Installation Guide

Binary versions

Before you install from source code you might want to check that already compiled binary versions are available to you. Magics and third-party dependent software packages might be available as binary packages for you platform in form of RPMs or Debian packages for Linux. Ubuntu maintains a Magics version in their system default repository.

Installing Magics through conda and pip

If you want to use Magics only through Python you have now choices to install Magics with your favourite Python package manager. With the release of Magics 4.0.0 (Feb 2019), the Python interface is separated from the library. This allowed the packaging through *pip* and *conda*.

Using pip

When using pip it is required to have the Magics library installed on the system!

```
pip install Magics
```

Ref: https://pypi.org/project/Magics/

Using conda

Conda will install the Magics library and all its dependencies for you. Please make sure to activate you conda environment before running your python program.

```
conda install -c conda-forge Magics
conda activate
python my-magics-script.py
```

Building Magics from source yourself

Requirements

The following table lists the dependency Magics requires to be build from source. **Please note, if you install this package from source you also might have to install the respective "-devel" packages of dependencies.**

Compilers		
C++	http://gcc.gnu.org/	The compiler must support C++ 17. GCC supports it from version 7
Fortran	http://gcc.gnu.org/fortran/	Only needed to run Fortran tests
Utilities		
cmake	https://cmake.org	version > 3.12
Third party libraries		
proj	https://proj.org	to handle projections
netcdf	http://www.unidata.ucar.edu/software/netcdf/	for netcdf support needed Please note: You also need to install the legacy C++ interface and HDF5
cairo + pango	https://www.cairographics.org http://www.pango.org/	for png/pdf support needed
expat	http://expat.sourceforge.net/	for XML parsing
ECMWF libraries		
ecCodes	ecCodes	Enables GRIB and BUFR support
odc	odc on GitHub	if ODB support needed

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.

Magics specific CMake options

After changing into the build Magics directory, the user has to run CMake with his/her own options. The command gives feedback on what requirements are fulfilled and what software is still required. Table below gives an overview of the different options of configure. The default (without any options) will compile a share library only and install it in /usr/local/.

cmake options	doc	default
ECCODES_PATH	where to find eccodes (if non-standard installation)	
ENABLE_NETCDF	enable netcdf support	on
NETCDF_PATH	where to find netcdf (if non-standard installation)	
ENABLE_ODB	enable odb support	off
ODB_API_PATH	where to find odb (if non-standard installation)	
ENABLE_FORTRAN	enable fortran interface	on
ENABLE_METVIEW	enable metview support(and Qt support)	off
ENABLE_CAIRO	enable Cairo support	on
PROJ4_PATH	where to find proj4 (if non-standard installation)	

To make sure that a feature is really enabled, you will have to specify with the option ex: *-DENABLE_NETCDF=ON*. In that case CMake will fail if the NetCDF support cannot be enabled.

Testing your build

The Magics code contains a directory called *test* in which, in separate sub-directories, tests for the various interfaces of Magics are provided. Test programs in Fortran and C are compiled and run if **MAGPLUS_HOME=$PWD make check** is invoked from the root directory. (Note that the **MAGPLUS_HOME** needs to be set!)

The output of the tests should verified before the library is installed. This setup does not check if the user setup is correct, but the code in test can be used to do so. More examples of source code can be found on the Magics web gallery .