Met.3D Documentation

Release 1.0.1

Met.3D documentation contributors

Jun 01, 2017
News

• Met.3D version 1.1 has been released! See Release notes for more information.

Met.3D is an open-source visualisation tool for interactive, three-dimensional visualisation of numerical ensemble weather predictions and similar numerical atmospheric model datasets. The tool is implemented in C++ and OpenGL 4 and runs on standard commodity hardware. Its only “special” requirement is an OpenGL 4.3 capable graphics card.

Note: Met.3D is open-source, and available on Gitlab. The software is licensed under the GNU General Public License, Version 3.

Met.3D currently runs under Linux. It has originally been designed for weather forecasting during atmospheric research field campaigns, however, is not restricted to this application. Besides being used as a visualisation tool, Met.3D is intended to serve as a framework to implement and evaluate new 3D and ensemble visualisation techniques for the atmospheric sciences.

Note: A Met.3D reference publication has been published in Geoscientific Model Development and is available online:


Met.3D is developed at the Computer Graphics & Visualization Group, Technische Universität München, Garching, Germany. We hope you find the tool useful for your work, too. Please let us know about your experiences.

The documentation for Met.3D is organised into the following sections:

• User Documentation
• Feature Documentation
• About Met.3D

Information about development is also available:

• Developer Documentation

Attention: The documentation you are reading is work in progress. We are adding bits and pieces whenever we find time. If you don’t find the information you are looking for, please contact us. If you like to contribute to the documentation, please let us know as well!
Release notes

Version 1.1 - April 25, 2017

New features:

• An interactive transfer function editor and functionality for basic direct volume rendering (DVR) has been added.
• Support for textured filled contours in horizontal sections to facilitate stippling and hatching.
• The synchronization control supports restriction of allowed times to available times in selected datasets and provides enhanced animation control.
• Visualizations can be directly saved to a graphics file from a scene view by pressing “s”.
• The trajectory actor can be created at runtime.
• Support for cmake to compile the source code.
• Enhanced support for contour line appearance in 2D sections.
• Support for different camera navigation modes: move camera, rotate scene, 2D top view.
• Many further small usability improvements...

Bugs fixed:

• Lots...
CHAPTER 2

Getting started

This document will show you how to get up and running with Met.3D, and provide you with a short overview of how to use the software. For installation, please follow the notes provided in the sections Met.3D installation with qmake and Met.3D installation with cmake, depending on which build system you prefer. The following sections assume that the tool has been successfully installed.

First steps with Met.3D contains a tutorial for the first steps with the Met.3D. Met.3D actors provides an overview of available visualization modules.

Note: Please note that Met.3D is being developed within a research project. We do our best to fix bugs in the software. However, you may encounter bugs when using Met.3D. Please let us know about any bug you encounter, so we can improve the software. Please also regularly check the open-source repository for software updates and this page for user guide updates.

Note: There is more functionality in Met.3D (in part experimental) than described in this user guide. We will complete the user guide in the future.
CHAPTER 3

Met.3D installation with qmake

This page provides installation guidelines for installing Met.3D on openSUSE systems using the Qt qmake build system. Met.3D requires a number of libraries to be installed and a few external data packages to be downloaded. Most of these packages can be installed via YaST, however, a few have to be downloaded and compiled manually.

**Note:** The installation guidelines have been tested with openSUSE 13.X systems but should be directly applicable to the current openSUSE 42. We have also successfully installed the software on other Linux distributions including Ubuntu (see cmake installation documentation).

System requirements: You need an OpenGL 4.3 capable graphics card and an appropriate Linux driver to run Met.3D. The driver will most likely be a proprietary driver; open-source drivers for Linux currently do not provide the required capabilities. Before you continue with the installation, make sure that graphics card and driver are installed. If everything is installed correctly, the `glxinfo` command should output something similar to (the important thing is the OpenGL version > 4.3):

```
% glxinfo | grep OpenGL
OpenGL vendor string: NVIDIA Corporation
OpenGL renderer string: GeForce GTX TITAN/PCIe/SSE2
OpenGL core profile version string: 4.4.0 NVIDIA 340.96
OpenGL core profile shading language version string: 4.40 NVIDIA via Cg compiler
```

A) Install available packages via YaST (or any rpm manager)

For openSUSE 13.X, these repositories are required (similar for other openSUSE versions):


Packages that need to be installed via YaST (or the system’s repository manager):

- libqt4 and libqt4-devel
Met.3D Documentation, Release 1.0.1

- liblog4cplus and log4cplus-devel
- gdal, libgdal and libgdal-devel
- netcdf, netcdf-devel
- hdf5, libhdf5 and hdf5-devel
- glew and glew-devel
- libfreetype and freetype-devel
- grib_api and grib_api-devel
- libGLU
- gsl and gsl-devel
- ... and we may have forgotten some. Please tell us!

B) Install required libraries from their sources

The libraries in this section are not available as rpm packages and need to be compiled manually.

1) glfx

Get the glfx sources from: https://code.google.com/p/glfx/

```bash
cd glfx
cmake -DCMAKE_INSTALL_PREFIX:PATH=/your/target/dir CMakeLists.txt
make
make install
```

Add `/your/target/dir` to the `met3D_inclib.pri.user` configuration (see below).

2) qcustomplot

Get the qcustomplot sources from: http://www.qcustomplot.com/

You will need the archives `QCustomPlot.tar.gz` and `QCustomPlot-sharedlib.tar.gz`.

Extract the `QCustomPlot.tar.gz` archive (the `/qcustomplot` directory) and put the contents of `QCustomPlot-sharedlib.tar.gz` inside the `/qcustomplot` directory. Go to

```bash
qcustomplot/qcustomplot-sharedlib/sharedlib-compilation
```

and run:

```bash
qmake
make
```

Next, copy the resulting libraries and the `qcustomplot.h` header to directories where Met.3D can find them (i.e. put the path in the `met3D_inclib.pri.user` configuration). These files are required:
3) netcdf-cxx4

Get the sources of the CURRENT (not the old!) NetCDF C++ interface from http://www.unidata.ucar.edu/downloads/netcdf/index.jsp (choose the latest stable distribution). The C++ library requires the regular C library to be installed (see rpm packages above).

```bash
./configure --prefix=/your/target/dir
make
make check
make install
```

Add `/your/target/dir` to the `met3D_inclib.pri.user` configuration (see below).

C) Download source and data packages

We recommend to place the following packages along with the Met.3D sources into a specific directory structure.

Create a base directory `met.3d-base` and a subdirectory `third-party`:

```
met.3d-base/
    third-party/
```

Change into `third-party` to execute the following commands.

1) qtpropertybrowser

Met.3D requires the qtpropertybrowser framework from the “qt-solutions” repository. The qtpropertybrowser sources are directly compiled into the Met.3D executable and hence do not have to be build beforehand. They can be downloaded with git:

```
[ in met.3d-base/third-party]
git clone https://github.com/qtproject/qt-solutions.git
```

2) Fonts

Met.3D requires a TrueType font file. We recommend the “FreeSans” font from the GNU FreeFont package. It can be downloaded from http://ftp.gnu.org/gnu/freefont/. At the time of writing, the most recent version is 20120503:
3) Vector and raster map, coastline and country borderline data

Met.3D requires a base map image in GeoTIFF format, as well as coastline and country borderline vector data in shapefile format. We recommend to use the free data from http://www.naturalearthdata.com. The medium resolution files (50m) work fine (they require roughly 300 MB of disk space).

For coastline data, we use the “Coastline” dataset (http://www.naturalearthdata.com/downloads/50m-physical-vectors/):

```
[in met.3d-base/third-party]
mkdir naturalearth
cd naturalearth
wget http://www.naturalearthdata.com/http//www.naturalearthdata.com/download/50m/physical/ne_50m_coastline.zip
unzip ne_50m_coastline.zip
```

For country boundaries, we use the “Admin 0 – Boundary Lines” dataset (http://www.naturalearthdata.com/downloads/50m-cultural-vectors/):

```
[in met.3d-base/third-party/naturalearth]
wget http://www.naturalearthdata.com/http//www.naturalearthdata.com/download/50m/cultural/ne_50m_admin_0_boundary_lines_land.zip
unzip ne_50m_admin_0_boundary_lines_land.zip
```

For the raster basemap, we use the “Cross Blended Hypso with Shaded Relief and Water” dataset (http://www.naturalearthdata.com/downloads/50m-raster-data/50m-cross-blend-hypso/):

```
[in met.3d-base/third-party/naturalearth]
unzip HYP_50M_SR_W.zip
```

You should now have the following directory structure (... denotes other files):

```
met.3d-base/
    third-party/
        qt-solutions/
            qtpropertybrowser/
                ...
            freefont-20120503/
                FreeSans.ttf
                ...
        naturalearth/
            HYP_50M_SR_W/
                HYP_50M_SR_W.tif
                ...
            ne_50m_coastline.shp
            ne_50m_admin_0_boundary_lines_land.shp
            ...
```

Chapter 3. Met.3D installation with qmake
D) Checkout Met.3D from the GIT repository

The latest version of Met.3D can be checked out from https://gitlab.com/wxmetvis/met.3d/.
Place the repository into the met.3d-base base directory:

```
[in met.3d-base]
git clone -b release_1.1 https://gitlab.com/wxmetvis/met.3d.git
```

E) Configure Met.3D

Modify the software configuration to match your system.

- In met.3d-base/met.3d/, copy the file met3D_inclib.pri.template to met3D_inclib.pri. and modify the contained paths according to your system.
- In met.3d-base/met.3d/config/, copy the files default_frontend.cfg.template and default_pipeline.cfg.template to local configuration files, e.g. to met.3d-base/config/frontend.cfg and met.3d-base/config/pipeline.cfg (these filenames are used in the following but you can choose other names as well). Modify the paths in pipeline.cfg to match the paths at which the forecast data is stored on your system. Modify frontend.cfg to specify the configuration of Met.3D after start-up. If you have followed the suggested directory structure, you should not need to modify frontend.cfg for first experiments with Met.3D. Note that you can use environment variables in the paths defined in the configuration files. In the example installation described in this document, two environment variables (MET3D_HOME and MET3D_BASE) are used (see below).

F) Compile Met.3D

Met.3D uses the Qt qmake build system. You can either build the tool from the command line or from within Qt Creator. From the command line follow the following steps:

In met.3d-base/, create a subdirectory build into which the executable can be built:

```
[in met.3d-base]
mkdir build
```

Change into build/ and run (the “-j 12” option for make starts 12 parallel compilation threads, modify this number to match the number of CPU cores in your system)

```
[in met.3d-base/build]
qmake ../met.3d/met3D.pro
make -j 12
```

Compilation may take a few minutes. If no errors are reported, an executable named met3D should be created.

G) Start Met.3D

Before Met.3D is started, the environment variable MET3D_HOME needs to be set to the Met.3D source directory (alternatively, at least the subdirectories /src/glsl and /config need to be accessible):
export MET3D_HOME=/your/path/to/met.3d-base/met.3d

For the example configuration described in this document, an additional environment variable MET3D_BASE is used in the configuration files to refer to the paths with third-party data (see default_frontend.cfg.template; feel free to change this if you like):

export MET3D_BASE=/your/path/to/met.3d-base

To start Met.3D, type (change the paths to the configuration files if they are stored in a different location):

[e.g. in met.3d-base/build]
./met3D --pipeline=$MET3D_HOME/config/pipeline.cfg --frontend=$MET3D_HOME/config/frontend.cfg

---

Note: Depending on the amount of available forecast data, the software may take a while to start. Also, on first startup the compilation of the OpenGL shader programs may take a while.

---

Note: On first start-up, you should only see an empty window. Please follow the user guide to learn how to create visualizations.
This page provides installation guidelines for installing Met.3D on openSUSE and Ubuntu 16.04 LTS systems using the cmake build system. Met.3D requires a number of libraries to be installed and a few external data packages to be downloaded. Most of these packages can be installed via the respective system package managers (YaST or aptitude), however, a few have to be downloaded and compiled manually.

Note: The installation guidelines have been tested with openSUSE 13.X systems but should be directly applicable to the current openSUSE 42. Under Ubuntu, we have tested with version 16.04 LTS.

System requirements: You need an OpenGL 4.3 capable graphics card and an appropriate Linux driver to run Met.3D. The driver will most likely be a proprietary driver; open-source drivers for Linux currently do not provide the required capabilities. Before you continue with the installation, make sure that graphics card and driver are installed. If everything is installed correctly, the `glxinfo` command should output something similar to (the important thing is the OpenGL version > 4.3):

```bash
$ glxinfo | grep OpenGL
OpenGL vendor string: NVIDIA Corporation
OpenGL renderer string: GeForce GTX TITAN/PCIE/SSE2
OpenGL core profile version string: 4.4.0 NVIDIA 340.96
OpenGL core profile shading language version string: 4.40 NVIDIA via Cg compiler
```

A) Install available packages via your package manager (YaST, aptitude, ...)

For openSUSE 13.X, these repositories are required (similar for other openSUSE versions):


Packages that need to be installed via YaST (or the system’s repository manager):
- libqt4 and libqt4-devel
- liblog4cplus and log4cplus-devel
- gdal, libgdal and libgdal-devel
- netcdf, netcdf-devel
- hdf5, libhdf5 and hdf5-devel
- glew and glew-devel
- libfreetype and freetype-devel
- grib_api and grib_api-devel
- libGLU
- gsl and gsl-devel
- ... and we may have forgotten some. Please tell us!

For Ubuntu 16.04, packages that need to be installed via aptitude:
- qt4-default, qt4-qmlviewer and libqt4-dev
- liblog4cplus-1.1-9 and liblog4cplus-dev
- gdal-bin, libgdal1i and libgdal-dev (deprecated, recommend to download latest version)
- libhdf5-10 and libhdf5-dev (in case it does not support GEOS, install the package manually)
- glew-utils and libglew-dev
- libfreetype6 and libfreetype6-dev
- libgrib-api0, libgrib-api-dev and libgrib-api-tools
- libglu1-mesa and libglu1-mesa-dev
- libgsl2 and libgsl-dev

B) Install required libraries from their sources

Some libraries in this section may not be available as rpm/apt packages and need to be compiled manually.

1) glfx

Get the glfx sources from: https://code.google.com/p/glfx/

```
cd glfx
cmake -DCMAKE_INSTALL_PREFIX:PATH=/your/target/dir CMakeLists.txt
make -j 12
make install
```

To make it easier for cmake to automatically find the libraries, choose one directory from cmake/common_settings.cmake as /your/target/dir. Test
2) qcustomplot

Get the qcustomplot sources from: http://www.qcustomplot.com/

You will need the archives QCustomPlot.tar.gz and QCustomPlot-sharedlib.tar.gz.

Extract the QCustomPlot.tar.gz archive (the /qcustomplot directory) and put the contents of QCustomPlot-sharedlib.tar.gz inside the /qcustomplot directory. Go to

```
qcustomplot/qcustomplot-sharedlib/sharedlib-compilation
```

and run:

```
qmake
make
```

Next, copy the resulting libraries and the qcustomplot.h header to directories where cmake can find them automatically (look at cmake/common_settings.cmake). These files are required:

```
./include:
qcustomplot.h

./lib64:
libqcustomplotd.so -> libqcustomplotd.so.1.2.1*
libqcustomplotd.so.1 -> libqcustomplotd.so.1.2.1*
libqcustomplotd.so.1.2 -> libqcustomplotd.so.1.2.1*
libqcustomplotd.so.1.2.1*
libqcustomplot.so -> libqcustomplot.so.1.2.1*
libqcustomplot.so.1 -> libqcustomplot.so.1.2.1*
libqcustomplot.so.1.2 -> libqcustomplot.so.1.2.1*
libqcustomplot.so.1.2.1*
```

3) netcdf-cxx4

Get the sources of the CURRENT (not the old!) NetCDF C++ interface from http://www.unidata.ucar.edu/downloads/netcdf/index.jsp (choose the latest stable distribution). The C++ library requires the regular C library to be installed (see rpm packages above).

```
./configure --prefix=/your/target/dir
make -j 12
make check
make install
```

To make it easier for cmake to automatically find the libraries, choose one directory from cmake/common_settings.cmake as /your/target/dir.

4) gdal-2.x (only Ubuntu)

Get the sources of the current GDAL version from http://download.osgeo.org/ and compile the source code:

```
./configure --prefix=/your/target/dir
make -j 12
make check
make install
```
To make it easier for cmake to automatically find the libraries, choose one directory from `cmake/common_settings.cmake` as `/your/target/dir`.

### 5) hdf5-1.x (only Ubuntu)

Before compiling HDF5, install the package `libgeos-dev` to enable GEOS support. Get the sources of the current HDF5 version from `https://support.hdfgroup.org/HDF5/` and compile the source code:

```
./configure --prefix=/your/target/dir
make -j12
make check
make install
```

To make it easier for cmake to automatically find the libraries, choose one directory from `cmake/common_settings.cmake` as `/your/target/dir`.

### C) Download source and data packages

We recommend to place the following packages along with the Met.3D sources into a specific directory structure.

Create a base directory `met.3d-base` and a subdirectory `third-party`:

```
met.3d-base/
    third-party/
```

Change into `third-party` to execute the following commands.

#### 1) qtpropertybrowser

Met.3D requires the `qtpropertybrowser` framework from the “qt-solutions” repository. The `qtpropertybrowser` sources are directly compiled into the Met.3D executable and hence do not have to be build beforehand. They can be downloaded with git:

```
[in met.3d-base/third-party]
git clone https://github.com/qtproject/qt-solutions.git
```

#### 2) Fonts

Met.3D requires a TrueType font file. We recommend the “FreeSans” font from the GNU FreeFont package. It can be downloaded from `http://ftp.gnu.org/gnu/freefont/`. At the time of writing, the most recent version is 20120503:

```
[in met.3d-base/third-party]
unzip freefont-ttf-20120503.zip
```

#### 3) Vector and raster map, coastline and country borderline data

Met.3D requires a base map image in GeoTIFF format, as well as coastline and country borderline vector data in shapefile format. we recommend to use the free data from `http://www.naturalearthdata.com`. The medium resolution files (50m) work fine (they require roughly 300 MB of disk space).
For coastline data, we use the “Coastline” dataset (http://www.naturalearthdata.com/downloads/50m-physical-vectors/):

```
[in met.3d-base/third-party]
mkdir naturalearth
cd naturalearth
wget http://www.naturalearthdata.com/http//www.naturalearthdata.com/download/50m/->physical/ne_50m_coastline.zip
unzip ne_50m_coastline.zip
```

For country boundaries, we use the “Admin 0 – Boundary Lines” dataset (http://www.naturalearthdata.com/downloads/50m-cultural-vectors/):

```
[in met.3d-base/third-party/naturalearth]
wget http://www.naturalearthdata.com/http//www.naturalearthdata.com/download/50m/->cultural/ne_50m_admin_0_boundary_lines_land.zip
unzip ne_50m_admin_0_boundary_lines_land.zip
```

For the raster basemap, we use the “Cross Blended Hypso with Shaded Relief and Water” dataset (http://www.naturalearthdata.com/downloads/50m-raster-data/50m-cross-blend-hypso/):

```
[in met.3d-base/third-party/naturalearth]
unzip HYP_50M_SR_W.zip
```

You should now have the following directory structure (... denotes other files):

```
met.3d-base/
    third-party/
      qt-solutions/
        qtpropertybrowser/
          *
            ...
        freefont-20120503/
          FreeSans.ttf
            ...
      naturalearth/
        HYP_50M_SR_W/
          HYP_50M_SR_W.tif
            ...
        ne_50m_coastline.shp
        ne_50m_admin_0_boundary_lines_land.shp
            ...
```

D) Checkout Met.3D from the GIT repository

The latest version of Met.3D can be checked out from https://gitlab.com/wxmetvis/met.3d/.

Place the repository into the met.3d-base base directory:

```
[in met.3d-base]
git clone -b release_1.1 https://gitlab.com/wxmetvis/met.3d.git
```
E) Configure cmake for Met.3D

We provide cmake scripts for Makefile creation and compilation of Met.3D. You can either build Met.3D from the command line, or use a cmake GUI (e.g., cmake-curses-gui, cmake-gui) to configure cmake. We recommend to use a cmake GUI, or, alternatively, start the build process within C++ IDEs like QtCreator, CLion, or Visual Studio Code (these typically provide functionality to open CMakeLists.txt and to run the build process).

From the command line:

First, in met.3d-base/, create a subdirectory build into which the executable can be built and change into this directory:

```
[in met.3d-base]
mkdir build
cd build
```

Create a Makefile by:

```
[in met.3d-base/build]
cmake -DCMAKE_BUILD_TYPE=RELEASE ../met.3d
```

Met.3D can also be built in debug mode; change `-DCMAKE_BUILD_TYPE=RELEASE` to `-DCMAKE_BUILD_TYPE=DEBUG` to achieve this.

If some libraries are not located within the default header/library folders (given in common_settings.cmake), it is likely that you have to manually set the include directory and used libraries for a certain package. For example, if cmake could not find the include directory of GDAL, it will output something like missing GDAL_INCLUDE_DIR. In that case, add `-DGDAL_INCLUDE_DIR=/real/path/to/gdal/includes` to the makefile command and run cmake again. Or use the GUI to set the missing directories and libraries and restart the configuring and generation process.

F) Compile Met.3D

After cmake has created the Makefile, run make (the “-j 12” option for make starts 12 parallel compilation threads, modify this number to match the number of CPU cores in your system).

```
[in met.3d-base/build]
make -j 12
```

Compilation may take a few minutes. If no errors are reported, an executable named met3D should be created in the build directory.

G) Configure Met.3D

Modify the software configuration to match your system.

- In met.3d-base/met.3d/config/, copy the files default_frontend.cfg.template and default_pipeline.cfg.template to local configuration files, e.g. to met.3d-base/config/frontend.cfg and met.3d-base/config/pipeline.cfg (these filenames are used in the following but you can choose other names as well). Modify the paths in pipeline.cfg to match the paths at which the forecast data is stored on your system. Modify frontend.cfg to specify the configuration of Met.3D after start-up. If you have followed the suggested directory structure, you should not need to modify frontend.cfg for first experiments with Met.3D. Note that you can use environment variables in the paths defined
in the configuration files. In the example installation described in this document, two environment variables (MET3D_HOME and MET3D_BASE) are used (see below).

**H) Start Met.3D**

Before Met.3D is started, the environment variable MET3D_HOME needs to be set to the Met.3D source directory (alternatively, at least the subdirectories /src/glsl and /config need to be accessible):

```
export MET3D_HOME=/your/path/to/met.3d-base/met.3d
```

For the example configuration described in this document, an additional environment variable MET3D_BASE is used in the configuration files to refer to the paths with third-party data (see default_frontend.cfg.template; feel free to change this if you like):

```
export MET3D_BASE=/your/path/to/met.3d-base
```

To start Met.3D, type (change the paths to the configuration files if they are stored in a different location):

```
[e.g. in met.3d-base/build]
./Met3D --pipeline=$MET3D_HOME/config/pipeline.cfg --frontend=$MET3D_HOME/config/frontend.cfg
```

**Note:** Depending on the amount of available forecast data, the software may take a while to start. Also, on first startup the compilation of the OpenGL shader programs may take a while.

**Note:** On first start-up, you should only see an empty window. Please follow the user guide to learn how to create visualizations.
CHAPTER 5

First steps with Met.3D

This section introduces the basic functionality of Met.3D. For the sections Starting Met.3D, Adding actors to the scene and Working with scenes and scene views, no forecast data is required.

Starting Met.3D

For the first steps with Met.3D, simply copy the default configuration files default_pipeline.cfg.template and default_frontend.cfg.template that are provided with the Met.3D source code (see installation). In the following, we assume that you have followed the directory structure suggested in installation and that your copies are named pipeline.cfg and frontend.cfg. Let the files be placed in the met.3d-base/met.3d/config/subdirectory. From met.3d-base/met.3d/, start Met.3D by entering:

```
$ ../build/met3D --pipeline=config/pipeline.cfg --frontend=config/frontend.cfg
```

You will see some debug and info output on the console. Fig. 5.1 shows the Met.3D window that appears on first start-up. It contains an empty visualization area.

Fig. 5.1: The (empty) Met.3D main window that appears when first starting the software.
vi-
su-
aliza-
tion
com-
po-
nents
to
the
empty
di-
aliza-
tion
area. In Met.3D, visualization components are called **actors**. Actors represent, for example, a graticule, a base map, a 2D horizontal or vertical cross-section, or 3D isosurfaces. Currently available actors are listed in **Met.3D actors**. Actors are grouped into **scenes**, which are simply a collection of actors. A single actor can be assigned to one or to multiple scenes. A scene can be visualized by one or by multiple **scene views**, which are the visualization areas in the Met.3D window. In Fig. 5.1, one scene view (“view 1”) is visible. It displays “Scene 1”, which currently does not contain any actors. To add actors to this scene, follow these steps:

- Select the menu entry “View/Scene Management” (or press “F4”) to open the scene management dialog displayed in Fig. 5.2. The dialog shows available scenes on the right hand side, and available actors on the left (the listed “Labels” actor corresponds to a special system actor that is always present and that is responsible for text rendering). The buttons below the list of actors provide functionality to create new default actors, to load actor configurations from file, and to delete existing actors.

  - First, create a new (default) graticule actor. Click on the “create” button in the “Actor settings” section. In the dialog that appears, select “Graticule” and name the actor correspondingly. Next, add the graticule actor to “Scene 1”. To do this, select the actor in the list of actors, then check the scenes in which it should appear in the list of scenes below (Fig. 5.3). When you click on “Scene 1”, the graticule will appear in the Met.3D window in the background.

Fig. 5.2: The scene management dialog. Open with “View/Scene Management” or by pressing “F4”.

  - Repeat the steps for a volume bounding box actor and a base map actor. Close the dialog. Your Met.3D window now looks like Fig. 5.4. The base map is black, as no map has been loaded yet. To load a map, the actor needs to be configured. On the left hand side of the Met.3D window, below the navigation controls for time and ensemble
member, the “Scene 1” tab lists the actors that are assigned to the scene. It provides access to the actors’ properties. The properties are organised in a tree structure (in the following called “property tree”) and grouped into different categories, depending on their function.

Fig. 5.3: After the new default graticule actor has been created, it can be assigned to “Scene 1” by selecting the actor and checking the corresponding scene(s).

´ Open the property tree for the base map actor. In the “actor properties” group, find the “load map file” entry. Click on the button to open a file dialog. Choose the raster base map that you have downloaded from Natural Earth (see installation) and confirm. After the file has been loaded, your Met.3D window looks like Fig. 5.5.
Fig. 5.4: The Met.3D main window after graticule, volume bounding box and base map have been added. No base map file has been loaded yet, hence the map appears black.

The configured base map actor can be saved to an actor configuration file, so that the configuration steps do not have to be repeated each time a base map actor is required (this is particularly important if more properties than just the path to a map file have to be set for a given actor). In the base map actor’s “configuration” group, select the “save” entry. For this tutorial, create a new subdirectory met.3d-base/config/ in your met.3d-base/ directory (see installation), and store the configuration to basemap_default.actor.conf. To check whether the configuration has been stored successfully, open the scene management dialog again. Select the base map actor in the list of actors in the “Actor settings” section and click “remove”. Next, click “create from file” and select the file that you have just stored. The actor reappears in the list and can be reassigned to “Scene 1”. Saved configurations can also be loaded automatically during the Met.3D start process. This is useful to let Met.3D recreate your favourite configuration each time it is started. Save the actor configuration for the volume bounding box and the graticule actors as well. We assume that the configurations are saved to volumeboundingbox_default.actor.conf and graticule_default.actor.conf, respectively. Next, close Met.3D and open the frontend.cfg file with a text editor. Find the [Actors] section and modify it according to (the size=3 entry is important):

```
[Actors]
size=3
1\config=$MET3D_BASE/config/volumeboundingbox_default.actor.conf
1\scenes=Scene 1
2\config=$MET3D_BASE/config/graticule_default.actor.conf
2\scenes=Scene 1
3\config=$MET3D_BASE/config/basemap_default.actor.conf
3\scenes=Scene 1
```

The listed configuration instructs Met.3D to create actors according to the specified configuration files on start-up. To check, restart Met.3D. The actors are loaded as previously in Fig. 5.5.

**Working with scenes and scene views**
Met.3D provides standard mouse interaction techniques to change the observer’s point of view within a scene view. Hold the left mouse button and drag to rotate the scene, the right button to pan, and use the scroll wheel to zoom.

**Hint:** The default mouse interaction for navigation can be changed in the [SceneNavigation] section of `frontend.cfg`. See the default file provided with Met.3D for examples.

The number and layout of displayed scene views can be changed by selecting one of the presets in the “View” menu (the preset configurations in the menu are also available through the keyboard short-cuts Alt+0..6). The default maximum number of scene views is four. For example, Fig. 5.6 shows the layout “one large view and three small views” (Alt+5).

Similar to actor configurations, viewpoints (camera positions) can be saved.

**Fig. 5.5:** The same as Fig. 5.4 with a base map file loaded.

**Fig. 5.6:** Met.3D window layout with one large and three small scene views. Views 1, 2 and 3 all show “Scene 1”, but from different viewpoints.

### 5.3. Working with scenes and scene views

...
stored at a later time. To save a given viewpoint, select the “System” tab on the left of the Met.3D window. Similar to actor properties, properties affecting the scene views are arranged in a property tree. Open the property for the scene view for which you would like to save the camera, and choose the “modify camera/save” property.

Multiple scene views can show the same scene. In the example in Fig. 5.6, views 1, 2 and 3 all show “Scene 1”, but from different viewpoints. To achieve this, open the scene management dialog and specify the scenes that the views display in the lower left area of the dialog (cf. Fig. 5.2). Also, in view 3 the vertical scaling of the scene is different to views 1 and 2. The vertical scaling can also be changed in the “System” tab on the left side of the Met.3D window. As an example, modify the “rendering/vertical scaling” parameter for view 1 and observe the difference. With “interaction/sync camera with view”, it is possible to synchronize the camera viewpoints of two scenes. This is useful if two different scenes are to be examined from the same viewpoint.

Actors can be assigned to multiple scenes by selecting the corresponding scenes in the scene management dialog (Fig. 5.3). This way, actors such as graticule, base map or volume bounding box can be shared among different scenes (representing “static” content) and combined with different forecast actors (“dynamic” content) in the individual scenes.

For actors that allow interaction with the user, the scene views provide an “interaction mode”. The interaction mode can be enabled by pressing “i” while a scene view is selected, or by checking the corresponding property in the scene view’s property tree. While the interaction mode is enabled for a scene view, the text “Interaction mode” appears at the bottom of the view and the camera is frozen.

As an example, the “movable poles” actor supports user interaction. It allows the user to move a pole within the scene by dragging a handle attached to a pole. The following steps add the actor to the current configuration:

- In the scene management dialog, create a new instance of the “movable poles” actor and add the instance to “Scene 1”. As no specific pole has been defined yet, there is nothing to see so far.

- In the actor’s property tree, the “actor properties” group contains the entry “add pole”. Click on the button to create a new pole. By default, it is placed at (lon/lat) = (0/0). Enter “Y=45” to manually specify the latitude. The pole is now located in western France. Fig. 5.7 shows the corresponding Met.3D window (note that the “one large, two small views” layout was activated by pressing Alt+4).

- Click on the large scene view (“view 1”) and press “i” to activate the interaction mode. Now, small spheres that act as handles appear at the top and bottom of the pole. Move the mouse pointer over the bottom handle of the pole. The handles are now highlighted in red (Fig. 5.8).
Fig. 5.7: Met.3D window with a movable pole.

- Click on the handle and drag the pole. Note how the pole’s position is updated in all scene views that display the scene.

**Adding forecast data to the scene**

Next, we add a horizontal cross-section that displays some forecast data. The goal is to create a forecast product that shows colour-coded wind speed, overlaid with contour lines of geopotential height and wind barbs.

Fig. 5.8: Met.3D window with a movable pole, in interaction mode.
Note: In the following, we assume that forecast data from the European Centre for Medium Range Weather Forecasts (ECMWF) of wind speed, u-component and v-component of horizontal wind, and of geopotential height is available.

Configuration of the data pipeline

Unlike other visualization tools, Met.3D does not allow the user to select and load a specific data file at runtime. Instead, data pipelines need to be configured before the tool is started. A pipeline provides access to a dataset that can be distributed over several files. All files of a dataset need to be located in a single directory. Here, we assume that a pipeline for ECMWF ensemble forecast data in NetCDF-CF format shall be configured.

- Open the file met.3d-base/met.3d/config/pipeline.cfg and find the section [NWPPipeline].
  Our pipeline is specified by:

```ini
1\name=ECMWF ENS EUR_LL10
1\path=/home/local/data/mss/grid/ecmwf/netcdf_Oct15
1\fileFilter=*ecmwf_ensemble_forecast*EUR_LL10*.nc
1\schedulerID=MultiThread
1\memoryManagerID=NWP
1\fileFormat=CF_NETCDF
1\enableRegridding=true
```

The prefix 1\ specifies the number of the pipeline (here, the size parameter in the [NWPPipeline] needs to be set to at least 1). name specifies the name under which the dataset can be accessed in Met.3D, path specifies the directory in which the forecast data files are located. All files in the specified directory that match the fileFilter specified next are considered to belong to the dataset. schedulerID can be set to MultiThread or SingleThread. With the first choice, data processing tasks will be distributed over the available CPU cores in your system. memoryManagerID needs to correspond to a memory manager instance defined in the [MemoryManager] section of the configuration file (see the template for examples, a memory manager needs to be given a name and amount of CPU memory it can consume). fileFormat can be set to either CF_NETCDF or ECMWF_GRIB. If enableRegridding is set to true, Met.3D’s regridding module will be activated.

- For the given example, simply adjust the path to match that of your system.

Adding a horizontal section

Restart Met.3D. The dataset is now registered within the tool with the data source identification ECMWF ENS EUR_LL10.

- In the scene management dialog, create a “horizontal cross-section”. Add the section to the current scene.

- To map the wind speed data to colour, a transfer function (i.e. a colour map) is required. Create and add a “1D transfer function” and close the scene management dialog. The Met.3D window now looks like Fig. 5.9 (where the default settings of colour bar and section may vary depending on the exact Met.3D version). The horizontal

---

1 Unfortunately, we cannot provide a sample dataset at the present time. If you have access to ECMWF data, please contact me so we can let you know how to obtain a suitable dataset.

2 This is a typical application case when forecast data distributed over multiple files are automatically downloaded by shell scripts from the forecast provider (e.g. ECMWF) as soon as they are available.

3 CF-compliant NetCDF files, e.g. as generated from ECMWF GRIB files with Unidata’s “netcdf-java” library, and files used by the DLR Mission Support System. We are interested in feedback about success or problems with other model data.

4 Only GRIB messages that are interpolated to a regular longitude-latitude grid in the horizontal will be considered. For model level and pressure level fields, all messages that correspond to a single 3D data field need to be put in the same file. Note that a GRIB pipeline needs to index all GRIB files located in the specified directory when they are first read. This may take several minutes. After the index has been created, do not alter the GRIB files.
section by default contains an instance of a graticule actor that replicates the graticule and coast lines at the elevation of the section.

- All actors that display forecast data make use of “variables” that represent a given forecast parameter (e.g. wind speed) and store the actor-related settings that correspond to this parameter. For example, in the case of a horizontal section, these settings include whether the parameter shall be visualized as line contours or as filled contours and, in the latter case, which colour map shall be used. To add a variable to the horizontal section, select “add new variable” in the actor’s “variables” group in the property tree. The dialog that opens is shown in Fig. 5.10. It lists all available forecast parameters that are available from the registered data sources.

Fig. 5.9: An empty horizontal section and a colour bar (1D transfer function) have been added to the scene.

- From the list, select the data source ECMWF ENS EUR_LL10 ENSFilter and the variable Windspeed_hybrid. The suffix “ENSFilter” in the data source name indicates that in the data pipeline, a module that computes statistical quantities from the ensemble (e.g. mean or standard deviation) has been attached to the original dataset. Confirm the forecast parameter selection. In the follow-up dialog about a synchronization control select “Synchronization” to synchronize the variable with the global time and ensemble settings.

- The variable appears in the property tree of the horizontal section actor. In the variable’s subgroup “rendering”, find the properties “transfer function” and “render mode”. Select the transfer function that you have created above and set the render mode to “filled contours”. The scene now looks like the screenshot in Fig. 5.11.
Fig. 5.10: Data source selection dialog. The dialog lists all forecast parameters that are available from the registered data pipelines.

- Next, the colour map needs to be adjusted. Open the property tree of the colour map. In the “actor properties/range” subgroup, set “decimals” to 0 and the minimum and maximum value to 10 and 80, respectively. The colour map type is set to “predefined” (vs. HCL, see 1D transfer function (Colour map)). Open the “predefined” group and select “hot_wind”. Also check the “reverse” field. Now, the horizontal sections displays the wind speed as shown in Fig. 5.12.

Fig. 5.11: A variable containing horizontal wind speed has been added to the horizontal section. The colour bar (1D transfer function) has been connected to the variable.

- Go back to the horizontal section actor’s property tree and add a second variable. This time, choose Geopotential_height_hybrid. For this variable, we set “render mode” to “line contours”. Potential
contour values can be specified in the text fields “thin contour levels” and “thick contour levels”. The strings can either be a list of values (e.g. 5000, 5500, 6000) or three values of format [from, to, step]. An example of the latter is [0, 26000, 40], which will display a contour line every 40 m between 0 and 26000 m. Note that the range of values is chosen to reflect all possible values that might be encountered at any vertical location of the section. Met.3D will recognize which of the values are applicable to a given elevation and only render the actually visible lines. Put [0, 26000, 40] into the “thin contour levels” field and [0, 26000, 200] into the “thick contour levels” field. The result is shown in Fig. 5.13.

![Fig. 5.12: Modification of the colour map to reflect the range of values in the wind speed variable.](image)

- To complete the forecast product, add two more variables to the actor: u-component_of_wind_hybrid and v-component_of_wind_hybrid. Leave the “render mode” for both variables set to “disabled”. Instead, open the “actor properties/wind barbs” property group of the horizontal section actor. At the bottom of the group, assign the u and v-components to the corresponding fields and click on “enabled”. For the screenshot in Fig. 5.14, I have also changed the colour of the barbs to blue.
move the section up and down, and use the time and ensemble navigation buttons in the top left of the Met.3D window to change time and/or ensemble member. Of course, the horizontal section configuration can be saved to a configuration file. Similar to the actors in Adding actors to the scene, a saved horizontal section actor configuration can be loaded at runtime in the scene management dialog. Alternatively, it can be listed in the frontend.cfg file to be loaded during start-up.

You can also have multiple instances of an actor in a scene. Fig. 5.15 shows an example of two horizontal sections stacked on top of each other (the lower one at 925 hPa and the upper one at 200 hPa). The pole is placed in the centre of the low pressure system at 925 hPa, its intersection with the upper section showing the relation of the position of low-level centre to the jet stream.
Fig. 5.15: Two identical horizontal sections stacked on top of each other. The vertical pole illustrates the relation between low pressure centre at 925 hPa and the jet stream at 200 hPa.

5.4. **Adding forecast data to the scene**
This section provides a short overview of the actors currently implemented in Met.3D. Major characteristics of the actors are listed, however, many actors provide more properties than currently documented.

**Note:** For all actors that visualize forecast data please note the following:

- The “variables” (see *Adding forecast data to the scene*) of the actors can be synchronized with the global time and ensemble settings but don’t have to be synchronized (e.g. if different ensemble members shall be compared).
- Availability of ensemble functionality depends on the data pipeline (if connected to an ensemble dataset, the pipeline can provide the data of different ensemble members or compute statistical quantities including ensemble mean and probabilities).

**Graticule**

The graticule actor draws a graticule and coast and border lines into...
The distance and colour of the graticule lines can be customised.

Coast and border lines are read from a shapefile (in this manual data from Natural Earth (see installation) are used; other shapefile datasets can be used as well).

Basemap
name says, the base map actor draws a base map into a scene (Fig. 6.2).

- Map data can be read from an arbitrary geo-referenced GeoTIFF file (in this manual data from Natural Earth (see installation) is used; other GeoTIFF datasets can be used as well).

- The bounding box can be customised, i.e. the map file can be larger than the displayed region.

- Colour saturation of
Volume bounding box

Fig. 6.3: Volume bounding box actor.

to illustrate the region covered by a data volume, or to increase spatial perception for horizontal sections.

- The vertical tick marks of a bounding box are labelled according to pressure.
1D transfer function (Colour map)

The 1D transfer function actor provides a colour map that is (a) displayed in a scene (Fig. 6.4), and that (b) can be used in other actors to map a data value to a colour.

- Data scalar range can be adjusted, as well as number of colour steps, labels, tick marks and position of the colour map in the visualization.
- The actor provides a number of predefined colour maps, as well as Hue-Chroma-Luminance (HCL) colour maps. For the latter, specification of HCL colour maps follows the website of Reto Stauffer.

Horizontal sections

The horizontal section actor draws a map of multiple forecast

Fig. 6.4: 1D transfer function (colour map) actor.

Fig. 6.5: Horizontal cross-section actor, together with a transfer function.
arbitrary pressure altitude (Fig. 6.5).

- The actor supports multiple forecast data fields.
- Each data field can be rendered as line contours, filled contours or pseudo-colour plot (only one filled contour or pseudo-colour plot is allowed per horizontal section).
- The horizontal section can be interactively moved up and down by the user.
- Each horizontal section contains an instance of a graticule actor to render graticule lines and coast and border lines.
- A surface shadow is available to improve spatial perception.
- Multiple horizontal sections are allowed in a single scene (cf. Fig. 5.15).
- Wind barbs are supported.

## Vertical sections

The vertical section actor draws vertical 2D cross-sections of multiple forecast data fields along an arbitrary path (Fig. 6.6).

- The actor supports multiple forecast data fields.
- Each data field can be rendered as line contours or filled contours.
- A vertical section can be drawn along arbitrarily many waypoints.
- The waypoints can be interactively moved by the user (to move the waypoints, enable a scene view’s “interaction mode”, see Working with scenes and scene views).
• Waypoints can be exported in “flight track file” for the DLR Mission Support System (MSS), or read from files exported by the MSS.

• Bottom and top pressure limits of the section can be adjusted.

**Surface topography**

The surface topography actor renders a terrain, using pressure as the vertical coordinate (Fig. 6.7).

- The actor renders the surface pressure field as terrain.
- Two data fields are required: Surface pressure (or a similar pressure surface) and a variable that maps to colour (via a transfer function).

**Volume actor**

The volume actor renders multiple isosurfaces
normal curve functionality (not shown).

- The actor renders multiple isosurfaces of a data field via raycasting.
- An isosurface can be coloured according to another data field (e.g. an isosurface of wind speed can be coloured by temperature or pressure).
- Shadows of the isosurface and normal curves are rendered to the (earth) surface.
- Normal curves are supported.
- Interactive region contribution analysis is supported.

**Movable poles**

The movable poles actor draws an arbitrary number of vertical axes into the scene (Fig. 6.9).

- The actor supports an arbitrary number of vertical axes.
- The axes can be labelled by pressure.
- The axes can be interactively moved by the user.
Trajectory actor

The trajectory actor draws precomputed Lagrangian particle trajectories into a scene, either as 3D tubes (Fig. 6.10) or as particle positions.

Note: The trajectory actor currently cannot be created during runtime. It needs to be specified as a predefined actor in the frontend configuration file (we are working on fixing this issue).

- Supports precomputed Lagrangian particle trajectories that are available in a file format similar to NetCDF-CF. Currently, a converter is available for LAGRANTO trajectories (please contact me).
- Trajectories can be drawn as 3D tubes, coloured by pressure elevation.
- Shadows of the trajectories are rendered to the surface.
- The individual positions of the trajectories can also be drawn, either for the entire trajectories or for individual time steps.
- Animations of particle positions are possible. Also, trajectory tubes can be drawn for the part of the trajectory between trajectory start and current time.
Supported data and file formats

This section provides an overview of the data and file formats supported by Met.3D.

- Met.3D supports gridded, structured data with a regular, georeferenced longitude-latitude grid in the horizontal, and either levels of constant pressure (in the following “pressure levels”) or hybrid sigma-pressure levels (as used, e.g., by the ECMWF integrated forecast system) in the vertical.

- Multiple datasets (that can have different grids) and ensemble datasets are natively supported.

- Also, trajectory data is supported.

- Currently, gridded data can be read from NetCDF files following the Climate and Forecast (CF) Metadata Conventions and from ECMWF GRIB files; trajectory data can be read from NetCDF files with a custom formatting.

Note: If your data cannot be read by Met.3D (i.e., if you don’t have access to your datasets after start-up), a (admittedly currently limited) set of error messages is displayed by the software in the start-up output on the text console. Please contact us if you need help.

Note: Display of non-georeferenced data on regular grids (e.g., output from LES models) is possible but requires some hacking on the NetCDF side. Please contact us if you are interested. We are working on making data import for such data easily possible in a future Met.3D version.

Internal data formats in Met.3D

Gridded, structured data

Met.3D supports gridded, structured data with a regular longitude-latitude grid in the horizontal, and either levels of constant pressure (in the following “pressure levels”) or hybrid sigma-pressure levels (as used, e.g., by the ECMWF integrated forecast system) in the vertical.
Internally, a single 2D or 3D scalar data field (i.e., one forecast variable, one time step, one ensemble member) is treated as the smallest data entity. A single data field is stored in a class derived from `MStructuredGrid`, these objects store all contextual information required to visualise the field (including latitude and longitude coordinates, pressure or hybrid level, time step and additional metadata). Example of derived classes are `MRegularLonLatGrid` for 2D fields, `MRegularLonLatStructuredPressureGrid` for 3D fields on pressure levels, `MLonLatHybridSigmaPressureGrid` for 3D fields on hybrid sigma-pressure levels. Time series and ensemble sets are composed of these basic data field entities.

**Trajectory data**

Trajectories are stored as bundles of trajectories (class `MTrajectories`). A single trajectory in a bundle consists of a simple list of positions in longitude-latitude-pressure space that resemble the trajectory. All trajectories in a bundle share the same time information, i.e. the time for position \( n \) is the same for all trajectories. This data layout corresponds to the smallest data entity being the bundle of all trajectories computed for a single ensemble member for the same time steps.

**Gridded data in NetCDF format**

Gridded, structured data can be read from NetCDF files that follow the Climate and Forecast (CF) Metadata Conventions. Define a dataset consisting of CF-compliant NetCDF files by adding the following entry to your pipeline. `cfg`:

```plaintext
\[
\text{name=YOUR DATASET NAME} \\
\text{path=/your/path/data/netcdf} \\
\text{fileFilter=ecmwf_ensemble_forecast*EUR_LL10*.nc} \\
\text{schedulerID=MultiThread} \\
\text{memoryManagerID=NWP} \\
\text{fileFormat=CF_NETCDF} \\
\text{enableRegridding=true}
\]
```

All files in the specified path that match the given file filter will be used for the dataset. See the file `default_pipeline.cfg.template` for further comments on the meaning of the individual keywords.

**Note:** Forecast variables, time steps and ensemble members can be arbitrarily distributed over the files that match the file filter. You can store all ensemble members in one file but have different files for each time step, or vice versa. Or everything can be stored in a single file.

The following example contains a NetCDF header of a file containing an ensemble forecast on pressure levels:

```plaintext
netcdf somegriddeddata.pl {
dimensions:
  lon = 101 ;
  lat = 41 ;
  isobaric = 12 ;
  time = 1 ;
  ens0 = 51 ;
variables:
  float lat(lat) ;
  lat:units = "degrees_north" ;
  float lon(lon) ;
  lon:units = "degrees_east" ;
  float isobaric(isobaric) ;
```

46 Chapter 7. Supported data and file formats
```csharp
isobaric:units = "hPa";
isobaric:long_name = "Isobaric surface";
isobaric:positive = "down";
int time(time);
time:units = "Hour since 2012-10-15T00:00:00.000Z";
time:standard_name = "time";
int ens0(ens0);
ens0:standard_name = "ensemble_member_id";
float Geopotential_isobaric(time, ens0, isobaric, lat, lon);
Geopotential_isobaric:long_name = "Geopotential @ Isobaric surface";
Geopotential_isobaric:units = "m2.s-2";
float Temperature_isobaric(time, ens0, isobaric, lat, lon);
Temperature_isobaric:long_name = "Temperature @ Isobaric surface";
Temperature_isobaric:units = "K";
...
```

- The **latitude and longitude dimensions** are recognised according to their `units` keyword; cf. the longitude/latitude section in CF-conventions.

- The **vertical pressure level dimension** requires `units` of pressure, as well the `positive` attribute being defined; cf. the vertical coordinate section in CF-conventions.

- Time **time dimension** is identified by its `units` attribute; cf. the time coordinate section in CF-conventions.

**Note:** The time encoded in the `units` attribute of the time dimension is used as the forecast base/initialisation time.

**Note:** The **ensemble dimension** is currently not specified in the CF conventions. Met.3D simply searches for a variable with a `standard_name` attribute set to `ensemble_member_id`. For now, the `CoordinateAxisType` used by the netcdf-java library `_CoordinateAxisType = "Ensemble"`) is also acceptable. (See the Met.3D source code method `NcCFVar::getEnsembleVar()` in `nccfvar.cpp`).

The following example contains a NetCDF header of a file containing an ensemble forecast on hybrid sigma-pressure levels; the required keyword of the vertical variable are different; cf. Appendix D of the CF-conventions.

```csharp
netcdf somegriddeddata.ml {
dimensions:
  lon = 101;
  lat = 41;
  hybrid = 62;
  time = 1;
  ens0 = 51;
variables:
  float lat(lat);
  lat:units = "degrees_north";
  float lon(lon);
  lon:units = "degrees_east";
  float hybrid(hybrid);
  hybrid:units = "sigma";
  hybrid:long_name = "Hybrid level";
  hybrid:positive = "down";
  hybrid:standard_name = "atmosphere_hybrid_sigma_pressure_coordinate";
  hybrid:formula = "\( p(time, level, lat, lon) = a(level) + b(level) \cdot ps(time, lat, lon) \)";
...}
```
hybrid:formula_terms = "ap: hyam b: hybm ps: Surface_pressure_surface" ;
int time(time) ;
time:units = "Hour since 2012-10-15T12:00:00.000Z" ;
time:standard_name = "time" ;
int ens0(ens0) ;
ens0::CoordinateAxisType = "Ensemble" ;
double hyam(hybrid) ;
hyam:long_name = "hybrid A coefficient at layer midpoints" ;
hyam:units = "Pa" ;
double hybm(hybrid) ;
hybm:long_name = "hybrid B coefficient at layer midpoints" ;
hybm:units = "1" ;
float Temperature_hybrid(time, ens0, hybrid, lat, lon) ;
Temperature_hybrid:long_name = "Temperature @ Hybrid level" ;
Temperature_hybrid:units = "K" ;
float Specific_humidity_hybrid(time, ens0, hybrid, lat, lon) ;
Specific_humidity_hybrid:long_name = "Specific humidity @ Hybrid level" ;
Specific_humidity_hybrid:units = "kg/kg" ;
...
}

Gridded data in GRIB format

Met.3D provides support for GRIB files written by ECMWF’s grib_api and ecCodes libraries (as output, e.g. by the ECMWF MARS archive or written by Metview).

In your pipeline.cfg file, simply change the fileFormat entry to ECMWF_GRIB:

```bash
name=YOUR DATASET NAME
path=/your/path/data/grib
fileFilter=*ecmwf_ensemble_forecast*EUR_LL10*.grb
schedulerID=MultiThread
memoryManagerID=NWP
fileFormat=ECMWF_GRIB
enableRegridding=true
```

- GRIB messages may be arbitrarily distributed over different files that match the specified fileFilter.
- As for NetCDF files, support is only provided for a horizontally regular longitude/latitude grid (GRIB gridType = regular_ll) and either pressure (GRIB typeOfLevel = isobaricInhPa) or hybrid sigma-pressure levels (GRIB typeOfLevel = hybrid) in the vertical.
- Hybrid sigma-pressure model levels additionally require the surface pressure field to be present in the dataset (GRIB shortName = sp).
- Analysis (GRIB dataType = an), forecast (GRIB dataType = fc) and perturbed/control (i.e., ensemble forecast; GRIB dataType = pf/cf) messages are interpreted.
- For 3D fields, a consistent consecutive list of vertical levels must be present for all time steps and ensemble members of a dataset. Levels need not be complete (i.e., there can be missing levels at the top or bottom) but the same levels need to be provided for all data fields in a dataset.
- For a given dataset, all GRIB messages must have the same horizontal extent.
Trajectory data in NetCDF format

Since the CF-conventions only provide limited support for trajectory data (in particular no ensemble dimension), we’re currently using a custom NetCDF layout that follows the CF-versions. There are a few limitations:

- There is only one time dimension per NetCDF file, which corresponds to the time of the particle positions along the trajectories. If you have a series of trajectory bundles started at different times (i.e., a bundle of trajectories for each forecast time step as used for WCB detection), you need one NetCDF file per trajectory bundle.

The following example contains a NetCDF header of a file containing an ensemble forecast on pressure levels:

```netcdf
netcdf sometrajectorydata {

dimensions:

time = 17 ;
trajectory = 215332 ;
ensemble = 51 ;
start_lon = 101 ;
start_lat = 41 ;
start_isobaric = 52 ;
time_interval = 8 ;

variables:

double time(time) ;
time:standard_name = "time" ;
time:long_name = "time" ;
time:units = "hours since 2012-10-19 06:00:00" ;
time:trajectory_starttime = "2012-10-19 06:00:00" ;
time:forecast_inittime = "2012-10-17 00:00:00" ;

float lon(ensemble, trajectory, time) ;
lon:standard_name = "longitude" ;
lon:long_name = "longitude" ;
lon:units = "degrees_east" ;

float lat(ensemble, trajectory, time) ;
lat:standard_name = "latitude" ;
lat:long_name = "latitude" ;
lat:units = "degrees_north" ;

float pressure(ensemble, trajectory, time) ;
pressure:standard_name = "air_pressure" ;
pressure:long_name = "pressure" ;
pressure:units = "hPa" ;
pressure:positive = "down" ;
pressure:axis = "Z" ;

float start_lon(start_lon) ;
start_lon:long_name = "longitude of start grid" ;
start_lon:units = "degrees_east" ;

float start_lat(start_lat) ;
start_lat:long_name = "latitude of start grid" ;
start_lat:units = "degrees_north" ;

float start_isobaric(start_isobaric) ;
start_isobaric:long_name = "Isobaric surface of start grid" ;
start_isobaric:units = "hPa" ;
start_isobaric:positive = "down" ;
start_isobaric:axistype = "pressure levels" ;

float time_interval(time_interval) ;
time_interval:long_name = "time interval" ;
time_interval:units = "hours" ;

float delta_pressure_per_time_interval(ensemble, trajectory, time_interval) ;
delta_pressure_per_time_interval:long_name = "max. delta pressure of trajectory in time interval around start time" ;
}
```
• In addition to the air parcel time, Met.3D requires the two time attributes `trajectory_starttime` and `forecast_inititme` that define the time at which the trajectory was started, as well as the forecast initialisation/base time of the forecast data on which the trajectory was computed.

• The `start_lon`, `start_lat` and `start_isobaric` variables are optional and define the grid from which the trajectory bundle was started (used for WCB detection).

• Similarly, the `time_interval` and `delta_pressure_per_time_interval` variables are optional and define pre-computed values for WCB detection. Contact us if you’re interested.
About Met.3D

Met.3D is being developed as a research effort to improve the visualization of meteorological data in research and forecasting.

Detailed information about the software and its uses can be found in the following publications:


Note: Currently, Met.3D research and development is continued, among others, in the context of the German Transregional Collaborative Research Center “Waves to Weather” (see the Waves to Weather cross-cutting activity “Visualisation”). During fall 2016, Met.3D will be used for weather forecasting during the NAWDEX field campaign.
The open-source version of Met.3D is developed by

- Marc Rautenhaus
- Bianca Tost
- Michael Kern
- Alexander Kumpf
- Christoph Heidelmann
- Fabian Schöttl
Large parts of the Met.3D source code have already been documented with the Doxygen code documentation system. A Doxygen configuration file is available in the Met.3D repository (in the doc/doxygen subdirectory). Please run Doxygen locally to build the corresponding documentation.

**Contribution**

If you are contributing to the Met.3D code base, please carefully read the *Met.3D contribution guidelines*. They contain information on the used GIT workflow and coding conventions. As the project is continuously growing, please adhere to the listed conventions.

**Architecture documentation**

Conceptual descriptions of the architecture of selected features in Met.3D are provided in *Architecture*. This section will be expanded in the future.
Met.3D contribution guidelines

This section describes basic guidelines for contributing to the Met.3D project. *GIT workflow* describes the basic workflow for fixing bugs or adding new features to the code, and *C++ coding style* and *GLSL coding style* describe the project’s code conventions.

**GIT workflow**

The Met.3D main repository is hosted on GitLab, with a mirror maintained on Bitbucket. We use the issue tracker on GitLab. We welcome bug and issue reports from users. If you want to report an issue, please create an account on GitLab and contact us with your account. We will add you to the project as an issue reporter.

There are many different GIT workflows out there, as described in this GitLab overview article. To contribute code to Met.3D, please fork the main repository, change the code in a feature branch in your own fork, and create a merge request to feed your work back into the main repository. If you are working on a specific issue, don’t forget to assign the issue to yourself in the issue tracker. Read GitLab’s Project forking workflow for more detailed instructions.

Also read and follow the GitLab documentation for merge requests, issues, etc. when working with the code. Our small difference is that your feature branches live in your forked repository instead of the main repository.

**GIT recipes**

This section contains short recipes for common git tasks.

- **Synchronize between different repositories**
  - To update the master branch of your repository from the public main repository, execute in your local clone:

    * `git remote add public_main https://gitlab.com/wxmetvis/met.3d.git`
      (only once, check with `git remote -v` if you have already added the remote repository)
    * `git checkout master`
    * `git fetch public_main master`
C++ coding style

Met.3D’s C++ coding style is based on the Qt Coding Style, which we have modified in some parts.

Note: Reference: Some of the following text and examples have been taken and adapted from the Qt Coding Style. Please see the referenced document for the original style.

Note: You may encounter lots of code in Met.3D that breaks the following conventions. This is mostly old code, which will be reformatted with time. For new code, please use the following conventions.

General file structure

Both header (*.h) and source (*.cpp) files start with a common header that indicates license and copyright information of the file:

```c++
//****************************************************************************
** This file is part of Met.3D -- a research environment for the
** three-dimensional visual exploration of numerical ensemble weather
** prediction data.
**
** Copyright 2016 ...

**
** Met.3D is free software: you can redistribute it and/or modify
** it under the terms of the GNU General Public License as published by
** the Free Software Foundation, either version 3 of the License, or
** (at your option) any later version.
**
** Met.3D is distributed in the hope that it will be useful,
** but WITHOUT ANY WARRANTY; without even the implied warranty of
** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
** GNU General Public License for more details.

** You should have received a copy of the GNU General Public License
```
After the header, C++ header files follow the structure:

```cpp
#ifndef MYMET3DCLASS_H
#define MYMET3DCLASS_H

// standard library imports
#include <memory>

// related third party imports
#include <GL/glew.h>

// local application imports
#include "gxfw/mactor.h"

namespace Met3D
{

/**
 * @brief The class MMyMet3DClass is briefly described in this Doxygen
 * comment.
 */

class MMyMet3DClass
{

public:

MMyMet3DClass();
~MMyMet3DClass();

/**
 * This comment documents the following function in Doxygen style. All
 * methods should be briefly described in the C++ header file. Do not
 * repeat the description in the source file.
 */

void setMyProperty(int someNumber);

protected:

// protected method and variable definitions ...

int myProperty;

private:

// private method and variable definitions ...

};

} // namespace Met3D

#endif // MYMET3DCLASS_H
```

After the header, C++ source files follow the structure:

```cpp
#include "mymet3dclass.h"

// standard library imports
#include <iostream>
```

11.2. C++ coding style 59
// related third party imports
#include <log4cplus/loggingmacros.h>

// local application imports
#include "util/mutil.h"

using namespace std;

namespace Met3D
{

//**************************************************************************
*** CONSTRUCTOR / DESTRUCTOR ***
**************************************************************************

MMyMet3DClass::MMyMet3DClass()
    : myProperty(42)
{
    // ... do something ...
}

MMyMet3DClass::~MMyMet3DClass()
{
    // ... do something ...
}

//**************************************************************************
*** PUBLIC METHODS ***
**************************************************************************

void MMyMet3DClass::setMyProperty(int someNumber)
{
    myProperty = someNumber;
}

//**************************************************************************
*** PROTECTED METHODS ***
**************************************************************************

// ... some definitions ...

//**************************************************************************
*** PRIVATE METHODS ***
**************************************************************************

// ... some definitions ...

} // namespace Met3D

Chapter 11. Met.3D contribution guidelines
**Indentation**

- Use four (4) spaces for indentation.
- Do not use tabs.

**Blank lines**

- Two (2) blank lines follow each method definition in the source files.
- Use a blank line to separate comments that describe not only the next code line but the next section of code.

**Variables declaration and naming**

We use the same variable declaration style as the Qt Coding Style:

- Each variable is declared on a separate line.
- Avoid short or meaningless name (e.g. r, rwqrr, laksjdiuqk).
- Only use single character names (e.g. i) for counters and temporaries, where the meaning is obvious.
- For pointers or references, always use a single space between the type and * or &, but no space between the * or & and the variable name.

```cpp
// Wrong:
int a, b;
MTask *c, *d;
MTask * t1;

// Correct:
int height;
int width;
MTask *nameOfThisTask;
MTask *nameOfThatTask;
MTask &myFancyTaskReference;
```

- Wait to declare a variable until it is needed.
- Use camel-case: Variables and functions start with a lower-case letter. Each consecutive word in a variable’s name starts with an upper-case letter.
- Avoid abbreviations.

```cpp
// Wrong:
short Cntr;
char ITEM_DELIM = ' ';

// Correct:
short counter;
char itemDelimiter = ' ';
```

- Classes always start with an upper-case letter.
- Public classes start with an ‘M’ (e.g. MTask) followed by an upper case letter.
- Acronyms are camel-cased (e.g. MHsvColourBar, not MHSVColourBar).
Line breaks

- Keep all code and comment lines shorter than 80 columns. (Hint: Most IDEs provide an option to highlight the 80-character-border.)
- Place commas at the end of a wrapped line, operators at the beginning of a new line.

```java
// Wrong:
if (longExpression +
    otherLongExpression +
    otherOtherLongExpression)
{
    // .. some code ..
}

// Correct:
if (longExpression
    + otherLongExpression
    + otherOtherLongExpression)
{
    // .. some code ..
}
```

Braces

- Use separate lines for braces.
- Also use braces if a code block only contains a single line.

```java
// Wrong:
if (condition) {
    // .. some code..
} else {
    // .. some code..
}

// Correct:
if (condition)
{
    // .. some code..
} else
{
    // .. some code..
}
```

Parentheses

- Group expressions by using parentheses.

```java
// Wrong:
if (a && b || c)

// Correct:
if ((a && b) || c)
```
Whitespace

- Always use a single white space after a keyword.

```cpp
// Wrong:
if(condition)

// Correct:
if(condition)
```

- Use a single white space after a comma.
- Use a single white space before and after a mathematical or logical operator.

```cpp
// Wrong:
myFunction(1,2,3);
a = 2+3;

// Correct:
myFunction(1, 2, 3);
a = 2 + 3;
```

- You may use additional white space where it enhances readability of the code, but please use sparingly.

```cpp
MStructuredGrid *mean = nullptr;
MStructuredGrid *stddev = nullptr;
```

Switch statements

- Case labels are in the same column as the switch statement.
- Every case must have a break statement at the end. If the break is avoided intentionally, indicate so by a comment – unless the next case follows immediately.

```cpp
switch (myEnum)
{
    case Value1:
        doSomething();
        break;
    case Value2:
    case Value3:
        doSomethingElse();
        // fall through
    default:
        defaultHandling();
        break;
}
```
Type casting

- Avoid C-style casts where possible.

```c
// Wrong:
char* blockOfMemory = (char*) malloc(data.size());

// Correct:
char* blockOfMemory = reinterpret_cast<char*>(malloc(data.size()));
```

Qt specifics

Met.3D heavily builds on Qt.
- Prefer Qt types to standard library types wherever possible (e.g. use QVector instead of std::vector).
- Use the Qt foreach statement to iterate over container elements.

```c
QList<MTask*> taskQueue;
foreach(MTask *task, taskQueue)
{
    doSomethingWith(task);
}
```

- To avoid compiler warnings for unused parameters in empty (e.g. virtual) functions, use the Q_UNUSED macro.

```c
virtual void onOtherActorCreated(MActor *actor) { Q_UNUSED(actor); }
```

Logging and console output

Met.3D uses the log4cplus library for logging and console output.
- Do not use std::cout to print output, use the log4cplus functions instead.
- Different functions are available for debug, error, info output (and others).
- Use the globally defined log object mlog for logging output.

```c
// Wrong:
std::cout << "Some debug message." << endl;

// Correct:
LOG4CPLUS_DEBUG(mlog, "Some debug message.");
```

Comments

- Write enough comments so that your code can be easily understood by a person who reads the code for the first time (consider being this person and check if you would understand what your code does).
- Doxygen comments used to document methods in the header files use the format shown in the template header above.
- Every function of a class (unless very obvious) should be commented in the corresponding header file.
- Inline comments are placed in the code by using the // indicator. Do not use /* ... */.
• Write comments as complete English sentences, starting with a capital letter and ending with a period.

• Comment should add information, not state the obvious.

```cpp
// Wrong:
// next I call my function
filterUserComments(comments);

// Correct:
// Users may have left weird comments, so get rid of those.
filterUserComments(comments);
```

• Very short comments can be placed at the end of a code line. These do not have to be complete sentences.

```cpp
k = 0; // number of ensemble members
```

### Compiler warnings

• Write your code such that as few compiler warnings as possible appear. Ideally, your code should have no warnings at all.

### GLSL coding style

Where applicable, also use the above C++ style for coding GLSL shader programs.

### General file structure

Met.3D uses the glfx framework. glfx unifies vertex, geometry and fragment shader programs in a single source file, a `.glsl` file. Use the same license and copyright header as for the C++ files. Following the header, the general structure of the `.glsl` files in Met.3D is:

```cpp
//****************************************************************************
*** CONSTANTS ***
*****************************************************************************/

// Use the `const` datatype instead of `#define`.
const float MISSING_VALUE = -999.0E9;

//****************************************************************************
*** INTERFACES ***
*****************************************************************************/

// Interfaces connect different shader stages.
interface GStoFS
{
    smooth vec4 colour;
};

//****************************************************************************
```
// Definition of uniform variables common to all shader stages.
uniform mat4 mvpMatrix;

// Include files.
#include "filename.glsl"

shader VSmain(in vec2 worldXY, out vec3 worldPos)
{
  // .. some code ..
}

shader GSmain(in vec3 worldPos[], out GStoFS output)
{
  // .. some code ..
}

shader FSmain(in GStoFS input, out vec4 fragColor)
{
  // .. some code ..
}

// You can define multiple shader programs using the following syntax:
// vs(number_of_gl_version)=functionName();
// gs(number_of_gl_version)=functionName() : in(geom_in), out(geom_out, max_vertices=max, stream=num)
// fs(number_of_gl_version)=functionName();
program SomeFancyOpenGlashader
{
  vs(420)=VSmain();
  gs(420)=GSmain() : in(points), out(line_strip, max_vertices = 4);
Comments

- Since there are no header files in GLSL, please put Doxygen-style comments for a function above the function definition.
Architecture overview

t.b.d.

Data analysis framework

A number of abstract classes are available to implement user triggered data analysis tasks. Such analysis computations are triggered by an actor with which the user has interacted. For example, the actor can allow the user to select a rendered isosurface and trigger some sort of computation that analyses the selected isosurface and displays the result in an additional graphic.

Implementation

The framework for such analysis tasks is implemented in abstractanalysis.h and consists of three abstract classes from which an actual implementation must derive:

- MAnalysisControl is the “broker” between the actor that triggers the analysis and an MAnalysisDataSource that implements the actual analysis algorithm. The actor instructs an analysis control to run the analysis, afterwards the control will display the results of the analysis in a separate widget.

- MAnalysisDataSource is a specialised MScheduledDataSource that implements the data analysis algorithm. The result is stored in an MAnalysisResult.

- MAnalysisResult is a specialised MAbstractDataItem. It stores the result of the analysis (which can be of any data type required by the analysis) and can be managed by a memory manager instance.

For implementing a new data analysis algorithm, you must inherit from all three classes.

- First, derive from MAnalysisResult and add member variables that store the data you need to store in your analysis result. Make sure to overwrite the getMemorySize_kb() method to ensure correct behaviour of
the memory manager. The abstract class already contains a member `textResult` that you can use to store a user-readable version of your analysis result as text.

- Next, create a (draft) version of your analysis data source by deriving from `MAnalysisDataSource`. As for any data source, you need to implement the methods `produceData()` and `createTaskGraph()`. Special to the analysis data sources is that the request received by these two methods is prepared by your implementation of `MAnalysisControl::prepareRequest()` (see below). The analysis control “talks” to the actor and creates the request that corresponds to the actor’s configuration and user input. The resulting request is then processed by the data source.

- Your analysis control class derived from `MAnalysisControl` needs to implement a number of methods as well.
  - The constructor needs to create a display widget (you need to implement one!) and call the super class methods `setDisplayWidget()` and `setDisplayTitle()`.
  - `createAnalysisSource()` simply needs to return a new instance of your class.
  - `updateAnalysisSourceInputs()` accesses the connected actor’s data sources and creates links to those data sources that are also required by the data source. This method is called by the super class method `run()` each time the actor triggers an analysis.
  - `prepareRequest()` can also access all of the actor’s configuration information (in particular the actor’s NWP variables) and needs to assemble a request that is passed to your analysis data source.
  - `displayResult()` needs to be able to take an analysis result (i.e. the object that you have derived from `MAnalysisResult`) and display its contents in the display widget you have created. This could be as simple as printing a line of text and could be as involved as creating a complex figure.

- The actor finally needs to call the method `MAnalysisControl::run()`. This will cause the control to talk to the actor, prepare the request, run the analysis and display the result.

### Examples

- `MValueExtractionAnalysis` takes a position in 3D space from an actor (e.g., the user clicks on an actor and the corresponding position is determined) and interpolates the values of all data fields (NWP variables) registered with the actor to this position. The result is output as text.

- `MRegionContributionAnalysis` identifies an isosurface of a probability field selected by the user in the raycaster actor and performs an ensemble analysis to determine which members have contributed to the selected probability region.