CodesUI - Installation guide

Overview

CodesUI uses **CMake** for its compilation and installation. This is part of the process of homogenising the installation procedures for all ECMWF packages.

CMake installation instructions

The **CMake** build system is used to build ECMWF software. The build process comprises two stages:

1. CMake runs some tests on the system and finds out if required software libraries and headers are available. It uses this information to create native build tools (e.g. Makefiles) for the current platform.
2. The actual build can take place, for example by typing '`make`'.

Prerequisite

To install any ECMWF software package, CMake needs to be installed on your system. On most systems it will be already installed or this can be done through the standard package manager to install software. For further information to install CMake see

> http://www.cmake.org/cmake/help/install.html

Directories

During a build with CMake there are three different directories involved: The **source dir**, the **build dir** and the **install dir**.

Source	Contains the software's source code. This is where a source tarball should be extracted to.	`/tmp/src/sw-package`
Build	Configuration and compiler outputs are generated here, including libraries and executables.	`/tmp/build/sw-package`
Install	Where the software will actually be used from. Installation to this directory is the final stage.	`/usr/local`

Of these, the source and build directories can be anywhere on the system. The installation directory is usually left at its default, which is `/usr/local`. Installing software here ensures that it is automatically available to users. It is possible to specify a different installation directory by adding `-DCMAKE_INSTALL_PREFIX=/path/to/install/dir` to the CMake command line.

ECMWF software does **not** support in-source builds. Therefore the build directory **cannot** be (a subdirectory of) the source directory.

Quick Build Example

Here is an example set of commands to set up and build a software package using default settings. More detail for a customised build is given below.

```
# unpack the source tarball into a temporary directory
mkdir -p /tmp/src
cd /tmp/src
tar xzvf software-version-Source.tar.gz

# configure and build in a separate directory
mkdir -p /tmp/build
cd /tmp/build
cmake /tmp/src/software-version-Source
make
```

On a machine with multiple cores, compilation will be faster by specifying the number of cores to be used simultaneously for the build, for example:

```
make -j8
```

If the `make` command fails, you can get more output by typing:

```
make VERBOSE=1
```

If the build is successful, you can install the software:

```
make install
```

General CMake options

Various options can be passed to the CMake command. The following table gives an overview of some of the general options that can be used. Options are passed to the `cmake` command by prefixing them with **-D**, for example **-DCMAKE_INSTALL_PREFIX=/path/to/dir**.

`CMAKE_INSTALL_PREFIX`	where to install the software	`/usr/local`
`CMAKE_BUILD_TYPE`	to select the type of compilation: • Debug • RelWithDebInfo • Release • Production	RelWithDebInfo (release with debug info)
`CMAKE_CXX_FLAGS`	Additional flags to pass to the C++ compiler	
`CMAKE_C_FLAGS`	Additional flags to pass to the C compiler	

The C and C++ compilers are chosen by CMake. This can be overwritten by setting the environment variables CC and CXX, before the call to `cmake`, to set the preferred compiler. Further the variable `CMAKE_CXX_FLAGS` can be used to set compiler flags for optimisation or debugging. For example, using `CMAKE_CXX_FLAGS="-O2 -mtune=native"` sets options for better optimisation.

Finding support libraries

If any support libraries are installed in non-default locations, CMake can be instructed where to find them by one of the following methods. First, the option CMAKE_PREFIX_PATH can be set to a colon-separated list of base directories where the libraries are installed, for example `-DCMAKE_PREFIX_PATH=/path/where/my/sw/is/installed`. CMake will check these directories for any package it requires. This method is therefore useful if many support libraries are installed into the same location.

Debugging configure failures

If CMake fails to configure your project, run with debug logging first:

```
cmake -DECBUILD_LOG_LEVEL=DEBUG [...] /path/to/source
```

This will output lots of diagnostic information (in blue) on discovery of dependencies and much more.

Requirements to build CodesUI

The following table lists the dependencies CodesUI requires to be built from source. Please note, if you install these package from source you also might have to install the respective "-devel" packages.

Compilers		
C++	http://gcc.gnu.org/	
Utilities		
make	http://www.gnu.org/software/make/	
Third party packages (best installed through system package manager)		
Qt	http://www.qt.io/	minimum **version 5.0.0 of Qt** is required
bash	https://www.gnu.org/software/bash/	
ECMWF libraries		
ecCodes	ecCodes	minimum **version 2.6.0. of ecCodes** is required

CMake options used in CodesUI

CMake options are passed to the `cmake` command by prefixing them with **-D**, for example **-DENABLE_QT_DEBUG=ON**.

CMake option	Description	Default

ENABLE_QT_DEBUG	outputs additional log messages from Qt-based modules	OFF
Path options - only required when support libraries are not installed in default locations		
CMake Option	**Description**	**Notes**
ECCODES_PATH	path to where ecCodes has been installed	
CMAKE_PREFIX_PATH	might be required if the Qt5 libraries are not found by default. Then they ican be specified like this: `-DCMAKE_PREFIX_PATH=/path/to/qt5/`	