The Metview Source Bundle

Overview

The Metview source bundle is a single package containing the source code for Metview, Magics, ecCodes and ODB API. It allows for a simpler way to build and install Metview and the ECMWF software on which it depends. Instead of downloading, configuring, building, testing and installing each package separately, the bundle allows these steps to be performed for just one software package - the bundle itself. Building the bundle triggers the building of each of the component packages automatically, and they will 'see' each other, so for example Metview will be linked with the ecCodes which is part of the bundle, with no need to point it to an already-installed version.

Please follow the general CMake instructions below, and refer to the installation instructions of the individual packages for the tables of available CMake options for each. Options given to CMake will be passed to every component package of the bundle. Certain of these are set by default, for example Magics will be built with Metview and BUFR support enabled.

Note that the bundle has two additional CMake options:

CMake Option	Description	Default
ENABLE_EXPOSE_SUBPACKAGES	Configures the sub-packages to be installed into the same directory level as Metview. If set to OFF, they will be installed into a sub-directory; this option could be useful in order to avoid clashing with pre-existing installations of, for example, ecCodes.	OFF
~~ENABLE_ECCODES~~ Please note: GRIB_API is no longer included in the bundle since the 2018.02.0 bundle release	~~Set this to OFF to build the bundle with the supplied GRIB_API instead of with the supplied ecCodes.~~	~~ON~~
ENABLE_ODB	Set this to ON to enable recognition and processing of ODB data with ODC	OFF

Download

Release 2024.4.0	Contains	Version	Change log
MetviewBundle-2024.4.0-Source.tar.gz	ecCodes	2.35.0	Latest news
	ODC	1.5.0	Change history
	Magics	4.15.4	Latest News
	Metview	5.22.0	Latest News

Release 2024.2.1	Contains	Version	Change log
MetviewBundle-2024.2.1-Source.tar.gz	ecCodes	2.34.1	Latest news
	ODC	1.5.0	Change history
	Magics	4.15.3	Latest News
	Metview	5.21.2	Latest News

Release 2024.2.0	Contains	Version	Change log
MetviewBundle-2024.2.0-Source.tar.gz	ecCodes	2.34.0	Latest news
	ODC	1.5.0	Change history
	Magics	4.15.2	Latest News
	Metview	5.21.1	Latest News

Release 2023.12.0	Contains	Version	Change log
MetviewBundle-2023.12.0-Source.tar.gz	ecCodes	2.33.0	Latest news
	ODC	1.5.0	Change history
	Magics	4.15.0	Latest News
	Metview	5.21.0	Latest News

Release 2023.10.0	Contains	Version	Change log

MetviewBundle-2023.10.0-Source.tar.gz	ecCodes	2.32.0	Latest news
	ODC	1.4.6	Change history
	Magics	4.14.2	Latest News
	Metview	5.20.0	Latest News

Release 2023.07.0	Contains	Version	Change log
MetviewBundle-2023.7.0-Source.tar.gz	ecCodes	2.31.0	Latest news
	ODC	1.4.6	Change history
	Magics	4.14.1	Latest News
	Metview	5.19.2	Latest News

Release 2023.04.1	Contains	Version	Change log
MetviewBundle-2023.4.1-Source.tar.gz	ecCodes	2.30.2	Latest news
	ODC	1.4.6	Change history
	Magics	4.13.0	Latest News
	Metview	5.19.1	Latest News

Release 2023.04.0	Contains	Version	Change log
MetviewBundle-2023.4.0-Source.tar.gz	ecCodes	2.30.0	Latest news
	ODC	1.4.6	Change history
	Magics	4.13.0	Latest News
	Metview	5.19.0	Latest News

Release 2023.01.0	Contains	Version	Change log
MetviewBundle-2023.1.0-Source.tar.gz	ecCodes	2.28.0	Latest news
	ODC	1.4.6	Change history
	Magics	4.13.0	Latest News
	Metview	5.18.0	Latest News

Release 2022.08.4	Contains	Version	Change log
MetviewBundle-2022.8.4-Source.tar.gz	ecCodes	2.27.1	Latest news
	ODC	1.4.6	Change history
	Magics	4.12.1	Latest News
	Metview	5.17.4	Latest News

Release 2022.08.3	Contains	Version	Change log
MetviewBundle-2022.8.3-Source.tar.gz	ecCodes	2.27.0	Latest news
	ODC	1.4.6	Change history
	Magics	4.12.1	Latest News
	Metview	5.17.3	Latest News

Release 2022.08.0	Contains	Version	Change log
MetviewBundle-2022.8.0-Source.tar.gz	ecCodes	2.27.0	Latest news
	ODC	1.4.6	Change history
	Magics	4.12.1	Latest News
	Metview	5.17.0	Latest News

Release 2022.05.0	Contains	Version	Change log

MetviewBundle-2022.5.0-Source.tar.gz	ecCodes	2.26.0	Latest news
	ODC	1.4.5	Change history
	Magics	4.12.0	Latest News
	Metview	5.16.0	Latest News

Release 2022.03.0	Contains	Version	Change log
MetviewBundle-2022.3.0-Source.tar.gz	ecCodes	2.25.0	Latest news
	ODC	1.4.4	Change history
	Magics	4.11.0	Latest News
	Metview	5.15.0	Latest News

Release 2021.12.1	Contains	Version	Change log
MetviewBundle-2021.12.1-Source.tar.gz	ecCodes	2.24.2	Latest news
	ODC	1.4.4	Change history
	Magics	4.10.1	Latest News
	Metview	5.14.1	Latest News

Release 2021.12.0	Contains	Version	Change log
MetviewBundle-2021.12.0-Source.tar.gz	ecCodes	2.24.0	Latest news
	ODC	1.4.4	Change history
	Magics	4.10.0	Latest News
	Metview	5.14.0	Latest News

Release 2021.08.0	Contains	Version	Change log
MetviewBundle-2021.8.0-Source.tar.gz	ecCodes	2.23.0	Latest news
	ODC	1.4.1	Change history
	Magics	4.9.0	Latest News
	Metview	5.13.0	Latest News

Release 2021.05.1	Contains	Version	Change log
MetviewBundle-2021.5.1-Source.tar.gz	ecCodes	2.22.1	Latest news
	ODC	1.3.0	Change history
	Magics	4.8.2	Latest News - archive
	Metview	5.12.1	Version 5.12 Updates

Release 2021.05.0	Contains	Version	Change log
MetviewBundle-2021.5.0-Source.tar.gz	ecCodes	2.22.0	Latest news
	ODC	1.3.0	Change history
	Magics	4.8.0	Latest News - archive
	Metview	5.12.0	Version 5.12 Updates

Release 2021.03.1	Contains	Version	Change log
MetviewBundle-2021.3.1-Source.tar.gz	ecCodes	2.21.0	Latest news
	ODC	1.3.0	Change history
	Magics	4.6.0	Latest News - archive
	Metview	5.11.1	Version 5.11 Updates

Release 2021.03.0	Contains	Version	Change log

MetviewBundle-2021.3.0-Source.tar.gz	ecCodes	2.21.0	Latest news
	ODC	1.3.0	Change history
	Magics	4.6.0	Latest News - archive
	Metview	5.11.0	Version 5.11 Updates

Release 2021.01.0	Contains	Version	Change log
MetviewBundle-2021.01.0-Source.tar.gz	ecCodes	2.20.0	Latest news
	ODC	1.2.0	Change history
	Magics	4.5.3	Latest News - archive
	Metview	5.10.2	Version 5.10 Updates

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**.

Directory	Use	Example
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
```

The software distribution will include a small set of tests which can help ensure that the build was successful. To start the tests, type:

```
ctest
```

As before if you have multiple cores, you can run the tests in parallel by:

```
ctest -j8
```


Some projects might not be set up to run tests in parallel. If you experience test failures, run the tests sequentially.

If the tests are 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 Option	Description	Default
`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	
`CMAKE_Fortran_FLAGS`	Additional flags to pass to the Fortran compiler	

The C, C++ and Fortran compilers are chosen by CMake. This can be overwritten by setting the environment variables CC, CXX and F77, 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.

Troubleshooting

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 the Metview bundle

The following table lists the dependencies that the bundle requires to be built from source. Please note that when you install these packages you also might have to install the respective "-devel" packages.

Compilers		
C++	http://gcc.gnu.org/	
Fortran	http://gcc.gnu.org/fortran/	
Utilities		
make	http://www.gnu.org/software/make/	
Third party libraries		
Qt	http://www.qt.io/	if Metview's user interface is required (version 4.6.2 or later of Qt is needed). *Note that on some systems it is also necessary to install the libQtWebKit-devel development package (it may have different names on different systems)*
gdbm	http://www.gnu.org.ua/software/gdbm/	
bash	https://www.gnu.org/software/bash/	
ImageMagick	http://www.imagemagick.org/script/index.php	if Metview's user interface is required
proj	http://trac.osgeo.org/proj/	
netcdf 4	http://www.unidata.ucar.edu/software/netcdf/	Please note: You also need to install the legacy C++ interface and HDF5
cairo	https://www.cairographics.org/	if png/pdf support needed
pango	http://www.pango.org/	if png/pdf support needed
expat	http://expat.sourceforge.net/	

Overview

The Metview source bundle is a single package containing the source code for Metview, Magics, ecCodes and ODB API. It allows for a simpler way to build and install Metview and the ECMWF software on which it depends. Instead of downloading, configuring, building, testing and installing each package separately, the bundle allows these steps to be performed for just one software package - the bundle itself. Building the bundle triggers the building of each of the component packages automatically, and they will 'see' each other, so for example Metview will be linked with the ecCodes which is part of the bundle, with no need to point it to an already-installed version.

Please follow the general CMake instructions below, and refer to the installation instructions of the individual packages for the tables of available CMake options for each. Options given to CMake will be passed to every component package of the bundle. Certain of these are set by default, for example Magics will be built with Metview and BUFR support enabled.

Note that the bundle has two additional CMake options:

CMake Option	Description	Default
ENABLE_EXPOSE_SUBPACKAGES	Configures the sub-packages to be installed into the same directory level as Metview. If set to OFF, they will be installed into a sub-directory; this option could be useful in order to avoid clashing with pre-existing installations of, for example, ecCodes.	OFF
~~ENABLE_ECCODES~~ Please note: GRIB_API is no longer included in the bundle since the 2018.02.0 bundle release	~~Set this to OFF to build the bundle with the supplied GRIB_API instead of with the supplied ecCodes.~~	~~ON~~
ENABLE_ODB	Set this to ON to enable recognition and processing of ODB data with ODC	OFF

Download

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**.

Directory	Use	Example
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
```

The software distribution will include a small set of tests which can help ensure that the build was successful. To start the tests, type:

```
ctest
```

As before if you have multiple cores, you can run the tests in parallel by:

```
ctest -j8
```


Some projects might not be set up to run tests in parallel. If you experience test failures, run the tests sequentially.

If the tests are 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 Option	Description	Default
`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	

`CMAKE_Fortran_FLAGS`	Additional flags to pass to the Fortran compiler	

The C, C++ and Fortran compilers are chosen by CMake. This can be overwritten by setting the environment variables CC, CXX and F77, 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.

Troubleshooting

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 the Metview bundle

The following table lists the dependencies that the bundle requires to be built from source. Please note that when you install these packages you also might have to install the respective "-devel" packages.

Compilers		
C++	http://gcc.gnu.org/	
Fortran	http://gcc.gnu.org/fortran/	
Utilities		
make	http://www.gnu.org/software/make/	
Third party libraries		
Qt	http://www.qt.io/	if Metview's user interface is required (version 4.6.2 or later of Qt is needed). *Note that on some systems it is also necessary to install the libQtWebKit-devel development package (it may have different names on different systems)*
gdbm	http://www.gnu.org.ua/software/gdbm/	
bash	https://www.gnu.org/software/bash/	
ImageMagick	http://www.imagemagick.org/script/index.php	if Metview's user interface is required
proj	http://trac.osgeo.org/proj/	
netcdf 4	http://www.unidata.ucar.edu/software/netcdf/	Please note: You also need to install the legacy C++ interface and HDF5
cairo	https://www.cairographics.org/	if png/pdf support needed
pango	http://www.pango.org/	if png/pdf support needed
expat	http://expat.sourceforge.net/	