do not fail if the flag cannot be added

1.2 ecbuild_add_cxx_flags

Add C++ compiler flags to CMAKE_CXX_FLAGS only if supported by compiler.

```
ecbuild_add_cxx_flags( <flag1> [ <flag2> ... ]
                       [ BUILD <build> ]
                       [ NAME <name> ]
                       [ NO_FAIL ] )
```

1.2.1 Options

BUILD [optional] add flags to `CMAKE_CXX_FLAGS_<build>` instead of `CMAKE_CXX_FLAGS`

NAME [optional] name of the check (if omitted, checks are enumerated)

NO_FAIL [optional] do not fail if the flag cannot be added

1.3 ecbuild_add_executable

Add an executable with a given list of source files.

```
ecbuild_add_executable( TARGET <name>
                        SOURCES <source1> [<source2> ...]
                        [ SOURCES_GLOB <glob1> [<glob2> ...] ]
                        [ SOURCES_EXCLUDE_REGEX <regex1> [<regex2> ...] ]
                        [ OBJECTS <obj1> [<obj2> ...] ]
                        [ TEMPLATES <template1> [<template2> ...] ]
                        [ LIBS <library1> [<library2> ...] ]
                        [ INCLUDES <path1> [<path2> ...] ]
                        [ DEFINITIONS <definition1> [<definition2> ...] ]
                        [ PERSISTENT <file1> [<file2> ...] ]
                        [ GENERATED <file1> [<file2> ...] ]
                        [ DEPENDS <target1> [<target2> ...] ]
                        [ CONDITION <condition> ]
                        [ PROPERTIES <prop1> <val1> [<prop2> <val2> ...] ]
                        [ NOINSTALL ]
                        [ VERSION <version> | AUTO_VERSION ]
                        [ CFLAGS <flag1> [<flag2> ...] ]
                        [ CXXFLAGS <flag1> [<flag2> ...] ]
                        [ FFLAGS <flag1> [<flag2> ...] ]
                        [ LINKER_LANGUAGE <lang> ]
                        [ OUTPUT_NAME <name> ] )
```

1.3.1 Options

TARGET [required] target name

SOURCES [required] list of source files

SOURCES_GLOB [optional] search pattern to find source files to compile (note: not recommend according to CMake guidelines) it is usually better to explicitly list the source files in the CMakeList.txt

SOURCES_EXCLUDE_REGEX [optional] search pattern to exclude source files from compilation, applies o the results of SOURCES_GLOB

OBJECTS [optional] list of object libraries to add to this target

TEMPLATES [optional] list of files specified as SOURCES which are not to be compiled separately (these are commonly template implementation files included in a header)

LIBS [optional] list of libraries to link against (CMake targets or external libraries)

INCLUDES [optional] list of paths to add to include directories

DEFINITIONS [optional] list of definitions to add to preprocessor defines

PERSISTENT [optional] list of persistent layer object files

GENERATED [optional] list of files to mark as generated (sets GENERATED source file property)

DEPENDS [optional] list of targets to be built before this target

CONDITION [optional] conditional expression which must evaluate to true for this target to be built (must be valid in a CMake `if` statement)

PROPERTIES [optional] custom properties to set on the target

NOINSTALL [optional] do not install the executable

VERSION [optional, AUTO_VERSION or LIBS_VERSION is used if not specified] version to use as executable version

AUTO_VERSION [optional, ignored if VERSION is specified] automatically version the executable with the package version

CFLAGS [optional] list of C compiler flags to use for all C source files

See usage note below.

CXXFLAGS [optional] list of C++ compiler flags to use for all C++ source files

See usage note below.

FFLAGS [optional] list of Fortran compiler flags to use for all Fortran source files

See usage note below.

LINKER_LANGUAGE [optional] sets the LINKER_LANGUAGE property on the target

OUTPUT_NAME [optional] sets the OUTPUT_NAME property on the target

1.3.2 Usage

The `CFLAGS`, `CXXFLAGS` and `FFLAGS` options apply the given compiler flags to all C, C++ and Fortran sources passed to this command, respectively. If any two `ecbuild_add_executable`, `ecbuild_add_library` or `ecbuild_add_test` commands are passed the *same* source file and each sets a different value for the compiler flags to be applied to that file (including when one command adds flags and another adds none), then the two commands will be in conflict and the result may not be as expected.

For this reason it is recommended not to use the `*FLAGS` options when multiple targets share the same source files, unless the exact same flags are applied to those sources by each relevant command.

Care should also be taken to ensure that these commands are not passed source files which are not required to build the target, if those sources are also passed to other commands which set different compiler flags.

1.4 ecbuild_add_fortran_flags

Add Fortran compiler flags to CMAKE_Fortran_FLAGS only if supported by the compiler.

```
ecbuild_add_fortran_flags( <flag1> [ <flag2> ... ]
                           [ BUILD <build> ]
                           [ NAME <name> ]
                           [ NO_FAIL ] )
```

1.4.1 Options

BUILD [optional] add flags to `CMAKE_Fortran_FLAGS_<build>` instead of `CMAKE_Fortran_FLAGS`

NAME [optional] name of the check (if omitted, checks are enumerated)

NO_FAIL [optional] do not fail if the flag cannot be added

1.5 ecbuild_add_lang_flags

This is mostly an internal function of ecbuild, wrapped by the macros ecbuild_add_c_flags, ecbuild_add_cxx_flags and ecbuild_add_fortran_flags.

Add compiler flags to the CMAKE_${lang}_FLAGS only if supported by compiler.

```
ecbuild_add_lang_flags( <flag1> [ <flag2> ... ]
                        LANG [C|CXX|Fortran]
                        [ BUILD <build> ]
                        [ NAME <name> ]
                        [ NO_FAIL ] )
```

1.5.1 Options

LANG: define the language to add the flag too

BUILD [optional] add flags to *CMAKE_${lang}_FLAGS_<build>* instead of *CMAKE_${lang}_FLAGS*

NAME [optional] name of the check (if omitted, checks are enumerated)

NO_FAIL [optional] do not fail if the flag cannot be added

1.6 ecbuild_add_library

Add a library with a given list of source files.

```
ecbuild_add_library( TARGET <name>
                     SOURCES <source1> [<source2> ...]
                     [ SOURCES_GLOB <glob1> [<glob2> ...] ]
                     [ SOURCES_EXCLUDE_REGEX <regex1> [<regex2> ...] ]
                     [ TYPE SHARED|STATIC|MODULE|OBJECT|INTERFACE ]
                     [ OBJECTS <obj1> [<obj2> ...] ]
                     [ TEMPLATES <template1> [<template2> ...] ]
                     [ LIBS <library1> [<library2> ...] ]
                     [ PRIVATE_LIBS <library1> [<library2> ...] ]
                     [ PUBLIC_LIBS <library1> [<library2> ...] ]
                     [ INCLUDES <path1> [<path2> ...] ]
                     [ PRIVATE_INCLUDES <path1> [<path2> ...] ]
                     [ PUBLIC_INCLUDES <path1> [<path2> ...] ]
                     [ DEFINITIONS <definition1> [<definition2> ...] ]
                     [ PRIVATE_DEFINITIONS <definition1> [<definition2> ...] ]
                     [ PUBLIC_DEFINITIONS <definition1> [<definition2> ...] ]
                     [ PERSISTENT <file1> [<file2> ...] ]
                     [ GENERATED <file1> [<file2> ...] ]
                     [ DEPENDS <target1> [<target2> ...] ]
                     [ CONDITION <condition> ]
                     [ PROPERTIES <prop1> <val1> [<prop2> <val2> ...] ]
                     [ NOINSTALL ]
                     [ HEADER_DESTINATION <path> ]
                     [ INSTALL_HEADERS LISTED|ALL ]
                     [ INSTALL_HEADERS_LIST <header1> [<header2> ...] ]
                     [ INSTALL_HEADERS_REGEX <pattern> ]
                     [ VERSION <version> | AUTO_VERSION ]
                     [ SOVERSION <soversion> | AUTO_SOVERSION ]
```

(continues on next page)

(continued from previous page)

```
                    [ CFLAGS <flag1> [<flag2> ...] ]
                    [ CXXFLAGS <flag1> [<flag2> ...] ]
                    [ FFLAGS <flag1> [<flag2> ...] ]
                    [ LINKER_LANGUAGE <lang> ]
                    [ OUTPUT_NAME <name> ] )
```

1.6.1 Options

TARGET [required] target name

SOURCES [required] list of source files

TYPE [optional] library type, one of:

SHARED libraries are linked dynamically and loaded at runtime

STATIC archives of object files for use when linking other targets.

MODULE plugins that are not linked into other targets but may be loaded dynamically at runtime using dlopen-like functionality

OBJECT files are just compiled into objects

INTERFACE no direct build output, but can be used to aggregate headers, compilation flags and libraries

SOURCES_GLOB [optional] search pattern to find source files to compile (note: not recommend according to CMake guidelines) it is usually better to explicitly list the source files in the CMakeList.txt

SOURCES_EXCLUDE_REGEX [optional] search pattern to exclude source files from compilation, applies o the results of SOURCES_GLOB

OBJECTS [optional] list of object libraries to add to this target

TEMPLATES [optional] list of files specified as SOURCES which are not to be compiled separately (these are commonly template implementation files included in a header)

LIBS [(DEPRECATED) optional] list of libraries to link against (CMake targets or external libraries), behaves as PUBLIC_LIBS Please use PRIVATE_LIBS or PUBLIC_LIBS or CMake command `target_link_libraries` instead

PRIVATE_LIBS [optional] list of libraries to link against (CMake targets or external libraries), they will not be exported

PUBLIC_LIBS [optional] list of libraries to link against (CMake targets or external libraries), they will be exported

INCLUDES [(DEPRECATED) optional] list of paths to add to include directories, behaves as PUBLIC_INCLUDES Please use PUBLIC_INCLUDES or PRIVATE_INCLUDES or CMake command `target_include_directories` instead

PUBLIC_INCLUDES [optional] list of paths to add to include directories which will be publicly exported to other targets and projects

PRIVATE_INCLUDES [optional] list of paths to add to include directories which won't be exported beyond this target

DEFINITIONS [(DEPRECATED) optional] list of definitions to add to preprocessor defines behaves as PRIVATE_DEFINITIONS Please use PRIVATE_DEFINITIONS or PUBLIC_DEFINITIONS or CMake command `target_compile_definitions` instead

PRIVATE_DEFINITIONS [optional] list of definitions to add to preprocessor defines that will not be exported beyond this target

PUBLIC_DEFINITIONS [optional] list of definitions to add to preprocessor defines that will be publicly exported to other targets and projects

PERSISTENT [optional] list of persistent layer object files

GENERATED [optional] list of files to mark as generated (sets GENERATED source file property)

DEPENDS [optional] list of targets to be built before this target

CONDITION [optional] conditional expression which must evaluate to true for this target to be built (must be valid in a CMake `if` statement)

PROPERTIES [optional] custom properties to set on the target

NOINSTALL [optional] do not install the library

HEADER_DESTINATION directory to install headers (if not specified, INSTALL_INCLUDE_DIR is used) Note: this directory will automatically be added to target_include_directories

INSTALL_HEADERS [optional] specify which header files to install:

LISTED install header files listed as SOURCES

ALL install all header files ending in .h, .hh, .hpp, .H

INSTALL_HEADERS_LIST [optional] list of extra headers to install

INSTALL_HEADERS_REGEX [optional] regular expression to match extra headers to install

VERSION [optional, AUTO_VERSION or LIBS_VERSION is used if not specified] build version of the library

AUTO_VERSION [optional, ignored if VERSION is specified] use MAJOR.MINOR package version as build version of the library

SOVERSION [optional, AUTO_SOVERSION or LIBS_SOVERSION is used if not specified] ABI version of the library

AUTO_SOVERSION [optional, ignored if SOVERSION is specified] use MAJOR package version as ABI version of the library

CFLAGS [optional] list of C compiler flags to use for all C source files

See usage note below.

CXXFLAGS [optional] list of C++ compiler flags to use for all C++ source files

See usage note below.

FFLAGS [optional] list of Fortran compiler flags to use for all Fortran source files

See usage note below.

LINKER_LANGUAGE [optional] sets the LINKER_LANGUAGE property on the target

OUTPUT_NAME [optional] sets the OUTPUT_NAME property on the target

1.6.2 Usage

The `CFLAGS`, `CXXFLAGS` and `FFLAGS` options apply the given compiler flags to all C, C++ and Fortran sources passed to this command, respectively. If any two `ecbuild_add_executable`, `ecbuild_add_library` or `ecbuild_add_test` commands are passed the *same* source file and each sets a different value for the compiler flags to be applied to that file (including when one command adds flags and another adds none), then the two commands will be in conflict and the result may not be as expected.

For this reason it is recommended not to use the `*FLAGS` options when multiple targets share the same source files, unless the exact same flags are applied to those sources by each relevant command.

Care should also be taken to ensure that these commands are not passed source files which are not required to build the target, if those sources are also passed to other commands which set different compiler flags.

1.7 ecbuild_add_option

Add a CMake configuration option, which may depend on a list of packages.

```
ecbuild_add_option( FEATURE <name>
                    [ DEFAULT ON|OFF ]
                    [ DESCRIPTION <description> ]
                    [ REQUIRED_PACKAGES <package1> [<package2> ...] ]
                    [ CONDITION <condition> ]
                    [ ADVANCED ] [ NO_TPL ] )
```

1.7.1 Options

FEATURE [required] name of the feature / option

DEFAULT [optional, defaults to ON] if set to ON, the feature is enabled even if not explicitly requested

DESCRIPTION [optional] string describing the feature (shown in summary and stored in the cache)

REQUIRED_PACKAGES [optional] list of packages required to be found for this feature to be enabled

Every item in the list should be a valid argument list for `ecbuild_find_package`, e.g.:

```
"NAME <package> [VERSION <version>] [...]"
```

Note: Arguments inside the package string that require quoting need to use the bracket argument syntax introduced in CMake 3.0 since regular quotes even when escaped are swallowed by the CMake parser.

Alternatively, the name of a CMake variable containing the string can be passed, which will be expanded by `ecbuild_find_package`:

```
set( ECCODES_FAIL_MSG
     "grib_api can be used instead (select with -DENABLE_ECCODES=OFF)" )
ecbuild_add_option( FEATURE ECCODES
                    DESCRIPTION "Use eccodes instead of grib_api"
                    REQUIRED_PACKAGES "NAME eccodes REQUIRED FAILURE_MSG ECCODES_
↪FAIL_MSG"
                    DEFAULT ON )
```

CONDITION [optional] conditional expression which must evaluate to true for this option to be enabled (must be valid in a CMake `if` statement)

ADVANCED [optional] mark the feature as advanced

NO_TPL [optional] do not add any `REQUIRED_PACKAGES` to the list of third party libraries

1.7.2 Usage

Features with `DEFAULT OFF` need to be explcitly enabled by the user with `-DENABLE_<FEATURE>=ON`. If a feature is enabled, all `REQUIRED_PACKAGES` are found and `CONDITION` is met, ecBuild sets the variable `HAVE_<FEATURE>` to `ON`. This is the variable to use to check for the availability of the feature.

If a feature is explicitly enabled but the required packages are not found, configuration fails. This only applies when configuring from *clean cache*. With an already populated cache, use `-DENABLE_<FEATURE>=REQUIRE` to make the feature a required feature (this cannot be done via the CMake GUI).

1.8 ecbuild_add_persistent

Add persistent layer object classes.

```
ecbuild_add_persistent( SRC_LIST <variable>
                        FILES <file1> [<file2> ...] ]
                        [ NAMESPACE <namespace> ] )
```

1.8.1 Options

SRC_LIST [required] CMake variable to append the generated persistent layer objects to

FILES [required] list of base names of files to build persistent class information for

The source file is expected to have a .h extension, the generated file gets a .b extension.

NAMESPACE [optional] C++ namespace to place the persistent class information in

1.9 ecbuild_add_resources

Add resources as project files but optionally exclude them from packaging.

```
ecbuild_add_resources( TARGET <name>
                       [ SOURCES <source1> [<source2> ...] ]
                       [ SOURCES_PACK <source1> [<source2> ...] ]
                       [ SOURCES_DONT_PACK <source1> [<source2> ...] ]
                       [ PACK <file1> [<file2> ...] ]
                       [ DONT_PACK <file1> [<file2> ...] ]
                       [ DONT_PACK_DIRS <directory1> [<directory2> ...] ]
                       [ DONT_PACK_REGEX <regex1> [<regex2> ...] ] )
```

1.9.1 Options

TARGET [required] target name (target will only be created if there are any sources)

SOURCES [optional, alias for SOURCES_PACK] list of source files included when packaging

SOURCES_PACK [optional, alias for SOURCES] list of source files included when packaging

SOURCES_DONT_PACK [optional] list of source files excluded when packaging

PACK [optional, priority over DONT_PACK, DONT_PACK_DIRS, DONT_PACK_REGEX] list of files to include when packaging

DONT_PACK [optional] list of files to exclude when packaging

DONT_PACK_DIRS [optional] list of directories to exclude when packaging

DONT_PACK_REGEX [optional] list of regular expressions to match files and directories to exclude when packaging

1.9.2 Note

All file and directory names are also *partially matched*. To ensure that only the exact file or directory name is matched at the end of the path add a `$` at the end and quote the name.

1.10 ecbuild_add_test

Add a test as a script or an executable with a given list of source files.

```
ecbuild_add_test( [ TARGET <name> ]
                  [ SOURCES <source1> [<source2> ...] ]
                  [ OBJECTS <obj1> [<obj2> ...] ]
                  [ COMMAND <executable> ]
                  [ TYPE EXE|SCRIPT|PYTHON ]
                  [ LABELS <label1> [<label2> ...] ]
                  [ ARGS <argument1> [<argument2> ...] ]
                  [ RESOURCES <file1> [<file2> ...] ]
                  [ TEST_DATA <file1> [<file2> ...] ]
                  [ MPI <number-of-mpi-tasks> ]
                  [ OMP <number-of-threads-per-mpi-task> ]
                  [ ENABLED ON|OFF ]
                  [ LIBS <library1> [<library2> ...] ]
                  [ INCLUDES <path1> [<path2> ...] ]
                  [ DEFINITIONS <definition1> [<definition2> ...] ]
                  [ PERSISTENT <file1> [<file2> ...] ]
                  [ GENERATED <file1> [<file2> ...] ]
                  [ DEPENDS <target1> [<target2> ...] ]
                  [ TEST_DEPENDS <target1> [<target2> ...] ]
                  [ CONDITION <condition> ]
                  [ PROPERTIES <prop1> <val1> [<prop2> <val2> ...] ]
                  [ ENVIRONMENT <variable1> [<variable2> ...] ]
                  [ WORKING_DIRECTORY <path> ]
                  [ CFLAGS <flag1> [<flag2> ...] ]
                  [ CXXFLAGS <flag1> [<flag2> ...] ]
                  [ FFLAGS <flag1> [<flag2> ...] ]
                  [ LINKER_LANGUAGE <lang> ] )
```

1.10.1 Options

TARGET [either TARGET or COMMAND must be provided, unless TYPE is PYTHON] target name to be built

SOURCES [required if TARGET is provided] list of source files to be compiled

OBJECTS [optional] list of object libraries to add to this target

COMMAND [either TARGET or COMMAND must be provided, unless TYPE is PYTHON] command or script to execute (no executable is built)

TYPE [optional] test type, one of:

EXE run built executable, default if TARGET is provided

SCRIPT run command or script, default if COMMAND is provided

PYTHON run a Python script (requires the Python interpreter to be found)

LABELS [optional] list of labels to assign to the test

The project name in lower case is always added as a label. Additional labels are assigned depending on the type of test:

executable for type `EXE`

script for type `SCRIPT`

python for type `PYTHON`

mpi if `MPI` is set

openmp if `OMP` is set

This allows selecting tests to run via `ctest -L <regex>` or tests to exclude via `ctest -LE <regex>`.

ARGS [optional] list of arguments to pass to TARGET or COMMAND when running the test

RESOURCES [optional] list of files to copy from the test source directory to the test directory

TEST_DATA [optional] list of test data files to download

MPI [optional] Run with MPI using the given number of MPI tasks.

If greater than 1, and `MPIEXEC` is not available, the test is disabled.

OMP [optional] number of OpenMP threads per MPI task to use.

If set, the environment variable OMP_NUM_THREADS will set. Also, in case of launchers like aprun, the OMP_NUMTHREADS_FLAG will be used.

ENABLED [optional] if set to OFF, the test is built but not enabled as a test case

LIBS [optional] list of libraries to link against (CMake targets or external libraries)

INCLUDES [optional] list of paths to add to include directories

DEFINITIONS [optional] list of definitions to add to preprocessor defines

PERSISTENT [optional] list of persistent layer object files

GENERATED [optional] list of files to mark as generated (sets GENERATED source file property)

DEPENDS [optional] list of targets to be built before this target

TEST_DEPENDS [optional] list of tests to be run before this one

CONDITION [optional] conditional expression which must evaluate to true for this target to be built (must be valid in a CMake `if` statement)

PROPERTIES [optional] custom properties to set on the target

ENVIRONMENT [optional] list of environment variables to set in the test environment

WORKING_DIRECTORY [optional] directory to switch to before running the test

CFLAGS [optional] list of C compiler flags to use for all C source files

See usage note below.

CXXFLAGS [optional] list of C++ compiler flags to use for all C++ source files

See usage note below.

FFLAGS [optional] list of Fortran compiler flags to use for all Fortran source files

See usage note below.

LINKER_LANGUAGE [optional] sets the LINKER_LANGUAGE property on the target

1.10.2 Usage

The `CFLAGS`, `CXXFLAGS` and `FFLAGS` options apply the given compiler flags to all C, C++ and Fortran sources passed to this command, respectively. If any two `ecbuild_add_executable`, `ecbuild_add_library` or `ecbuild_add_test` commands are passed the *same* source file and each sets a different value for the compiler flags to be applied to that file (including when one command adds flags and another adds none), then the two commands will be in conflict and the result may not be as expected.

For this reason it is recommended not to use the `*FLAGS` options when multiple targets share the same source files, unless the exact same flags are applied to those sources by each relevant command.

Care should also be taken to ensure that these commands are not passed source files which are not required to build the target, if those sources are also passed to other commands which set different compiler flags.

1.11 ecbuild_append_to_rpath

Append paths to the rpath.

```
ecbuild_append_to_rpath( RPATH_DIRS )
```

`RPATH_DIRS` is a list of directories to append to `CMAKE_INSTALL_RPATH`.

- If a directory is absolute, simply append it.
- If a directory is relative, build a platform-dependent relative path (using `@loader_path` on Mac OSX, `$ORIGIN` on Linux and Solaris) or fall back to making it absolute by prepending the install prefix.

1.12 ecbuild_bundle_initialize

Initialise the ecBuild environment for a bundle. *Must* be called *before* any call to `ecbuild_bundle`.

```
ecbuild_bundle_initialize()
```

1.13 ecbuild_bundle

Declare a subproject to be built as part of this bundle.

```
ecbuild_bundle( PROJECT <name>
                STASH <repository> | GIT <gituri> | SOURCE <path>
                [ BRANCH <gitbranch> | TAG <gittag> ]
                [ UPDATE | NOREMOTE ]
                [ MANUAL ]
                [ RECURSIVE ] )
```

1.13.1 Options

PROJECT [required] project name for the Git repository to be managed

STASH [DEPRECATED ; cannot be combined with GIT or SOURCE] Stash repository in the form <project>/<repository>

GIT [cannot be combined with STASH or SOURCE] Git URL of the remote repository to clone (see `git help clone`)

SOURCE [cannot be combined with STASH or GIT] Path to an existing local repository, which will be symlinked

BRANCH [optional, cannot be combined with TAG] Git branch to check out

TAG [optional, cannot be combined with BRANCH] Git tag or commit id to check out

UPDATE [optional, requires BRANCH, cannot be combined with NOREMOTE] Create a CMake target update to fetch changes from the remote repository

NOREMOTE [optional, cannot be combined with UPDATE] Do not fetch changes from the remote repository

MANUAL [optional] Do not automatically switch branches or tags

RECURSIVE [optional] Do a recursive fetch or update

1.13.2 Usage

A bundle is used to build a number of projects together. Each subproject needs to be declared with a call to ecbuild_bundle, where the order of projects is important and needs to respect dependencies: if project B depends on project A, A should be listed before B in the bundle.

The first time a bundle is built, the sources of all subprojects are cloned into directories named according to project in the *source* tree of the bundle (which means these directories should be added to `.gitignore`). If the `SOURCE` option is used it must point to an existing local repository on disk and no new repository is cloned. Be aware that using the `BRANCH` or `TAG` option leads to the corresponding version being checked out in that repository!

Subprojects are configured and built in order. Due to being added as a subproject, the usual project discovery mechanism (i.e. locating and importing a `<project>-config.cmake` file) is not used. Also there are no `<project>-config.cmake` files being generated for individual subprojects. However there *are* package-config files being generated for each library.

To switch off a subproject when building a bundle, set the CMake variable `BUNDLE_SKIP_<PNAME>` where `PNAME` is the capitalised project name.

1.14 ecbuild_bundle_finalize

Finalise the ecBuild environment for a bundle. *Must* be called *after* the last call to `ecbuild_bundle`.

```
ecbuild_bundle_finalize()
```

1.14.1 Options

See documentation for ecbuild_install_project() since all arguments are forwarded to an internal call to that macro.

If no arguments are passed, then the default installation NAME is set to the default project name ${CMAKE_PROJECT_NAME}

1.15 ecBuild Cache

During initialisation, ecBuild introspects the compiler and operating system and performs a number of checks. The result of these is written to a dedicated `ecbuild-cache.cmake` file in the build tree. This cache may be used to speed up subsequent *clean* builds i.e. those where no CMakeCache.txt exists yet.

To use the ecBuild cache, configure with `-DECBUILD_CACHE=<cache-file>`, where `<cache-file>` is the path to an existing `ecbuild-cache.cmake`.

Note: The ecBuild cache is specific to compiler *and* operating system. Do *not* attempt to use a cache file created on a different machine or with a different compiler!

1.16 ecbuild_check_c_source_return

Compile and run a given C source code and return its output.

```
ecbuild_check_c_source_return( <source>
                               VAR <name>
                               OUTPUT <name>
                               [ INCLUDES <path1> [ <path2> ... ] ]
                               [ LIBS <library1> [ <library2> ... ] ]
                               [ DEFINITIONS <definition1> [ <definition2> ... ] ] )
```

1.16.1 Options

VAR [required] name of the check and name of the CMake variable to write result to

OUTPUT [required] name of CMake variable to write the output to

INCLUDES [optional] list of paths to add to include directories

LIBS [optional] list of libraries to link against (CMake targets or external libraries)

DEFINITIONS [optional] list of definitions to add to preprocessor defines

1.16.2 Usage

This will write the given source to a .c file and compile and run it with ecbuild_try_run. If successful, `${VAR}` is set to 1 and `${OUTPUT}` is set to the output of the successful run in the CMake cache.

The check will not run if `${VAR}` is defined (e.g. from ecBuild cache).

1.17 ecbuild_check_cxx_source_return

Compile and run a given C++ code and return its output.

```
ecbuild_check_cxx_source_return( <source>
                                 VAR <name>
                                 OUTPUT <name>
                                 [ INCLUDES <path1> [ <path2> ... ] ]
                                 [ LIBS <library1> [ <library2> ... ] ]
                                 [ DEFINITIONS <definition1> [ <definition2> ... ] ] )
```

1.17.1 Options

VAR [required] name of the check and name of the CMake variable to write result to

OUTPUT [required] name of CMake variable to write the output to

INCLUDES [optional] list of paths to add to include directories

LIBS [optional] list of libraries to link against (CMake targets or external libraries)

DEFINITIONS [optional] list of definitions to add to preprocessor defines

1.17.2 Usage

This will write the given source to a .cxx file and compile and run it with ecbuild_try_run. If successful, `${VAR}` is set to 1 and `${OUTPUT}` is set to the output of the successful run in the CMake cache.

The check will not run if `${VAR}` is defined (e.g. from ecBuild cache).

1.18 ecbuild_check_fortran

Check for Fortran features.

```
ecbuild_check_fortran( [ FEATURES <feature1> [ <feature2> ... ] ]
                       [ REQUIRED <feature1> [ <feature2> ... ] ]
                       [ PRINT ] )
```

1.18.1 Options

FEATURES [optional] list of optional features to check for

REQUIRED [optional] list of required features to check for, fails if not detected

PRINT [optional] print a summary of features checked for, found and not found

1.18.2 Note

If neither `FEATURES` nor `REQUIRED` are given, check for all features.

1.19 ecbuild_check_fortran_source_return

Compile and run a given Fortran code and return its output.

```
ecbuild_check_fortran_source_return( <source>
                                     VAR <name>
                                     OUTPUT <name>
                                     [ INCLUDES <path1> [ <path2> ... ] ]
                                     [ LIBS <library1> [ <library2> ... ] ]
                                     [ DEFINITIONS <def1> [ <def2> ... ] ] )
```

1.19.1 Options

VAR [required] name of the check and name of the CMake variable to write result to

OUTPUT [required] name of CMake variable to write the output to

INCLUDES [optional] list of paths to add to include directories

LIBS [optional] list of libraries to link against (CMake targets or external libraries)

DEFINITIONS [optional] list of definitions to add to preprocessor defines

1.19.2 Usage

This will write the given source to a .f file and compile and run it with ecbuild_try_run. If successful, `${VAR}` is set to 1 and `${OUTPUT}` is set to the output of the successful run in the CMake cache.

The check will not run if `${VAR}` is defined (e.g. from ecBuild cache).

1.20 ecbuild_check_urls

Check multiple URL validity.

```
ecbuild_check_urls( NAMES <name1> [ <name2> ... ]
                          RESULT <result> )
```

curl or wget is required (curl is preferred if available).

1.20.1 Options

NAMES [required] list of names of the files to check, including the directory structure on the server hosting test files (if available)

RESULT [required] check result (0 if all URLs exist, more if not)

1.20.2 Usage

Check whether files exist on `<ECBUILD_DOWNLOAD_BASE_URL>/<NAME>` for each name given in the list of `NAMES`. `RESULT` is set to the number of missing files.

1.20.3 Examples

Check file ... existence:

```
ecbuild_check_urls( NAMES test/data/dir/msl1.grib test/data/dir/msl2.grib
                        RESULT FILES_EXIST )
```

1.21 ecbuild_compiler_flags

Set compiler specific default compilation flags for a given language.

```
ecbuild_compiler_flags( <lang> )
```

The procedure is as follows:

1. ecBuild does **not** set `CMAKE_<lang>_FLAGS` i.e. the user can set these via `-D` or the CMake cache and these will be the "base" flags.
2. ecBuild **overwrites** `CMAKE_<lang>_FLAGS_<btype>` in the CMake cache for all build types with compiler specific defaults for the currently loaded compiler i.e. any value set by the user via `-D` or the CMake cache **has no effect**.
3. Any value the user provides via `ECBUILD_<lang>_FLAGS` or `ECBUILD_<lang>_FLAGS_<btype>` **overrides** the corresponding `CMAKE_<lang>_FLAGS` or `CMAKE_<lang>_FLAGS_<btype>` **without being written to the CMake cache**.

1.22 Using custom compilation flags

If compilation flags need to be controlled on a per source file basis, ecBuild supports defining custom rules in a CMake or JSON file.

When using this approach, *default compilation flags are NOT loaded*!

1.22.1 Overriding compilation flags on a per source file basis using CMake rules

Compiler flags can be overridden on a per source file basis by setting the CMake variable `ECBUILD_COMPILE_FLAGS` to the *full path* of a CMake file defining the override rules. If set, `<PNAME>_ECBUILD_COMPILE_FLAGS` takes precendence and `ECBUILD_COMPILE_FLAGS` is ignored, allowing for rules that only apply to a subproject (e.g. in a bundle).

Flags can be overridden in 3 different ways:

1. By defining project specific flags for a language and (optionally) build type e.g.

```
set(<PNAME>_Fortran_FLAGS "...") # common flags for all build types
set(<PNAME>_Fortran_FLAGS_DEBUG "...") # only for DEBUG build type
```

2. By defining source file specific flags which are *combined* with the project and target specific flags

```
set_source_files_properties(<source>
  PROPERTIES COMPILE_FLAGS "..."  # common flags for all build types
             COMPILE_FLAGS_DEBUG "...") # only for DEBUG build type
```

3. By defining source file specific flags which *override* the project and target specific flags

```
set_source_files_properties(<source>
  PROPERTIES OVERRIDE_COMPILE_FLAGS "..."
             OVERRIDE_COMPILE_FLAGS_DEBUG "...")
```

See `examples/override-compile-flags` in the ecBuild source tree for a complete example using this technique.

1.22.2 Overriding compilation flags on a per source file basis using JSON rules

Compiler flags can be overridden on a per source file basis by setting the CMake variable `ECBUILD_SOURCE_FLAGS` to the *full path* of a JSON file defining the override rules. If set, `<PNAME>_ECBUILD_SOURCE_FLAGS` takes precendence and `ECBUILD_SOURCE_FLAGS` is ignored, allowing for rules that only apply to a subproject (e.g. in a bundle).

The JSON file lists shell glob patterns and the rule to apply to each source file matching the pattern, defined as an array `[op, flag1, ...]` containing an operator followed by one or more flags. Valid operators are:

+ Add the flags to the default compilation flags for matching files

= Set the flags for matching files, disregarding default compilation flags

/ Remove the flags from the default compilation flags for matching files

Rules can be nested to e.g. only apply to a subdirectory by setting the rule to a dictionary, which will only apply to source files matching its pattern.

An example JSON file demonstrating different rule types is given below:

```
{
  "*"       : [ "+", "-g0" ],
  "*.cxx"   : [ "+", "-cxx11" ],
  "*.f90"   : [ "+", "-pipe" ],
  "foo.c"   : [ "+", "-O0" ],
  "foo.cc"  : [ "+", "-O2", "-pipe" ],
  "bar/*": {
    "*.f90" : [ "=", "-O1" ]
```

(continues on next page)

(continued from previous page)

```
  },
  "baz/*": {
    "*.f90" : [ "/", "-pipe" ],
    "*.f90" : [ "/", "-O2" ],
    "*.f90" : [ "+", "-O3" ]
  }
}
```

See `examples/override-compile-flags` in the ecBuild source tree for a complete example using this technique.

1.23 ecbuild_declare_project

Initialise an ecBuild project. A CMake project must have previously been declared with `project( <name> ... )`.

```
ecbuild_declare_project()
```

Sets the following CMake variables

<PROJECT_NAME>_GIT_SHA1 Git revision (if project is a Git repo)

<PROJECT_NAME>_GIT_SHA1_SHORT short Git revision (if project is a Git repo)

<PROJECT_NAME>_VERSION version as given in project(VERSION)

<PROJECT_NAME>_VERSION_MAJOR major version number

<PROJECT_NAME>_VERSION_MINOR minor version number

<PROJECT_NAME>_VERSION_PATCH patch version number

INSTALL_BIN_DIR relative install directory for executables

INSTALL_LIB_DIR relative install directory for libraries

INSTALL_INCLUDE_DIR relative install directory for include files

INSTALL_DATA_DIR relative install directory for data

INSTALL_CMAKE_DIR relative install directory for CMake files

Generation of the first two variables can be disabled by setting the ECBUILD_RECORD_GIT_COMMIT_SHA1 option to OFF. This prevents makefiles from being regenerated whenever the Git revision changes.

1.23.1 Customising install locations

The relative installation directories of components can be customised by setting the following CMake variables on the command line or in cache:

INSTALL_BIN_DIR directory for installing executables (default: `bin`)

INSTALL_LIB_DIR directory for installing libraries (default: `lib`)

INSTALL_INCLUDE_DIR directory for installing include files (default: `include`)

INSTALL_DATA_DIR directory for installing data (default: `share/<project_name>`)

INSTALL_CMAKE_DIR directory for installing CMake files (default: `lib/cmake/<project_name>`)

Using *relative* paths is recommended, which are interpreted relative to the CMAKE_INSTALL_PREFIX. Using absolute paths makes the build non-relocatable and may break the generation of relocatable binary packages.

1.24 ecbuild_dont_pack

Specify files and directories to exclude from packaging.

```
ecbuild_dont_pack( [ FILES <file1> [ <file2> ... ] ]
                   [ DIRS <dir1> [ <dir2> ... ] ]
                   [ REGEX <regex> ] )
```

1.24.1 Options

FILES [optional, one of FILES, DIRS, REGEX required] list of files to exclude from packaging

DIRS [optional, one of FILES, DIRS, REGEX required] list of directories to exclude from packaging

REGEX [optional, one of FILES, DIRS, REGEX required] regular expression to match files / directories to exclude from packaging

1.25 ecbuild_download_resource

Download a file from a given URL and save to FILE at configure time.

```
ecbuild_download_resource( FILE URL )
```

curl or wget is required (curl is preferred if available).

The default timeout is 30 seconds, which can be overridden with ECBUILD_DOWNLOAD_TIMEOUT. Downloads are by default only tried once, use ECBUILD_DOWNLOAD_RETRIES to set the number of retries.

1.26 ecbuild_echo_target_property

Output a given property of a given target.

```
ecbuild_echo_target_property( <target> <property> )
```

1.27 ecbuild_echo_target

Output all possible target properties of a given target.

```
ecbuild_echo_target( <target> )
```

1.28 ecbuild_echo_targets

Output all possible target properties of the specified list-of-targets. This is very useful for debugging.

```
ecbuild_echo_targets( <list-of-targets> )
```

1.29 ecbuild_enable_fortran

Enable the Fortran language.

```
ecbuild_enable_fortran( [ MODULE_DIRECTORY <directory> ] [ REQUIRED ] )
```

1.29.1 Options

MODULE_DIRECTORY [optional, defaults to `${PROJECT_BINARY_DIR}/module`] set the CMAKE_Fortran_MODULE_DIRECTORY

NO_MODULE_DIRECTORY [optional] unset CMAKE_Fortran_MODULE_DIRECTORY

REQUIRED [optional] fail if no working Fortran compiler was detected

1.30 ecbuild_evaluate_dynamic_condition

Add a CMake configuration option, which may depend on a list of packages.

```
ecbuild_evaluate_dynamic_condition( condition outVariable )
```

1.30.1 Options

condition A list of boolean statements like OPENSSL_FOUND AND ENABLE_OPENSSL

1.31 ecbuild_filter_list

Filters a list for NOTFOUND entries and non existing TARGETS.

```
ecbuild_filter_list( [INCLUDES] [LIBS]
                     LIST <list>
                     [LIST_INCLUDE <output_list>]
                     [LIST_EXCLUDE <output_list>])
```

1.31.1 Options

INCLUDES [optional] Consider existing dirs as valid

LIBS [optional] Consider existing targets, files and compile flags as valid

LIST [required] a list

LIST_INCLUDE [optional] The output list with all valid entries of LIST

LIST_EXCLUDE [optional] The output list with all invalid entries of LIST

1.32 ecbuild_find_fortranlibs

Find the Fortran (static) link libraries.

```
ecbuild_find_fortranlibs( [ COMPILER gfortran|pgi|xlf|intel ]
                          [ REQUIRED ] )
```

1.32.1 Options

COMPILER [optional, defaults to gfortran] request a given Fortran compiler (`gfortran`, `pgi`, `xlf`, `intel`)

REQUIRED [optional] fail if Fortran libraries were not found

1.33 ecbuild_find_lexyacc

Find flex and bison (preferred) or lex and yacc.

1.33.1 Input variables

The following CMake variables can set to skip search for bison or yacc:

SKIP_BISON do not search for flex and bison

SKIP_YACC do not search for lex and yacc

1.33.2 Output variables

The following CMake variables are set if flex and bison were found:

FLEX_FOUND flex was found

BISON_FOUND bison was found

FLEX_EXECUTABLE path to the flex executable

BISON_EXECUTABLE path to the bison executable

The following CMake variables are set if lex and yacc were found:

LEXYACC_FOUND Found suitable combination of bison, lex, yacc, flex

LEX_FOUND lex was found

YACC_FOUND yacc was found

LEX_EXECUTABLE path to the lex executable

YACC_EXECUTABLE path to the yacc executable

1.34 ecbuild_find_mpi

Find MPI and check if MPI compilers successfully compile C/C++/Fortran.

```
ecbuild_find_mpi( [ COMPONENTS <component1> [ <component2> ... ] ]
                  [ REQUIRED ] )
```

1.34.1 Options

COMPONENTS [optional, defaults to C] list of required languages bindings

REQUIRED [optional] fail if MPI was not found

1.34.2 Input variables

ECBUILD_FIND_MPI [optional, defaults to TRUE] test C/C++/Fortran MPI compiler wrappers (assume working if FALSE)

1.34.3 Output variables

The following CMake variables are set if MPI was found:

```
MPI_FOUND
MPI_LIBRARY
MPI_EXTRA_LIBRARY
```

The following CMake variables are set if C bindings were found:

```
MPI_C_FOUND
MPI_C_COMPILER
MPI_C_COMPILE_FLAGS
MPI_C_INCLUDE_PATH
MPI_C_LIBRARIES
MPI_C_LINK_FLAGS
```

The following CMake variables are set if C++ bindings were found:

```
MPI_CXX_FOUND
MPI_CXX_COMPILER
MPI_CXX_COMPILE_FLAGS
MPI_CXX_INCLUDE_PATH
MPI_CXX_LIBRARIES
MPI_CXX_LINK_FLAGS
```

The following CMake variables are set if Fortran bindings were found:

```
MPI_Fortran_FOUND
MPI_Fortran_COMPILER
MPI_Fortran_COMPILE_FLAGS
MPI_Fortran_INCLUDE_PATH
MPI_Fortran_LIBRARIES
MPI_Fortran_LINK_FLAGS
```

1.35 ecbuild_enable_mpi

Find MPI, add include directories and set compiler flags.

```
ecbuild_enable_mpi( [ COMPONENTS <component1> [ <component2> ... ] ]
                    [ REQUIRED ] )
```

For each MPI language binding found, set the corresponding compiler flags and add the include directories.

See `ecbuild_find_mpi` for input and output variables.

1.35.1 Options

COMPONENTS [optional, defaults to C] list of required languages bindings

REQUIRED [optional] fail if MPI was not found

1.36 ecbuild_include_mpi

Add MPI include directories and set compiler flags, assuming MPI was found.

For each MPI language binding found, set corresponding compiler flags and add include directories. `ecbuild_find_mpi` must have been called before.

1.37 ecbuild_find_omp

Find OpenMP.

```
ecbuild_find_omp( [ COMPONENTS <component1> [ <component2> ... ] ]
                  [ REQUIRED ]
                  [ STUBS ] )
```

1.37.1 Options

COMPONENTS [optional, defaults to C] list of required languages bindings

REQUIRED [optional] fail if OpenMP was not found

STUBS [optional] search for OpenMP stubs

1.37.2 Output variables

The following CMake variables are set if OpenMP was found:

OMP_FOUND OpenMP was found

For each language listed in COMPONENTS, the following variables are set:

OMP_<LANG>_FOUND OpenMP bindings for LANG were found

OMP_<LANG>_FLAGS OpenMP compiler flags for LANG

If the STUBS option was given, all variables are also set with the OMPSTUBS instead of the OMP prefix.

1.38 ecbuild_enable_omp

Find OpenMP for C, C++ and Fortran and set the compiler flags for each language for which OpenMP support was detected.

1.39 ecbuild_enable_ompstubs

Find OpenMP stubs for C, C++ and Fortran and set the compiler flags for each language for which OpenMP stubs were detected.

1.40 ecbuild_find_package

Find a package and import its configuration.

```
ecbuild_find_package( [ NAME ] <name>
                      [ [ VERSION ] <version> [ EXACT ] ]
                      [ COMPONENTS <component1> [ <component2> ... ] ]
                      [ URL <url> ]
                      [ DESCRIPTION <description> ]
                      [ TYPE <type> ]
                      [ PURPOSE <purpose> ]
                      [ FAILURE_MSG <message> ]
                      [ REQUIRED ]
                      [ QUIET ] )
```

1.40.1 Options

NAME [required] package name (used as `Find<name>.cmake` and `<name>-config.cmake`)

VERSION [optional] minimum required package version

COMPONENTS [optional] list of package components to find (behaviour depends on the package)

EXACT [optional, requires VERSION] require the exact version rather than a minimum version

URL [optional] homepage of the package (shown in summary and stored in the cache)

DESCRIPTION [optional] literal string or name of CMake variable describing the package

TYPE [optional, one of RUNTIME|OPTIONAL|RECOMMENDED|REQUIRED] type of dependency of the project on this package (defaults to OPTIONAL)

PURPOSE [optional] literal string or name of CMake variable describing which functionality this package enables in the project

FAILURE_MSG [optional] literal string or name of CMake variable containing a message to be appended to the failure message if the package is not found

REQUIRED [optional (equivalent to TYPE REQUIRED, and overrides TYPE argument)] fail if package cannot be found

QUIET [optional] do not output package information if found

1.40.2 Input variables

The following CMake variables influence the behaviour if set (`<name>` is the package name as given, `<NAME>` is the capitalised version):

<name>_ROOT install prefix path of the package

<name>_PATH install prefix path of the package, prefer <name>_ROOT

<NAME>_PATH install prefix path of the package, prefer <name>_ROOT

<name>_DIR directory containing the `<name>-config.cmake` file (usually `<install-prefix>/lib/cmake/<name>`), prefer <name>_ROOT

CMAKE_PREFIX_PATH Specify this when most packages are installed in same prefix

The environment variables `<name>_ROOT`, `<name>_PATH`, `<NAME>_PATH`, `<name>_DIR` are taken into account only if the corresponding CMake variables are unset.

Note, some packages are found via `Find<name>.cmake` and may have their own mechanism of finding paths with other variables, e.g. `<name>_HOME`. See the corresponing `Find<name>.cmake` file for datails, or use *cmake –help-module Find<name>* if it is a standard CMake-recognized module.

1.40.3 Usage

The search proceeds as follows:

1. If <name> is a subproject of the top-level project, search for `<name>-config.cmake` in `<name>_BINARY_DIR`.
2. If `Find<name>.cmake` exists in `CMAKE_MODULE_PATH`, search using it.
3. If any paths have been specified by the user via CMake or environment variables as given above:
   - search for `<name>-config.cmake` in those paths only
   - fail if the package was not found in any of those paths
   - **Search paths are in order from high to low priority:**
     - `<name>_DIR`
     - `<name>_ROOT`
     - `<name>_PATH`
     - `<NAME>_PATH`
     - `ENV{<name>_ROOT}`

- ENV{<name>_PATH}
- ENV{<NAME>_PATH}
- CMAKE_PREFIX_PATH
- ENV{<name>_DIR}
- ENV{CMAKE_PREFIX_PATH}
- system paths

See CMake documentation of `find_package()` for details on search

4. Fail if the package was not found and is REQUIRED.

1.41 ecbuild_find_package_search_hints

Detect more search hints and possibly add to <name>_ROOT

```
ecbuild_find_package_search_hints( NAME <name> )
```

This is called within ecbuild_find_package(). Alternatively it can be called anywhere before a standard find_package()

1.41.1 Motivation

Since CMake 3.12 the recommended approach to find_package is via <name>_ROOT which can be set both as variable or in the environment. Many environments still need to be adapted to this, as they are set up with the ecbuild 2 convention <name>_PATH or <NAME>_PATH. Furthermore this allows compatibility with <name>_ROOT for CMake versions < 3.12

1.41.2 Procedure

1) **If neither <name>_ROOT nor <name>_DIR are set in scope:** Try setting <name>_ROOT variable to first valid in list [<name>_PATH ; <NAME>_PATH]

2) **If 1) was not succesfull and neither <name>_ROOT nor <name>_DIR are set in environment:** Try setting <name>_ROOT variable to first valid in list [ENV{<name>_PATH} ; ENV{<NAME>_PATH}]

3) **Overcome CMake versions < 3.12 that do not yet recognize <name>_ROOT in scope or environment**

   If CMake version < 3.12:

   If <name>_DIR not defined in scope or environment, but <name>_ROOT IS defined in scope or environment Try setting <name>_DIR to a valid cmake-dir deduced from <name>_ROOT. Warning: Deduction is not feature-complete (it could be improved, but should now cover 99% of cases)

   It is advised to use CMake 3.12 instead.

1.42 ecbuild_find_perl

Find perl executable and its version.

```
ecbuild_find_perl( [ REQUIRED ] )
```

1.42.1 Options

REQUIRED [optional] fail if perl was not found

1.42.2 Output variables

The following CMake variables are set if perl was found:

PERL_FOUND perl was found

PERL_EXECUTABLE path to the perl executable

PERL_VERSION perl version

PERL_VERSION_STRING perl version (same as `PERL_VERSION`)

1.43 ecbuild_find_python

Find Python interpreter, its version and the Python libraries.

```
ecbuild_find_python( [ VERSION <version> ] [ REQUIRED ] [ NO_LIBS ] )
```

1.43.1 Options

VERSION [optional] minimum required version

REQUIRED [optional] fail if Python was not found

NO_LIBS [optional] only search for the Python interpreter, not the libraries

Unless `NO_LIBS` is set, the `python-config` utility, if found, is used to determine the Python include directories, libraries and link line. Set the CMake variable `PYTHON_NO_CONFIG` to use CMake's FindPythonLibs instead.

1.43.2 Output variables

The following CMake variables are set if python was found:

PYTHONINTERP_FOUND Python interpreter was found

PYTHONLIBS_FOUND Python libraries were found

PYTHON_FOUND Python was found (both interpreter and libraries)

PYTHON_EXECUTABLE Python executable

PYTHON_VERSION_MAJOR major version number

PYTHON_VERSION_MINOR minor version number

PYTHON_VERSION_PATCH patch version number

PYTHON_VERSION_STRING Python version

PYTHON_INCLUDE_DIRS Python include directories

PYTHON_LIBRARIES Python libraries

PYTHON_SITE_PACKAGES Python site packages directory

1.44 ecbuild_generate_config_headers

Generates the ecBuild configuration header for the project with the system introspection done by CMake.

```
ecbuild_generate_config_headers( [ DESTINATION <directory> ] )
```

1.44.1 Options

DESTINATION [optional] installation destination directory

1.45 ecbuild_generate_fortran_interfaces

Generates interfaces from Fortran source files.

```
ecbuild_generate_fortran_interfaces( TARGET <name>
                                     DESTINATION <path>
                                     DIRECTORIES <directory1> [<directory2> ...]
                                     [ PARALLEL <integer> ]
                                     [ INCLUDE_DIRS <name> ]
                                     [ GENERATED <name> ]
                                     [ SOURCE_DIR <path> ]
                                     [ SUFFIX <suffix> ]
                                     [ FCM_CONFIG_FILE <file> ]
                                   )
```

1.45.1 Options

TARGET [required] target name

DESTINATION [required] sub-directory of `CMAKE_CURRENT_BINARY_DIR` to install target to

DIRECTORIES [required] list of directories in `SOURCE_DIR` in which to search for Fortran files to be processed

PARALLEL [optional, defaults to 1] number of processes to use (always 1 on Darwin systems)

INCLUDE_DIRS [optional] name of CMake variable to store the path to the include directory containing the resulting interfaces

GENERATED [optional] name of CMake variable to store the list of generated interface files, including the full path to each

SOURCE_DIR [optional, defaults to `CMAKE_CURRENT_SOURCE_DIR`] directory in which to look for the subdirectories given as arguments to `DIRECTORIES`

SUFFIX [optional, defaults to ".intfb.h"] suffix to apply to name of each interface file

FCM_CONFIG_FILE [optional, defaults to the `fcm-make-interfaces.cfg` file in the ecbuild project] FCM configuration file to be used to generate interfaces

Usage

The listed directories will be recursively searched for Fortran files of the form `<fname>.[fF]`, `<fname>.[fF]90`, `<fname>.[fF]03` or `<fname>.[fF]08`. For each matching file, a file `<fname><suffix>` will be created containing the interface blocks for all external subprograms within it, where `<suffix>` is the value given to the `SUFFIX` option. If a file contains no such subprograms, no interface file will be generated for it.

1.46 ecbuild_generate_project_config

Generate the <project>-config.cmake file

```
ecbuild_generate_project_config(<template>
                                [FILENAME <filename>]
                                [PATH_VARS <var1> ...])
```

1.46.1 Options

<template> [required] path to the template to use

FILENAME [optional] name of the output file

PATH_VARS [optional] list of paths to be exported to the config template

1.46.2 Usage

The PATH_VARS parameter has the same meaning as for the configure_package_config_file macro in CMakePackageConfigHelpers: the value of ${varN} should be relative to the install directory (PROJECT_BINARY_DIR for build-dir export and INSTALL_PREFIX for the installed package). A reliable path will be computed and can be evaluated from the template through PACKAGE_${varN}.

1.47 ecbuild_generate_yy

Process lex/yacc files.

```
ecbuild_generate_yy( YYPREFIX <prefix>
                     YACC <file>
                     LEX <file>
                     DEPENDANT <file1> [ <file2> ... ]
                     [ SOURCE_DIR <dir> ]
                     [ OUTPUT_DIRECTORY <dir> ]
                     [ YACC_TARGET <file> ]
                     [ LEX_TARGET <file> ]
                     [ YACC_FLAGS <flags> ]
                     [ LEX_FLAGS <flags> ]
                     [ BISON_FLAGS <flags> ]
                     [ FLEX_FLAGS <flags> ] )
```

1.47.1 Options

YYPREFIX [required] prefix to use for file and function names

YACC [required] base name of the yacc source file (without .y extension)

LEX [required] base name of the lex source file (without .l extension)

DEPENDANT [required] list of files which depend on the generated lex and yacc target files At least one should be an existing source file (not generated itself).

SOURCE_DIR [optional, defaults to CMAKE_CURRENT_SOURCE_DIR] directory where yacc and lex source files are located

OUTPUT_DIRECTORY [optional, defaults to CMAKE_CURRENT_BINARY_DIR] output directory for yacc and lex target files

YACC_TARGET [optional, defaults to YACC] base name of the generated yacc target file (without .c extension)

LEX_TARGET [optional, defaults to LEX] base name of the generated lex target file (without .c extension)

YACC_FLAGS [optional, defaults to -t] flags to pass to yacc executable

LEX_FLAGS [optional] flags to pass to lex executable

BISON_FLAGS [optional, defaults to -t] flags to pass to bison executable

FLEX_FLAGS [optional, defaults to -l] flags to pass to flex executable

1.48 ecbuild_get_cxx11_flags

Set the CMake variable `${CXX11_FLAGS}` to the C++11 flags for the current compiler (based on macros from https://github.com/UCL/GreatCMakeCookOff).

```
ecbuild_get_cxx11_flags( CXX11_FLAGS )
```

1.49 ecbuild_get_date

Set the CMake variable `${DATE}` to the current date in the form YYYY.mm.DD.

```
ecbuild_get_date( DATE )
```

1.50 ecbuild_get_timestamp

Set the CMake variable `${TIMESTAMP}` to the current date and time in the form YYYYmmDDHHMMSS.

```
ecbuild_get_timestamp( TIMESTAMP )
```

1.51 ecbuild_get_test_data

Download a test data set at build time.

```
ecbuild_get_test_data( NAME <name>
                       [ TARGET <target> ]
                       [ DIRNAME <dir> ]
                       [ DIRLOCAL <dir> ]
                       [ MD5 <hash> ]
                       [ EXTRACT ]
                       [ NOCHECK ] )
```

curl or wget is required (curl is preferred if available).

1.51.1 Options

NAME [required] name of the test data file

TARGET [optional, defaults to test_data_<name>] CMake target name

DIRNAME [optional] use when there is a directory structure on the server that hosts test files

DIRLOCAL : optional, defaults to ".", local directory in which the test data is copied

MD5 [optional, ignored if NOCHECK is given] md5 checksum of the data set to verify. If not given and NOCHECK is *not* set, download the md5 checksum and verify

EXTRACT [optional] extract the downloaded file (supported archives: tar, zip, tar.gz, tar.bz2)

NOCHECK [optional] do not verify the md5 checksum of the data file

1.51.2 Usage

Download test data from `<ECBUILD_DOWNLOAD_BASE_URL>/<DIRNAME>/<NAME>`

If the `ECBUILD_DOWNLOAD_BASE_URL` variable is not set, the default URL `http://download.ecmwf.org/test-data` is used.

If the `DIRNAME` argument is not given, test data will be downloaded from `<ECBUILD_DOWNLOAD_BASE_URL>/<project>/<relative path to current dir>/<NAME>`

By default, the downloaded file is verified against an md5 checksum, either given as the `MD5` argument or downloaded from the server otherwise. Use the argument `NOCHECK` to disable this check.

The default timeout is 30 seconds, which can be overridden with `ECBUILD_DOWNLOAD_TIMEOUT`. Downloads are by default only tried once, use `ECBUILD_DOWNLOAD_RETRIES` to set the number of retries.

1.51.3 Examples

Do not verify the checksum:

```
ecbuild_get_test_data( NAME msl.grib NOCHECK )
```

Checksum agains remote md5 file:

```
ecbuild_get_test_data( NAME msl.grib )
```

Checksum agains local md5:

```
ecbuild_get_test_data( NAME msl.grib MD5 f69ca0929d1122c7878d19f32401abe9 )
```

1.52 ecbuild_get_test_multidata

Download multiple test data sets at build time.

```
ecbuild_get_test_multidata( NAMES <name1> [ <name2> ... ]
                            TARGET <target>
                            [ DIRNAME <dir> ]
                            [ DIRLOCAL <dir> ]
                            [ LABELS <label1> [<label2> ...] ]
                            [ EXTRACT ]
                            [ NOCHECK ] )
```

curl or wget is required (curl is preferred if available).

1.52.1 Options

NAMES [required] list of names of the test data files

TARGET [optional] CMake target name

DIRNAME [optional] use when there is a directory structure on the server that hosts test files

DIRLOCAL : optional, defaults to ".", local directory in which the test data is copied

LABELS [optional] list of labels to assign to the test

Lower case project name and `download_data` are always added as labels.

This allows selecting tests to run via `ctest -L <regex>` or tests to exclude via `ctest -LE <regex>`.

EXTRACT [optional] extract downloaded files (supported archives: tar, zip, tar.gz, tar.bz2)

NOCHECK [optional] do not verify the md5 checksum of the data file

1.52.2 Usage

Download test data from <ECBUILD_DOWNLOAD_BASE_URL>/<DIRNAME> for each name given in the list of NAMES. Each name may contain a relative path, which is appended to DIRNAME and may be followed by an md5 checksum, separated with a : (the name must not contain spaces).

If the ECBUILD_DOWNLOAD_BASE_URL variable is not set, the default URL http://download.ecmwf.org/test-data is used.

If the DIRNAME argument is not given, test data will be downloaded from <ECBUILD_DOWNLOAD_BASE_URL>/<project>/<relative path to current dir>/<NAME>

By default, each downloaded file is verified against an md5 checksum, either given as part of the name as described above or a remote checksum downloaded from the server. Use the argument NOCHECK to disable this check.

1.52.3 Examples

Do not verify checksums:

```
ecbuild_get_test_multidata( TARGET get_grib_data NAMES foo.grib bar.grib
                            DIRNAME test/data/dir NOCHECK )
```

Checksums agains remote md5 file:

```
ecbuild_get_test_multidata( TARGET get_grib_data NAMES foo.grib bar.grib
                            DIRNAME test/data/dir )
```

Checksum agains local md5:

```
ecbuild_get_test_multidata( TARGET get_grib_data DIRNAME test/data/dir
                            NAMES msl.grib:f69ca0929d1122c7878d19f32401abe9 )
```

1.53 ecbuild_git

Manages an external Git repository.

```
ecbuild_git( PROJECT <name>
             DIR <directory>
             URL <giturl>
             [ BRANCH <gitbranch> | TAG <gittag> ]
             [ UPDATE | NOREMOTE ]
             [ MANUAL ]
             [ RECURSIVE ] )
```

1.53.1 Options

PROJECT [required] project name for the Git repository to be managed

DIR [required] directory to clone the repository into (can be relative)

URL [required] Git URL of the remote repository to clone (see `git help clone`)

BRANCH [optional, cannot be combined with TAG] Git branch to check out

TAG [optional, cannot be combined with BRANCH] Git tag or commit id to check out

UPDATE [optional, requires BRANCH, cannot be combined with NOREMOTE] Create a CMake target update to fetch changes from the remote repository

NOREMOTE [optional, cannot be combined with UPDATE] Do not fetch changes from the remote repository

MANUAL [optional] Do not automatically switch branches or tags

RECURSIVE [optional] Do a recursive fetch or update

1.54 ecbuild_install_project

Set up packaging and export configuration.

```
ecbuild_install_project( NAME <name> [ DESCRIPTION <description> ] )
```

1.54.1 Options

NAME [required] project name used for packaging

DESCRIPTION [optional] project description used for packaging

1.54.2 Usage

`ecbuild_install_project` should be called at the very end of any ecBuild project (only followed by `ecbuild_print_summary`), sets up packaging of the project with cpack and exports the configuration and targets for other projects to use.

Unless ECBUILD_SKIP_<PROJECT_NAME>_EXPORT is set, the following files are generated:

<project>-config.cmake default project configuration

<project>-config-version.cmake project version number

<project>-targets.cmake exported targets

<project>-import.cmake extra project configuration (optional)

<project>-post-import.cmake extra project configuration (optional)

For `<project>-import.cmake` to be exported to build and install tree, `<project>-import.cmake` or `<project>-import.cmake.in` must exist in the source tree. The same applies for `<project>-post-import.cmake`. The 'import' file is included before defining the targets (e.g. to call find_dependency), whereas the 'post-import' file is included after (e.g. to

define aliases). `<project>-config.cmake.in` and `<project>-config-version.cmake.in` can be provided in the source tree to override the default templates used to generate `<project>-config.cmake` and `<project>-config-version.cmake`.

If the project is added as a subdirectory, the following CMake variables are set in the parent scope:

<PROJECT_NAME>_FOUND set to `TRUE`

<PROJECT_NAME>_VERSION version string

<PROJECT_NAME>_FEATURES list of enabled features

<PROJECT_NAME>_HAVE_<FEATURE> set to 1 for each enabled features

1.55 ecbuild_list_add_pattern

Exclude items from a list that match a list of patterns.

```
ecbuild_list_add_pattern( LIST <input_list>
                          GLOB <pattern1> [ <pattern2> ... ]
                          [ SOURCE_DIR <source_dir> ]
                          [ QUIET ] )
```

1.55.1 Options

LIST [required] list variable to be appended to

GLOB [required] Regex pattern of exclusion

SOURCE_DIR [optional] Directory from where to start search

QUIET [optional] Don't warn if patterns don't match

1.56 ecbuild_list_exclude_pattern

Exclude items from a list that match a list of patterns.

```
ecbuild_list_exclude_pattern( LIST <input_list>
                              REGEX <regex1> [ <regex2> ... ]
                              [ QUIET ] )
```

1.56.1 Options

LIST [required] list variable to be cleaned

REGEX [required] Regex pattern of exclusions

QUIET [optional] Don't warn if patterns don't match

1.57 Logging

ecBuild provides functions for logging based on a log level set by the user, similar to the Python logging module:

ecbuild_debug logs a `STATUS` message if log level <= `DEBUG`

ecbuild_info logs a `STATUS` message if log level <= `INFO`

ecbuild_warn logs a `WARNING` message if log level <= `WARN`

ecbuild_error logs a `SEND_ERROR` message if log level <= `ERROR`

ecbuild_critical logs a `FATAL_ERROR` message if log level <= `CRITICAL`

ecbuild_deprecate logs a `DEPRECATION` message as a warning enable CMAKE_ERROR_DEPRECATED to raise an error instead disable CMAKE_WARN_DEPRECATED to hide deprecations

Furthermore there are auxilliary functions for outputting CMake variables, CMake lists and environment variables if the log level is `DEBUG`:

ecbuild_debug_var logs given CMake variables if log level <= `DEBUG`

ecbuild_debug_list logs given CMake lists if log level <= `DEBUG`

ecbuild_debug_env_var logs given environment variables if log level <= `DEBUG`

ecbuild_debug_property logs given global CMake property if log level <= `DEBUG`

1.58 ecbuild_parse_version

Parse version string of the form "<major>[.<minor>[.<patch>[.<tweak>]]][<suffix>]"

```
ecbuild_parse_version( <version_str> [ PREFIX <prefix> ] )
```

1.58.1 Options

PREFIX [optional] string to be prefixed to all defined variables. If not given, the value "_" will be used.

1.58.2 Notes

Following variables if possible:

<prefix>_VERSION_STR = <major>[.<minor>[.<patch>[.<tweak>]]][<suffix>] <prefix>_VERSION = <major>[.<minor>[.<patch>[.<tweak>]]] <prefix>_VERSION_MAJOR = <major> <prefix>_VERSION_MINOR = <minor> <prefix>_VERSION_PATCH = <patch> <prefix>_VERSION_TWEAK = <tweak> <prefix>_VERSION_SUFFIX = <suffix>

1.59 ecbuild_parse_version_file

Parse version string of the form "<major>[.<minor>[.<patch>[.<tweak>]]][<suffix>]" contained in a file

```
ecbuild_parse_version_file( <file> [ PREFIX <prefix> ] )
```

1.59.1 Options

PREFIX [optional] string to be prefixed to all defined variables. If not given, the value "_" will be used.

1.59.2 Notes

Following variables if possible:

<prefix>_VERSION_STR = <major>[.<minor>[.<patch>[.<tweak>]]][<suffix>] <prefix>_VERSION = <major>[.<minor>[.<patch>[.<tweak>]]] <prefix>_VERSION_MAJOR = <major> <prefix>_VERSION_MINOR = <minor> <prefix>_VERSION_PATCH = <patch> <prefix>_VERSION_TWEAK = <tweak> <prefix>_VERSION_SUFFIX = <suffix>

1.60 ecbuild_pkgconfig

Create a pkg-config file for the current project.

```
ecbuild_pkgconfig( [ NAME <name> ]
                   [ FILENAME <filename> ]
                   [ TEMPLATE <template> ]
                   [ URL <url> ]
                   [ DESCRIPTION <description> ]
                   [ LIBRARIES <lib1> [ <lib2> ... ] ]
                   [ IGNORE_INCLUDE_DIRS <dir1> [ <dir2> ... ] ]
                   [ IGNORE_LIBRARIES <lib1> [ <lib2> ... ] ]
                   [ LANGUAGES <language1> [ <language2> ... ] ]
                   [ VARIABLES <variable1> [ <variable2> ... ] ]
                   [ NO_PRIVATE_INCLUDE_DIRS ] )
```

1.60.1 Options

NAME [optional, defaults to lower case name of the project] name to be given to the package

FILENAME [optional, defaults to `<NAME>.pc`] file to be generated, including .pc extension

TEMPLATE [optional, defaults to `${ECBUILD_CMAKE_DIR}/pkg-config.pc.in`] template configuration file to use

This is useful to create customised pkg-config files.

URL [optional, defaults to `${PROJECT_NAME}_URL`] url of the package

DESCRIPTION [optional, defaults to `${PROJECT_NAME}_DESCRIPTION`] description of the package

LIBRARIES [required] list of package libraries

IGNORE_INCLUDE_DIRS [optional] list of include directories to ignore

IGNORE_LIBRARIES [optional] list of libraries to ignore i.e. those are removed from `LIBRARIES`

VARIABLES [optional] list of additional CMake variables to export to the pkg-config file

LANGUAGES [optional, defaults to all loaded languages] list of languages to use. Accepted languages: C CXX Fortran

NO_PRIVATE_INCLUDE_DIRS do not add include directories of dependencies to Cflags

This is mainly useful for Fortran only packages, when only modules need to be added to Cflags.

1.60.2 Input variables

The following CMake variables are used as default values for some of the options listed above:

<PROJECT_NAME>_DESCRIPTION package description

<PROJECT_NAME>_URL package URL

<PROJECT_NAME>_VERSION package version

<PROJECT_NAME>_GIT_SHA1 Git revision

1.60.3 Usage

It is good practice to provide a separate pkg-config file for each library a package exports. This can be achieved as follows:

```
foreach( _lib ${${PNAME}_LIBRARIES} )
  if( TARGET ${_lib} )
    ecbuild_pkgconfig( NAME ${_lib}
                       DESCRIPTION "..."
                       URL "..."
                       LIBRARIES ${_lib} )
  endif()
endforeach()
```

1.61 ecbuild_print_summary

Print a summary of the project, build environment and enabled features.

```
ecbuild_print_summary()
```

If `project_summary.cmake` exist in the source root directory, a project summary is printed by including this file.

For a top level project, a summary of the build environment and a feature summary are also printed.

1.62 ecbuild_regex_escape

Escape regular expression special characters from the input string.

```
ecbuild_regex_escape(<string> <output_variable>)
```

1.63 ecbuild_remove_fortran_flags

Remove Fortran compiler flags from CMAKE_Fortran_FLAGS.

```
ecbuild_remove_fortran_flags( <flag1> [ <flag2> ... ] [ BUILD <build> ] )
```

1.63.1 Options

BUILD [optional] remove flags from CMAKE_Fortran_FLAGS_<build> instead of CMAKE_Fortran_FLAGS

1.64 ecbuild_requires_macro_version

Check that the ecBuild version satisfied a given minimum version or fail.

```
ecbuild_requires_macro_version( <minimum-version> )
```

1.65 ecbuild_separate_sources

Separate a given list of sources according to language.

```
ecbuild_separate_sources( TARGET <name>
                          SOURCES <source1> [ <source2> ... ] )
```

1.65.1 Options

TARGET [required] base name for the CMake output variables to set

SOURCES [required] list of source files to separate

1.65.2 Output variables

If any file of the following group of extensions is present in the list of sources, the corresponding CMake variable is set:

<target>_h_srcs source files with extension .h, .hxx, .hh, .hpp, .H .tcc .txx .tpp

<target>_c_srcs source files with extension .c

<target>_cxx_srcs source files with extension .cc, .cxx, .cpp, .C

<target>_fortran_srcs source files with extension .f, .F, .for, f77, .f90, .f95, .F77, .F90, .F95

<target>_cuda_srcs source files with extension .cu

1.66 ecbuild_target_flags

Override compiler flags for a given target.

```
ecbuild_target_flags( <target> <c_flags> <cxx_flags> <fortran_flags> )
```

Required arguments:

target Target name

c_flags Target specific C flags (can be empty)

cxx_flags Target specific CXX flags (can be empty)

fortran_flags Target specific Fortran flags (can be empty)

There are 3 cases, only the first applicable case takes effect:

1. Use custom rules from user specified ECBUILD_COMPILE_FLAGS file and append target specific flags.
2. Use JSON rules from user specified ECBUILD_SOURCE_FLAGS file and append target specific flags.
3. Only the target specific flags are applied to all matching source files.

1.67 ecbuild_try_run

Try compiling and then running some code.

```
ecbuild_try_run( <run_result_var> <compile_result_var>
                 <bindir> <srcfile>
                 [ CMAKE_FLAGS <flag> [ <flag> ... ] ]
                 [ COMPILE_DEFINITIONS <def> [ <def> ... ] ]
                 [ LINK_LIBRARIES <lib> [ <lib> ... ] ]
                 [ COMPILE_OUTPUT_VARIABLE <var> ]
                 [ RUN_OUTPUT_VARIABLE <var> ]
                 [ OUTPUT_VARIABLE <var> ]
                 [ ARGS <arg> [ <arg> ... ] ] )
```

Try compiling a <srcfile>. Returns TRUE or FALSE for success or failure in <compile_result_var>. If the compile succeeded, runs the executable and returns its exit code in <run_result_var>. If the executable was built, but failed to run, then <run_result_var> will be set to FAILED_TO_RUN. See the CMake try_compile command for information on how the test project is constructed to build the source file.

1.67.1 Options

CMAKE_FLAGS [optional] Specify flags of the form `-DVAR:TYPE=VALUE` to be passed to the `cmake` command-line used to drive the test build.

The example in CMake's `try_compile` shows how values for variables `INCLUDE_DIRECTORIES`, `LINK_DIRECTORIES`, and `LINK_LIBRARIES` are used.

COMPILE_DEFINITIONS [optional] Specify `-Ddefinition` arguments to pass to `add_definitions` in the generated test project.

COMPILE_OUTPUT_VARIABLE [optional] Report the compile step build output in a given variable.

LINK_LIBRARIES [optional] Specify libraries to be linked in the generated project. The list of libraries may refer to system libraries and to Imported Targets from the calling project.

If this option is specified, any `-DLINK_LIBRARIES=...` value given to the `CMAKE_FLAGS` option will be ignored.

OUTPUT_VARIABLE [optional] Report the compile build output and the output from running the executable in the given variable. This option exists for legacy reasons. Prefer `COMPILE_OUTPUT_VARIABLE` and `RUN_OUTPUT_VARIABLE` instead.

RUN_OUTPUT_VARIABLE [optional] Report the output from running the executable in a given variable.

1.67.2 Other Behavior Settings

Set the `CMAKE_TRY_COMPILE_CONFIGURATION` variable to choose a build configuration.

1.67.3 Behavior when Cross Compiling

When cross compiling, the executable compiled in the first step usually cannot be run on the build host. The `try_run` command checks the `CMAKE_CROSSCOMPILING` variable to detect whether CMake is in cross-compiling mode. If that is the case, it will still try to compile the executable, but it will not try to run the executable unless the `CMAKE_CROSSCOMPILING_EMULATOR` variable is set. Instead it will create cache variables which must be filled by the user or by presetting them in some CMake script file to the values the executable would have produced if it had been run on its actual target platform. These cache entries are:

<RUN_RESULT_VAR> Exit code if the executable were to be run on the target platform.

<RUN_RESULT_VAR>__TRYRUN_OUTPUT Output from stdout and stderr if the executable were to be run on the target platform. This is created only if the `RUN_OUTPUT_VARIABLE` or `OUTPUT_VARIABLE` option was used.

In order to make cross compiling your project easier, use `try_run` only if really required. If you use `try_run`, use the `RUN_OUTPUT_VARIABLE` or `OUTPUT_VARIABLE` options only if really required. Using them will require that when cross-compiling, the cache variables will have to be set manually to the output of the executable. You can also "guard" the calls to `try_run` with an `if` block checking the `CMAKE_CROSSCOMPILING` variable and provide an easy-to-preset alternative for this case.

1.68 ecbuild_warn_unused_files

Print warnings about unused source files in the project.

```
ecbuild_warn_unused_files()
```

If the CMake variable CHECK_UNUSED_FILES is set, ecBuild will keep track of any source files (.c, .cc, .cpp, .cxx) which are not part of a CMake target. If set, this macro reports unused files if any have been found. This is considered a fatal error unless UNUSED_FILES_LEVEL is set to a value different from ERROR.

Note: Enabling CHECK_UNUSED_FILES can slow down the CMake configure time considerably!

CHAPTER

TWO

ECBUILD FIND PACKAGE HELPERS

2.1 FindFFTW

Find the FFTW library.

```
find_package(FFTW [REQUIRED] [QUIET]
             [COMPONENTS [single] [double] [long_double] [quad]])
```

By default, search for the double precision library `fftw3`

2.1.1 Search procedure

1) FFTW_LIBRARIES and FFTW_INCLUDE_DIRS set by user –> Nothing is searched and these variables are used instead
2) Find MKL implementation via FFTW_ENABLE_MKL –> If FFTW_ENABLE_MKL is explicitely set to ON, only MKL is considered

   If FFTW_ENABLE_MKL is explictely set to OFF, MKL will not be considered If FFTW_ENABLE_MKL is undefined, MKL is preferred

   –> MKLROOT environment variable helps to detect MKL (See FindMKL.cmake)
3) Find official FFTW impelementation –> FFTW_ROOT variable / environment variable helps to detect FFTW

2.1.2 Components

If a different version or multiple versions of the library are required, these need to be specified as `COMPONENTS`. Note that double must be given explicitly if any `COMPONENTS` are specified.

The libraries corresponding to each of the `COMPONENTS` are:

single `FFTW::fftw3f`

double `FFTW::fftw3`

long_double `FFTW::fftw3l`

quad `FFTW::fftw3q`

2.1.3 Output variables

The following CMake variables are set on completion:

FFTW_FOUND true if FFTW is found on the system

FFTW_LIBRARIES full paths to requested FFTW libraries

FFTW_INCLUDE_DIRS FFTW include directory

2.1.4 Input variables

The following CMake variables are checked by the function:

FFTW_USE_STATIC_LIBS if true, only static libraries are found

FFTW_ROOT if set, this path is exclusively searched

FFTW_DIR equivalent to FFTW_ROOT (deprecated)

FFTW_PATH equivalent to FFTW_ROOT (deprecated)

FFTW_LIBRARIES User overriden FFTW libraries

FFTW_INCLUDE_DIRS User overriden FFTW includes directories

FFTW_ENABLE_MKL User requests use of MKL implementation

2.2 FindJemalloc

Find the Jemalloc library.

```
find_package( Jemalloc [REQUIRED] [QUIET] )
```

2.2.1 Output variables

The following CMake variables are set on completion:

Jemalloc_FOUND true if Jemalloc is found on the system

JEMALLOC_LIBRARIES full paths to requested Jemalloc libraries

JEMALLOC_INCLUDE_DIRS Jemalloc include directory

2.2.2 Input variables

The following CMake and environment variables are considered:

Jemalloc_ROOT

2.3 FindTcmalloc

Find the Tcmalloc library.

```
find_package( Tcmalloc [REQUIRED] [QUIET] )
```

2.3.1 Output variables

The following CMake variables are set on completion:

Tcmalloc_FOUND true if Tcmalloc is found on the system

TCMALLOC_LIBRARIES full paths to requested Tcmalloc libraries

TCMALLOC_LIBRARY_DIR Directory containing the TCMALLOC_LIBRARIES

TCMALLOC_INCLUDE_DIRS Tcmalloc include directories

2.3.2 Input variables

The following CMake / Environment variables are considered in order:

Tcmalloc_ROOT CMake variable / Environment variable

CHAPTER

THREE

ECBUILD THIRD PARTY SCRIPTS