Gridtools library

Release 0.3.2+a36684e

MOM6 Community

Nov 22, 2021

CONTENTS:

CHAPTER

ONE

ABOUT THIS DOCUMENTATION

1.1 User guides

This readthedocs site hosts information on the *gridtools* user documentation for working with MOM6 ocean model grids. The MOM6 user guide provides a high-level overview of the model as well as the API reference (documentation of source code).

1.2 Download, compile and run

The *gridtools* python library installation tutorial provides a step by step process.

Additional details on the conda package management tool and pip installation process for the gridtools library can be found here.

1.3 Optional packages

If the packages below are not installed, the gridtools library capabilities will become limited to classes and functions that do not require the optional packages.

FRE-NCtools: Tools for manipulating and creating netCDF inputs for FMS managed models. [GeophysicalFDL-GFDLMSGroup21]

xesmf: Universal Regridder for Geospatial Data.

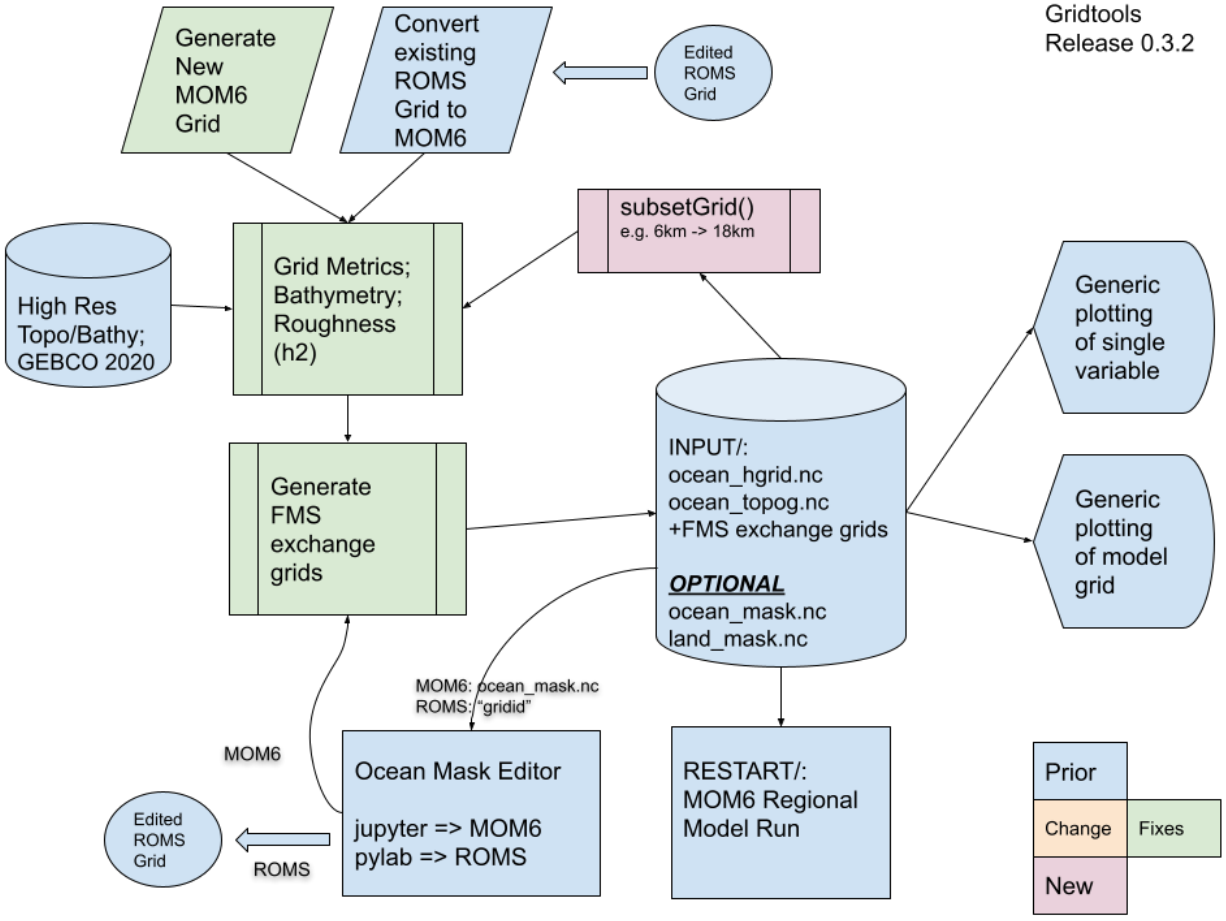
xgcm: Python package for analyzing general circulation model output data.

CHAPTER

TWO

TUTORIALS

The tutorials in this section explore some of the operational pathways provided by the gridtools library.



2.1 Build and Edit a MOM6 Grid in Jupyter

This tutorial works through the following operational elements using the grid generation application and library functions:

- *Build an initial grid*
- *Obtain roughness and topography grids*
- *Write FMS coupler and mosaic files*
- *Examine the topography grid*
- *Use the editor to make some mask updates*
- *Apply mask changes to the model grid*
- *Write updated FMS coupler and mosaic files*
- *Check final ocean mask*

2.1.1 Prerequisites

This tutorial assumes the *gridtools* software has been installed. A local installation tutorial is available.

It is also assumes that a jupyter lab session is also available.

This tutorial does not cover any details on the controls in the grid generation application. Please see "*Grid Generation*" for details on the controls in the application.

An existing notebook with the complete code can be found in the examples directory of the github repository. The name of the example is `NewGridMOM6.ipynb`.

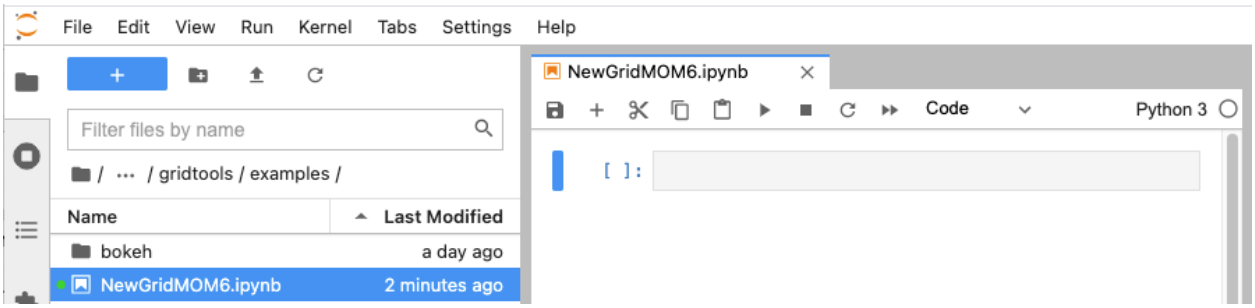
This tutorial assumes that code is being systematically added cell by cell. Subsequent sections of code need to be set to `True` to allow those code blocks to run.

The *GEBCO 2020* topographic dataset is needed for this example. Be sure the `GEBCO_2020.nc` is available for use. The source of this dataset is available here.

This example will run on the binder website. Please follow these "*instructions*" to download the `GEBCO_2020.nc` dataset for use with binder.

2.1.2 Getting Started

Start a **NEW** notebook and rename it to something useful.



2.1.3 Build an initial grid

The first step is to load the *gridtools* library. The second step is creating a *gridtools* object and then launching the grid generation application.

Starting the grid generation application

Create three notebook cells. Each cell will have its own blocks of code. When each cell is run, a number will appear in the brackets. It is **not** important that the numbers match between your notebook and the tutorial.

[1]:

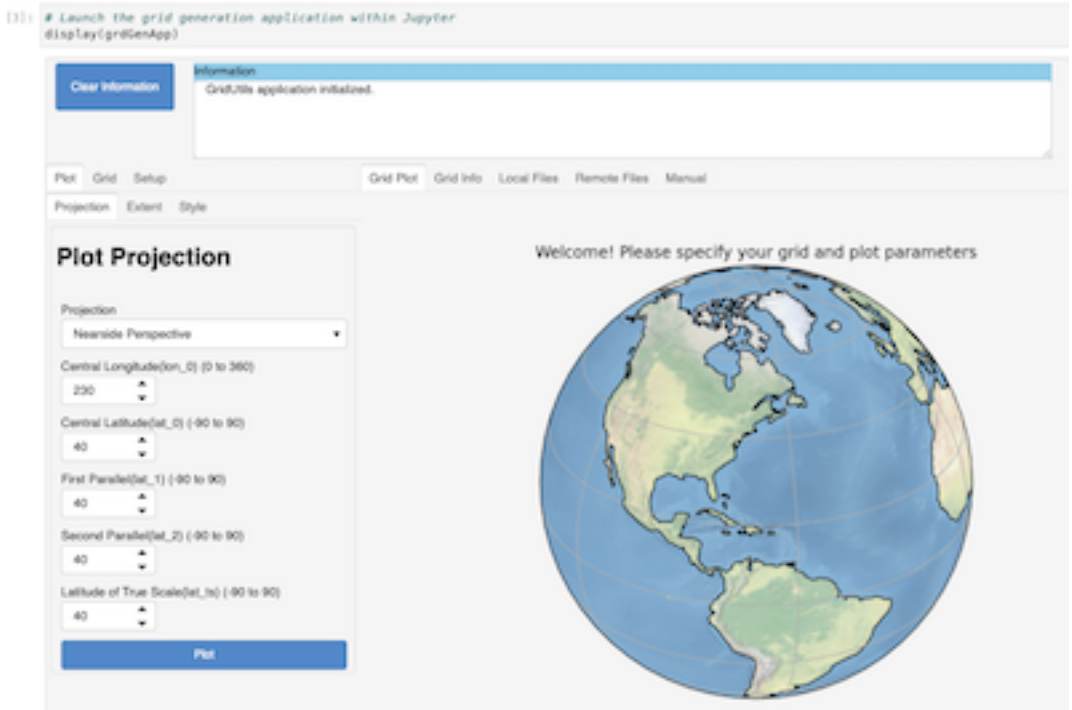
```
# Load some basic python libraries
import os, cartopy
# This loads the GridUtils portion of the gridtools library
from gridtools.gridutils import GridUtils
```

[2]:

```
# Create a GridUtils object
grd = GridUtils()

# Create a grid generation application object
grdGenApp = grd.app()
```

[3]:

```
# Launch the grid generation application within Jupyter
display(grdGenApp)
```



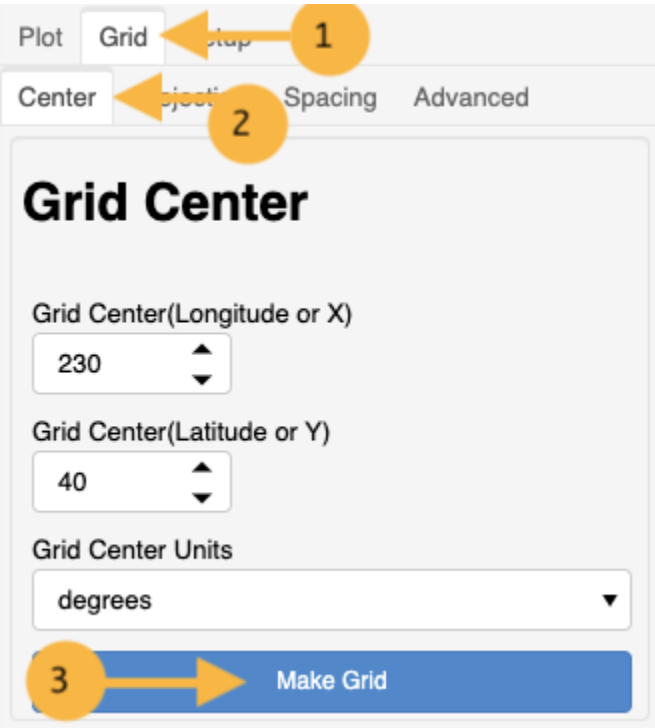
While working with the application, all grid information is stored internally with the `grd` python object created above

in cell #2. Once work is completed with the application, the `grd` object will be used to plot and further manipulate the model grid.

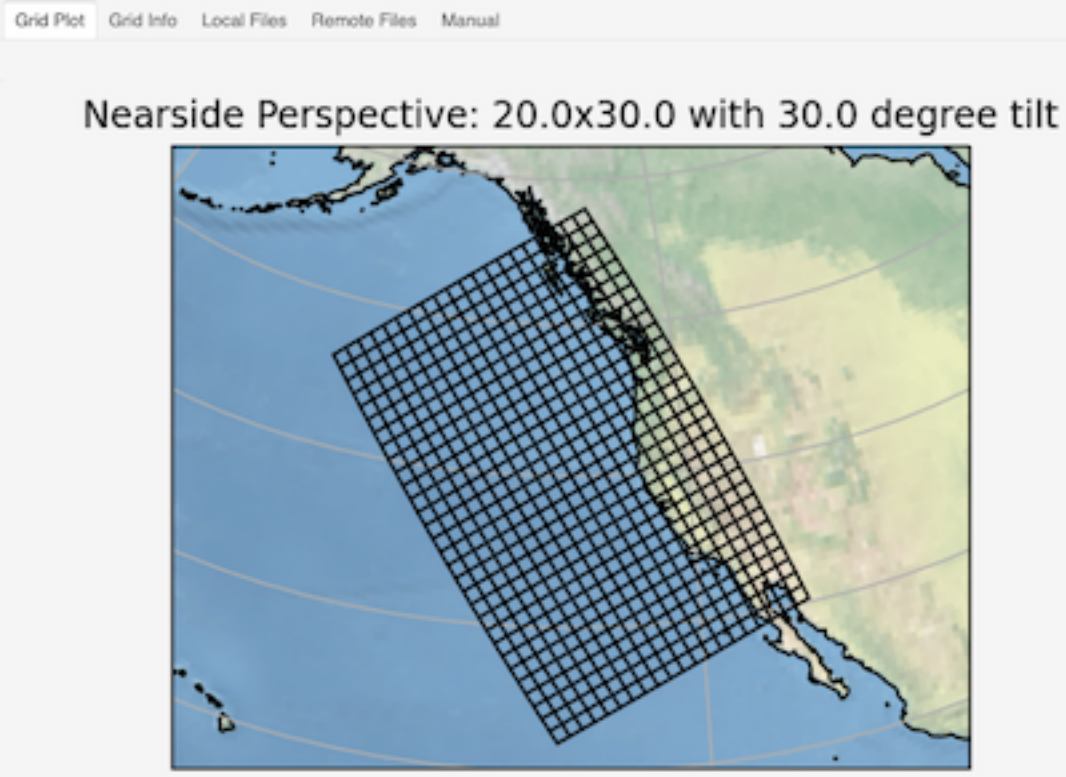
Using the default start up settings of the grid generation application will generate a 20x30 ocean model grid in the **Lambert Conformal Conic** projection centered at **40 degrees North** and **230 degrees West**.

For additional details about the operation of the grid generator, such as adjusting plot, grid parameters and other parameters, please see "*Grid Generation*".

Please click on "*Grid*" next to the "*Plot*" tab. In the "*Center*", tab, please click on "*Make Grid*".



The area below "*Grid Plot*" should update and display a 20x30 ocean model grid.



The new grid is stored with the `grd` object and can be used to generate roughness and topography grids.

2.1.4 Obtain roughness and topography grids

The location of the *GEBCO 2020* file needs to be set appropriately.

[4]:

```
# Detach logger from application
grd.detachLoggingFromApplication()

# Source of GEBCO 2020 topographic grid
highResTopographyFile = "/import/AKWATERS/jrcermakiii/bathy/gebco/GEBCO_2020.nc"

if os.path.isfile(highResTopographyFile):
    topoGrids = grd.computeBathymetricRoughness(highResTopographyFile,
        depthName='elevation',
        maxMb=99, superGrid=False, useClipping=False,
        auxVariables=['depth'])
```

The routine `computeBathymetricRoughness` is called with the location of the *GEBCO 2020* topography. This routine normally only returns a roughness calculation (`h2`). As seen above, a request was made for the `depth` grid. Since *GEBCO 2020* topographic grid is an **elevation** we have to turn the `depth` grid into a **depth** by taking the negative of the grid.

[5]:

```
# Turn the diagnosed topography grid into an actual depth
topoGrids['depth'] = -(topoGrids['depth'])
```

2.1.5 Write FMS coupler and mosaic files

Let us write the FMS coupler and mosaic files for the current model grid, roughness and topography. Edit the `wrkDir` variable so it points to an empty directory. A subdirectory called `INPUT` will also need to be created.

In a later step, the model grid is rewritten. This can be to the existing `INPUT` directory or another directory `INPUT2` to allow comparison.

[6]:

```
# Write current model grid files
wrkDir = "/home/cermak/workdir/configs/zOutput"
inputDir = os.path.join(wrkDir, "INPUT")
input2Dir = os.path.join(wrkDir, "INPUT2")

# Write FMS coupler and mosaic files
grd.makeSoloMosaic(
    topographyGrid=topoGrids['depth'],
    writeLandmask=True,
    writeOceanmask=True,
    inputDirectory=inputDir,
    overwrite=True
)

# Write topographic variable
topoGrids.to_netcdf(os.path.join(inputDir, 'ocean_topog.nc'),
        encoding=grd.removeFillValueAttributes(data=topoGrids))

# Write the model grid
grd.saveGrid(filename=os.path.join(inputDir, "ocean_hgrid.nc"))
```

Note: By default, `makeSoloMosaic` will only output the files needed by the FMS coupler. Two extra parameters were provided to write an ocean and land mask. These will be used later for the ocean mask editor. The land and ocean masks are impacted if additional parameters, MASKING_DEPTH or MINIMUM_DEPTH, are specified. If these are not specified, these default to a depth of zero (0.0) meters. For more details, see *makeSoloMosaic()*.

2.1.6 Examine the topography grid

In this section, two graphics are prepared. The first is a look at the current topography grid. The second graphic is the ocean mask.

Let us take a closer look at the model grid by plotting a high resolution coastline over the topography.

First, some plot parameters have to be specified. The function *plotGrid()* is called. This function returns figure and axes matplotlib objects that can be further manipulated. The figures are displayed by using a display() function.

[7]:

```
# Examine the topography grid
grd.setPlotParameters({
    'figsize': (8,8),
    'projection': {
        'name': 'LambertConformalConic',
```

(continues on next page)

(continued from previous page)

```
        'lon_0': 230.0,
        'lat_1': 25.0,
        'lat_2': 55.0
    },
    'extent': [-160.0 ,-100.0, 20.0, 60.0],
    'iLinewidth': 1.0,
    'jLinewidth': 1.0,
    'showGridCells': True,
    'iColor': 'k',
    'jColor': 'k',
    'transform': cartopy.crs.PlateCarree(),
    'satelliteHeight': 35785831.0
})
(figure, axes) = grd.plotGrid(showModelGrid = True,
        plotVariables={
        'depth': {
            'values': topoGrids['depth'],
            'title': 'Ocean topography (meters)',
            'cbar_kwargs': {
                'orientation': 'horizontal',
            }
        }
    })
display(figure)

# Examine the ocean mask
oceanMask = grd.openDataset(os.path.join(inputDir, 'ocean_mask.nc'))

# Define our own color map (same used in mask editor)
import matplotlib.pyplot as plt
land_color = (0.6, 1.0, 0.6)
sea_color  = (0.6, 0.6, 1.0)
maskCM = plt.matplotlib.colors.ListedColormap(
    [land_color, sea_color], name='land/sea')

# MOM6 places lon and lat in x and y
# x and y need to be lon and lat coordinates for the mask editor
oceanMask = oceanMask.rename({
    'x': 'lon',
    'y': 'lat'
})
oceanMask = oceanMask.set_coords(['lon', 'lat'])

(figureMask, axesMask) = grd.plotGrid(showModelGrid = True,
        plotVariables={
        'mask': {
            'values': oceanMask['mask'],
            'title': 'Ocean mask (1 = ocean)',
            'cmap': 'land/sea',
            'cbar_kwargs': {
                'orientation': 'horizontal',
            }
```

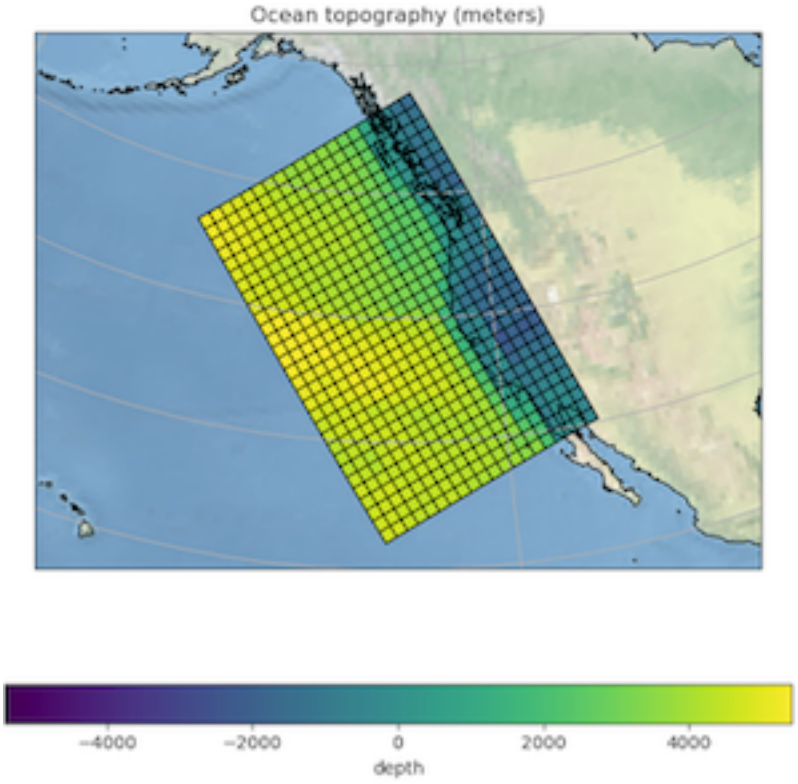
(continues on next page)

(continued from previous page)

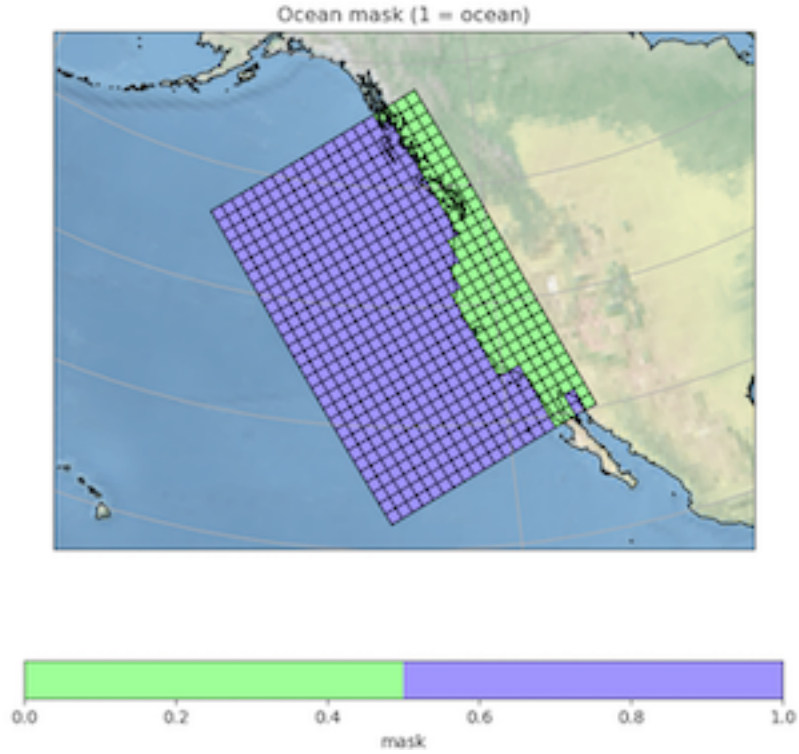
```
        }
    })
display(figureMask)

# Zoom in to take a closer look
grd.setPlotParameters({
    'extent': [-140.0 ,-120.0, 49.0, 59.0]
})

(figureMaskZoom, axesMaskZoom) = grd.plotGrid(showModelGrid = True,
        plotVariables={
        'mask': {
            'values': oceanMask['mask'],
            'title': 'Ocean mask (1 = ocean): Zoom',
            'cmap': 'land/sea',
            'cbar_kwargs': {
                'orientation': 'horizontal',
            }
        }
    })
display(figureMaskZoom)
```

When this cell is run, three plots should appear.

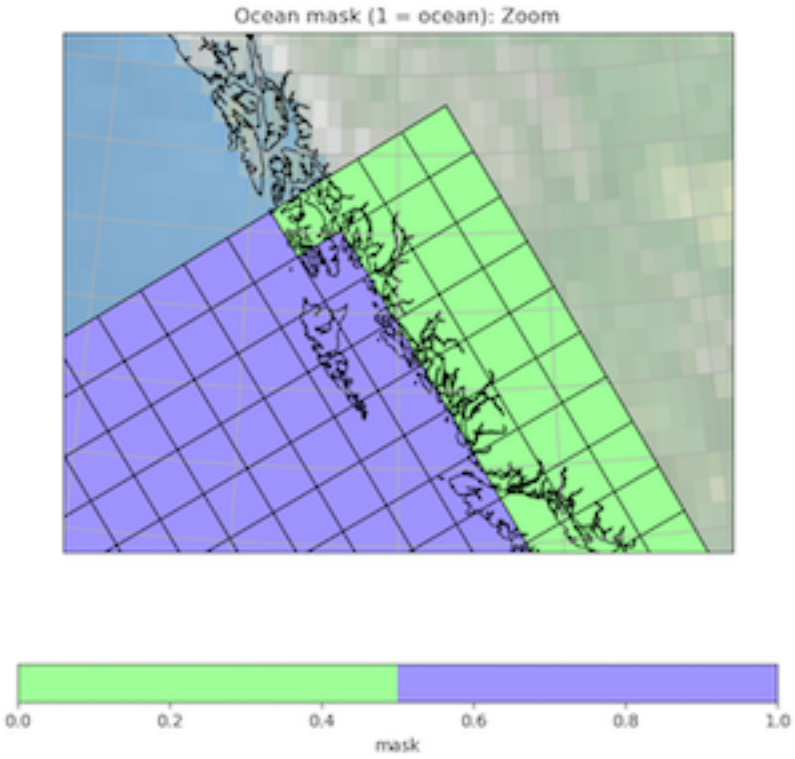
Ocean Topography



Ocean Mask Full Grid



Ocean Mask Zoomed



The ocean mask looks pretty good. In the next section, start the grid editor to change some of the points from ocean to land and land to ocean.

2.1.7 Use the editor to make some mask updates

To start up the mask editor, create a mask editor object with the desired projection. Create the mask editor application object and then use the display() function to launch the application.

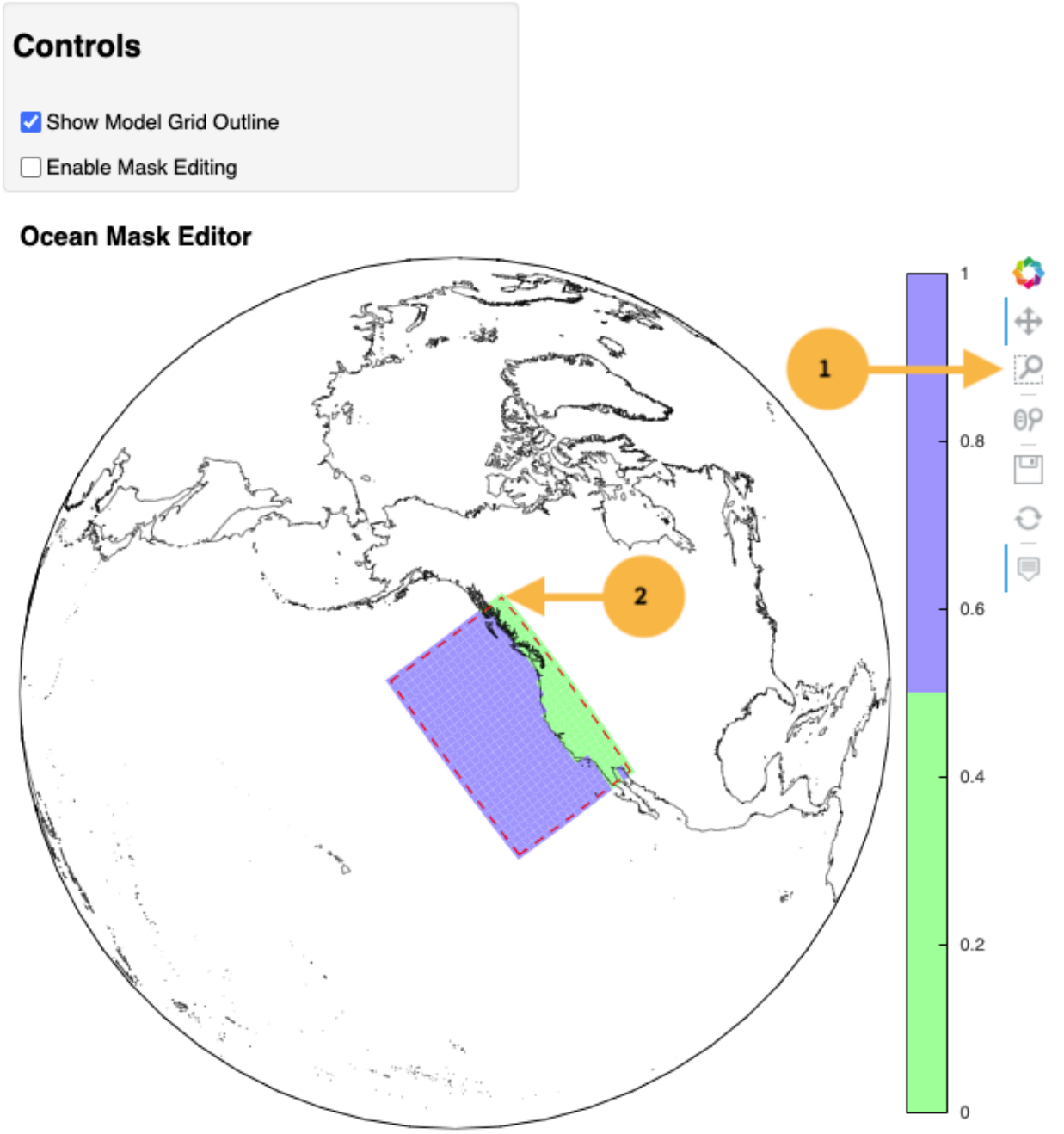
For additional details about the operation of the grid editor, please see "*Jupyter Mask Editor*".

[8]:

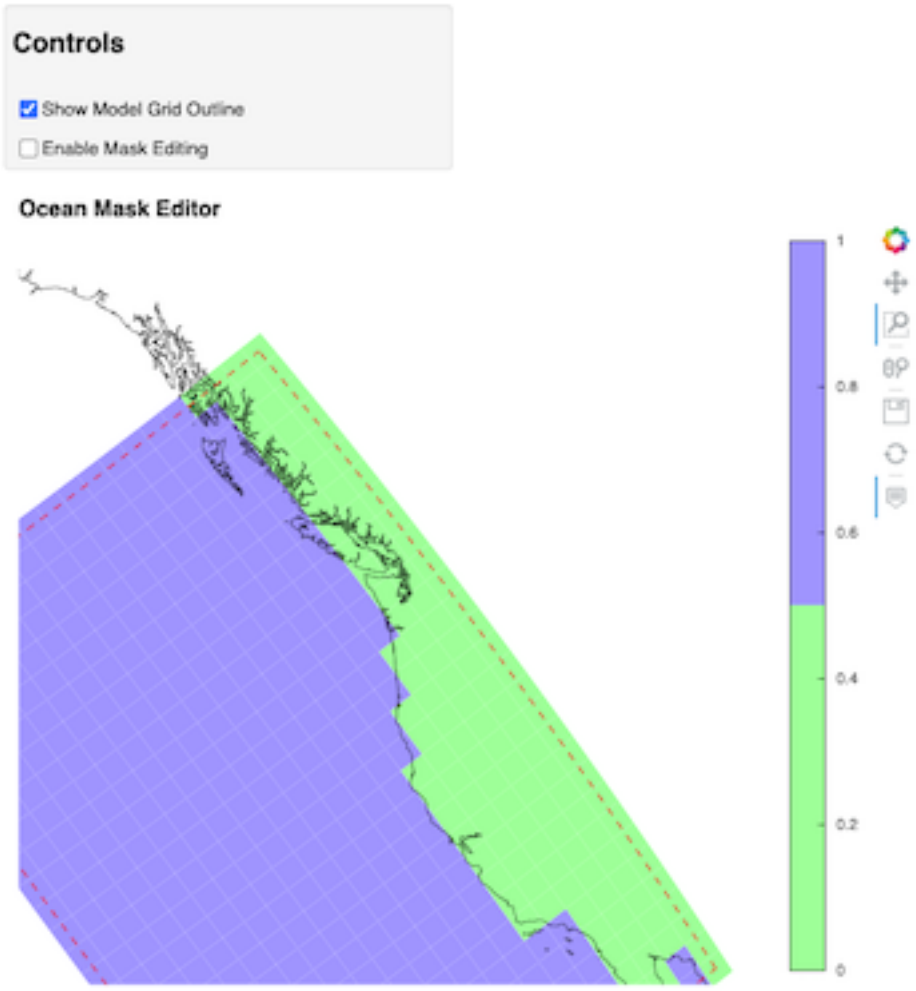
```
# Load the mask editor application module from gridtools
from gridtools.app import maskEditor

# Set a map projection for the mask editor to use
crs = cartopy.crs.Orthographic(-140, 45)

# Create the mask editor object
appObj = maskEditor(crs=crs, ds=oceanMask['mask'])

# Create the mask editor application object
app = appObj.createMaskEditorApp()

# Launch the application
display(app)
```

A successful launch of the application should look similar to the figure below. Start by selecting the zoom control and

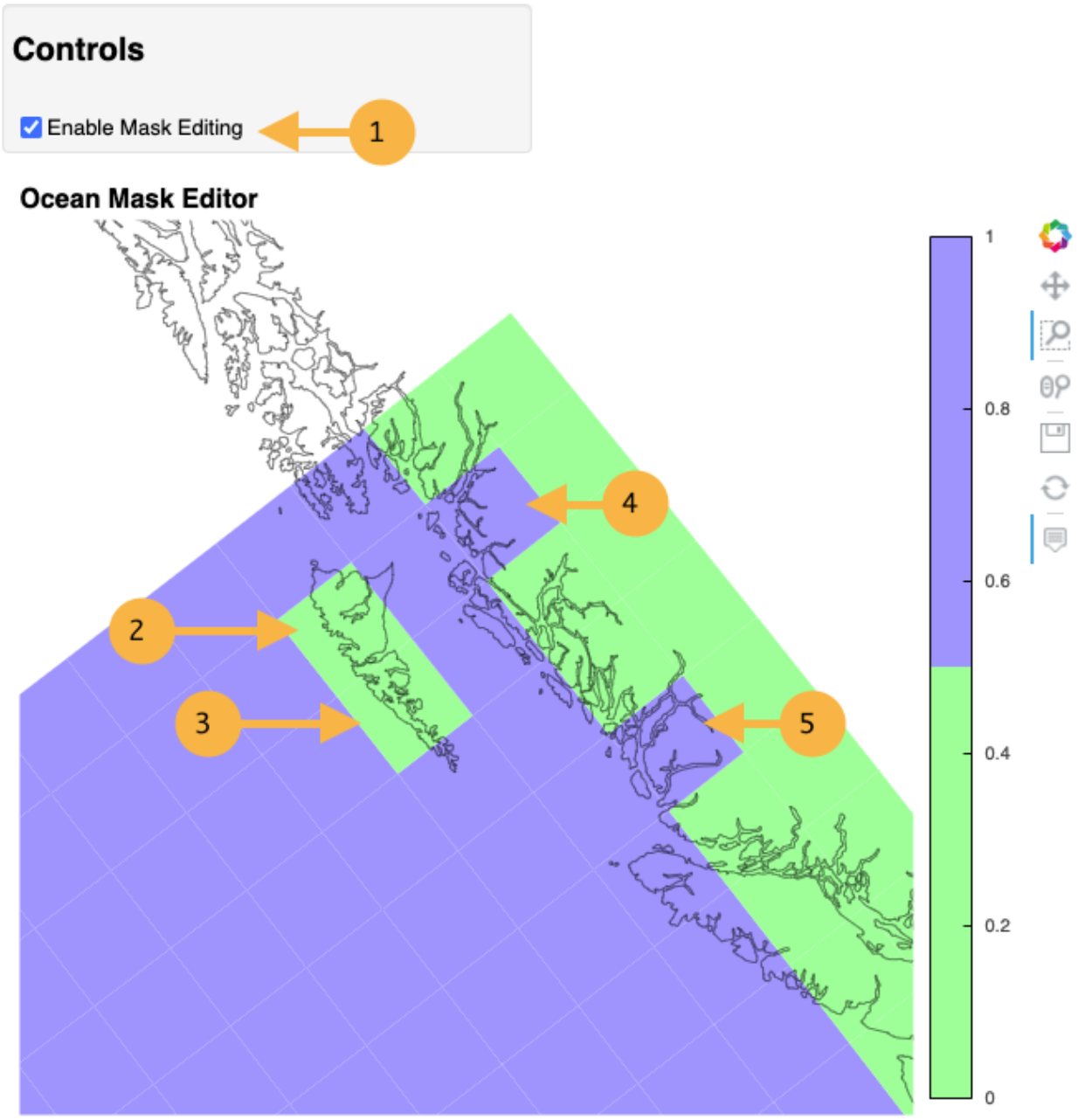
zooming into the same area as the figure above.



Once the zoom tool is selected, click and draw a box over the region to zoom. Releasing the mouse button should result in a redrawn map.



Clicking on the “Enable Mask Editing” checkbox, will allow mouse clicks on the grid to flip between land and ocean. Click two ocean boxes to change them to land. Click two land points to turn them to ocean.



The new ocean mask can be saved using the following code.

[9]:

```
# Save the new ocean mask
newMask = oceanMask['mask'].copy()
newMask = newMask.reset_coords(names = ['lat', 'lon'])
grd.saveDataset(os.path.join(inputDir, 'ocean_mask_new.nc'), newMask,
                overwrite=True, mapVariables = {'lon': 'x', 'lat': 'y'},
                hashVariables = ['mask', 'x', 'y'])
```

2.1.8 Apply mask changes to the model grid

The new ocean mask is applied to the current model grid. In this example, the default values are passed to *MASKING_DEPTH*, *MINIMUM_DEPTH* and *MAXIMUM_DEPTH* to show that these parameters can be set. Be sure that these match the parameter values specified in your MOM6 input files.

[10]:

```
# Apply new ocean mask to ocean model grid
topoGrids['depth'] = grd.applyExistingOceanmask(topoGrids, 'depth',
    os.path.join(inputDir, 'ocean_mask_new.nc'), 'mask',
    MASKING_DEPTH=0.0, MINIMUM_DEPTH=0.0, MAXIMUM_DEPTH=-99999.0)
```

Of the four points that were changed, this should be the expected result after running the above routine:

```
The (diagnosed) maximum depth of the ocean is 5413.075256 meters.
Beginning application of new ocean mask (changes noted, if any).
 * Number of land mask points with new depth of 0.000000: 2
 * Number of ocean points with new depth of 0.000000: 2
```

Warning: The two ocean points with a depth of 0.000000 in this case is incorrect. For MOM6, a *MASKING_DEPTH* set to 0.000000 means that depths of **0.000000 or shallower** will be masked as land. When the *MASKING_DEPTH* and *MINIMUM_DEPTH* are **EQUAL**, an additional depth of *epsilon* is applied so the new point is actually an ocean point. The value of *epsilon* may need to be changed if the new ocean points are masked by MOM6.

See: *applyExistingOceanmask()* or *applyExistingLandmask()* for additional details.

2.1.9 Write updated FMS coupler and mosaic files

To finish the process of updating the model grid, the FMS coupler, mosaic, topography and model grid are written.

[11]:

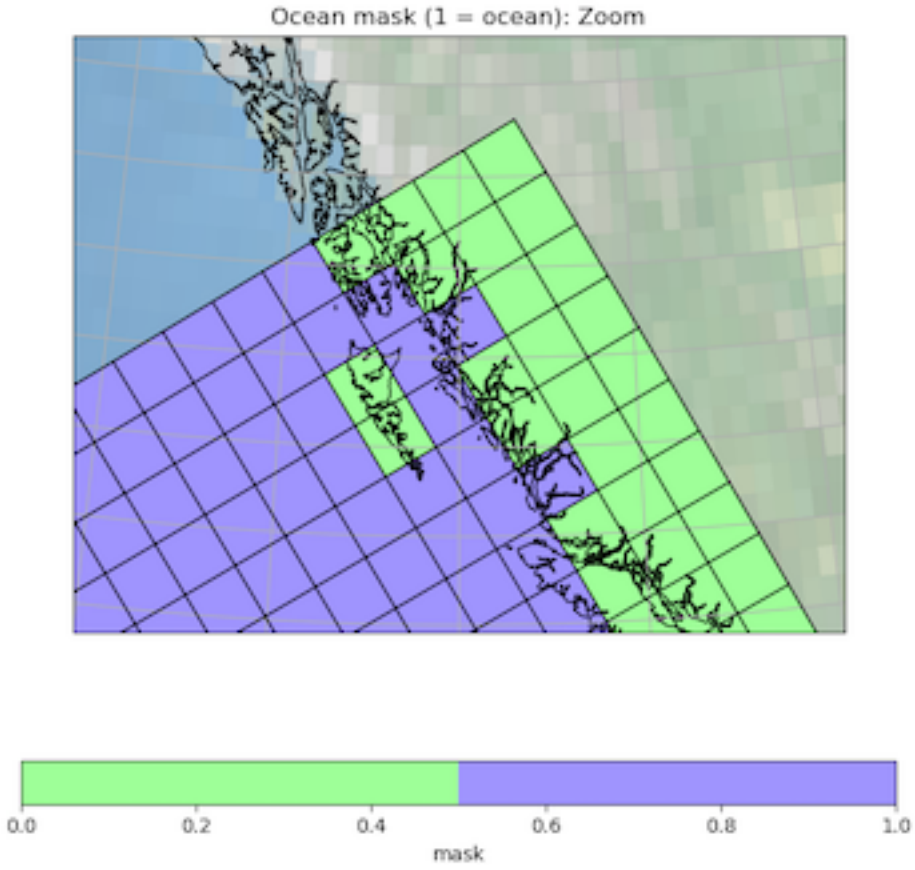
```
# Rewrite FMS coupler and mosaic files
grd.makeSoloMosaic(
    topographyGrid=topoGrids['depth'],
    writeLandmask=True,
    writeOceanmask=True,
    inputDirectory=input2Dir,
    overwrite=True,
    MASKING_DEPTH=0.0, MINIMUM_DEPTH=0.0, MAXIMUM_DEPTH=-99999.0
)

# Be sure to save previously diagnosed `h2` grid
topoGrids.to_netcdf(os.path.join(input2Dir, 'ocean_topog.nc'),
        encoding=grd.removeFillValueAttributes(data=topoGrids))

grd.saveGrid(filename=os.path.join(input2Dir, "ocean_hgrid.nc"))
```

2.1.10 Check final ocean mask

Plot the final ocean mask to be sure the points are correctly represented.

[12]:

```
(figureMaskZoom2, axesMaskZoom2) = grd.plotGrid(showModelGrid = True,
        plotVariables={
        'mask': {
            'values': oceanMask['mask'],
            'title': 'Ocean mask (1 = ocean): Zoom',
            'cmap': maskCM,
            'cbar_kwargs': {
                'orientation': 'horizontal',
            }
        }
    })
display(figureMaskZoom2)
```



2.2 Build a new grid

This section describes how to set the grid parameters for building a new grid.

Please see *gridtools.gridutils.GridUtils.setGridParameters()* for explanation of all available parameters.

By default, a grid center is defined. A total distance in the x and y direction of the grid is defined. The number of grid points is determined by the defined resolution of the grid cells in the x and y direction.

This example creates the IBCAO grid as described by the Technical Reference and User's Guide. [JCW+00] [MJV03]

The user manual states that the IBCAO grid is a Cartesian grid which has coordinates in meters in the x and y direction. The grid center or **Origin** is the North Pole. The Cartesian grid system starts at an (x, y) of $(0, 0)$ at the North Pole.

The grid size is 580.5 km in both directions giving a size of 2902500 meters on all sides of the North Pole.

To create an IBCAO grid suitable for MOM6, grid parameters need to be set to appropriate values.

The user manual states the IBCAO grid is in the **Polarsterographic** projection. For *gridtools*, this is specified as `Stereographic`. In that projection, the true scale is preserved at 75 degrees North. For *gridtools*, this is specified by setting `lat_ts` to `75.0`. Although the grid in Cartesian space is $(0, 0)$, in *gridtools*, the grid center needs to be specified in map coordinates for the projection center and the grid center.

To specify that the grid should be centered at the North Pole, in *gridtools* set the grid projection parameters `lon_0` to `0.0` and `lat_0` to `90.0`. The grid parameters `CenterX` is also `0.0` and `CenterY` is `90.0`. The `CenterUnits` should be set to `degrees`.

The user manual specifies a standard radius of the Earth by stating that the "horizontal datum is World Geodetic System (WGS 84)". This is specified in *gridtools* using the projection parameter of `WGS84` for the `ellps` (ellipsoid) parameter.

The user manual states the grid distance is 2.5 km. The grid resolution is 2500 meters. In *gridtools*, `dx` and `dy` are set to the total distance of the grid or `5805000.0` meters. The `gridResolution` is set to `2500.0` meters. The units should be set to `meters` for `dxUnits`, `dyUnits`, and `gridResolutionUnits`.

The grid parameters `tilt` is optional. The default value of `0.0` is shown in the example.

For now, MOM6 grids should all be created using a `gridMode` of `2` to specify creation of a supergrid. For MOM6 grids, the `gridType` should be `MOM6`.

The parameters `ensureEvenI` and `ensureEvenJ` ensure the supergrid is properly sized. Use the default value of `True` for now.

Here is the command pulling all the above parameters together ready to create the IBCAO grid:

```
# Create a gridtools object
from gridtools.gridutils import GridUtils
grd = GridUtils()

# Define IBCAO grid for gridtools library
grd.setGridParameters({
    'projection': {
        'name': "Stereographic",
        'ellps': 'WGS84',
        'lon_0': 0.0,
        'lat_0': 90.0,
        'lat_ts': 75.0,
    },
    'centerX': 0.0,
    'centerY': 90.0,
    'cneterUnits': 'degrees',
```

(continues on next page)

(continued from previous page)

```
    'dx': 5805000.0,
    'dy': 5805000.0,
    'dxUnits': 'meters',
    'dyUnits': 'meters',
    'gridResolution': 2500.0,
    'gridResolutionUnits': 'meters',
    'tilt': 0.0,
    'gridMode': 2,
    'gridType': 'MOM6',
    'ensureEvenI': True,
    'ensureEvenJ': True
})
```

After setting the grid parameters, the next command will instruct *gridtools* to make the grid:

```
grd.makeGrid()
```

For systems with smaller amounts of memory, example #6 constructs a mini IBCAO grid with fewer grid points.

2.3 Edit a grid

There are two general ways to edit a ocean model grid.

2.3.1 Edit a grid mask using Jupyter

A jupyter notebook was written for editing the ocean mask for a model grid.

The first step is to load the ocean mask for a model grid.

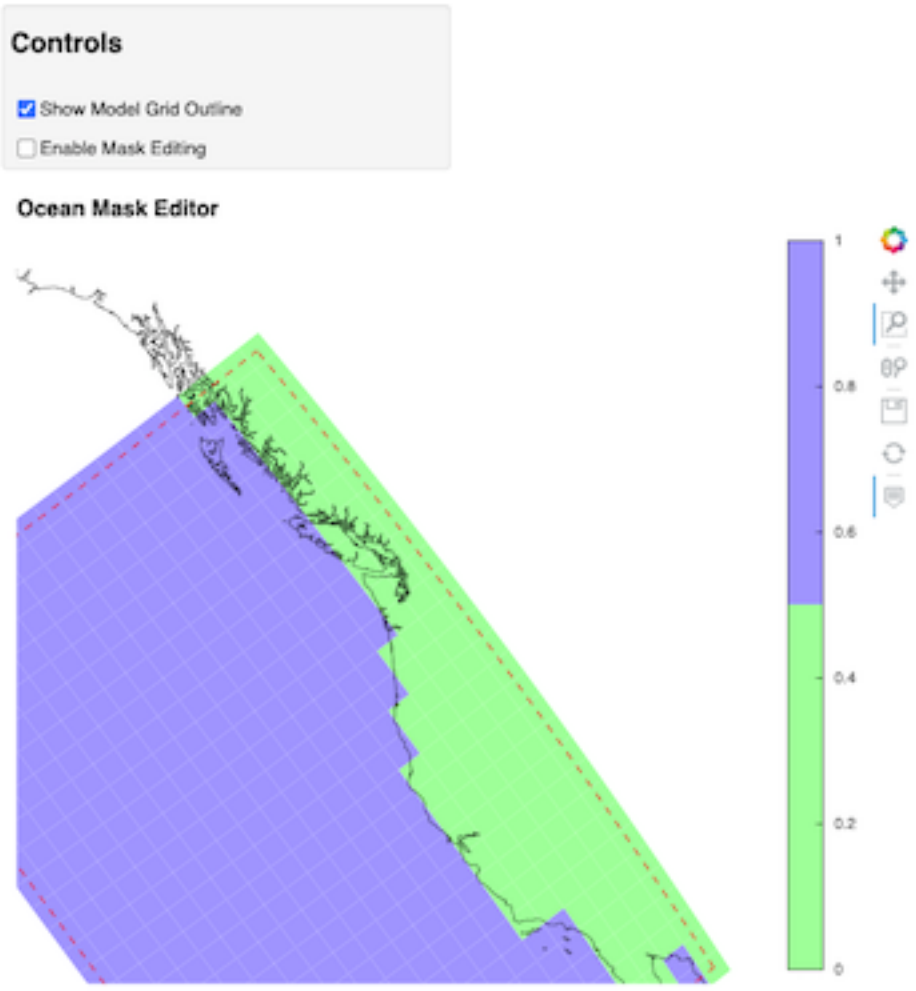
```
# Import needed python modules
import os, sys
from gridtools.gridutils import GridUtils
from gridtools.app import maskEditor
import cartopy.crs as ccrs

# Define the model grid directory
# For existing MOM6 grids, this is often an "INPUT" directory
wrkDir = '/import/AKWATERS/jrcermakiii/configs/zOutput'
inputDir = os.path.join(wrkDir, 'INPUT')

# Create a gridtools object
grd = GridUtils()

# Read the ocean mask
oceanMask = grd.openDataset(os.path.join(wrkDir, 'ocean_mask_Example7.nc'))

# The mask editor requires 'lat' and 'lon' coordinates for mapping.
# For MOM6, the x and y dimensions are renamed to lon and lat
oceanMask = oceanMask.rename({
    'x': 'lon',
```

(continues on next page)

(continued from previous page)

```
    'y': 'lat'
})
oceanMask = oceanMask.set_coords(['lon', 'lat'])

# Set a map projection for the mask editor
crs = ccrs.Orthographic(-140, 45)

# Create the mask editor
appObj = maskEditor(crs=crs, ds=oceanMask['mask'])
app = appObj.createMaskEditorApp()
display(app)
```

This will launch the map editor within the jupyter notebook.



When editing is complete, use and additional jupyter cells to save the edited ocean mask variable `oceanMask`.

To keep this cell from running when starting the mask editor, the `runBlock` is set to False. When it is time to save the new mask, set `runBlock` to True and run the cell.

```
runBlock = False
```

(continues on next page)

(continued from previous page)

```
if runBlock:
    # Save the new mask without coordinates
    newMask = oceanMask['mask'].copy()
    newMask = newMask.reset_coords(names = ['lat', 'lon'])
    grd.saveDataset(os.path.join(wrkDir, 'ocean_mask_new_Example7.nc'), newMask,
                    overwrite=True, mapVariables = {'lon': 'x', 'lat': 'y'},
                    hashVariables = ['mask', 'x', 'y'])
```

2.3.2 Edit a grid mask with pylab

The pylab grid mask editor was initially written for the ROMS ocean model. It has been ported to gridtools and will operate on ROMS grids. The editor needs to be updated to work more efficiently with ROMS and MOM6 grids.

See "*Pylab Mask Editor*" for instructions on how to launch the tool. The function call to write the ROMS grid is the same as the pyroms tool.

CHAPTER

THREE

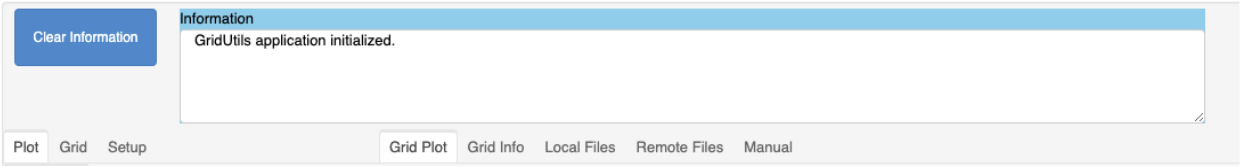
APPLICATIONS

This section describes application controls in greater detail.

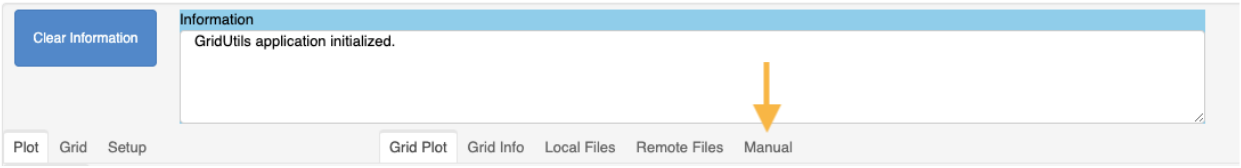
3.1 Grid Generation

This section provides detail on the controls of the grid generation application written for Jupyter notebooks.

The primary controls are located at the top of the grid generation application.



A subset of these descriptions are built into the grid generations "*Manual*" tab.

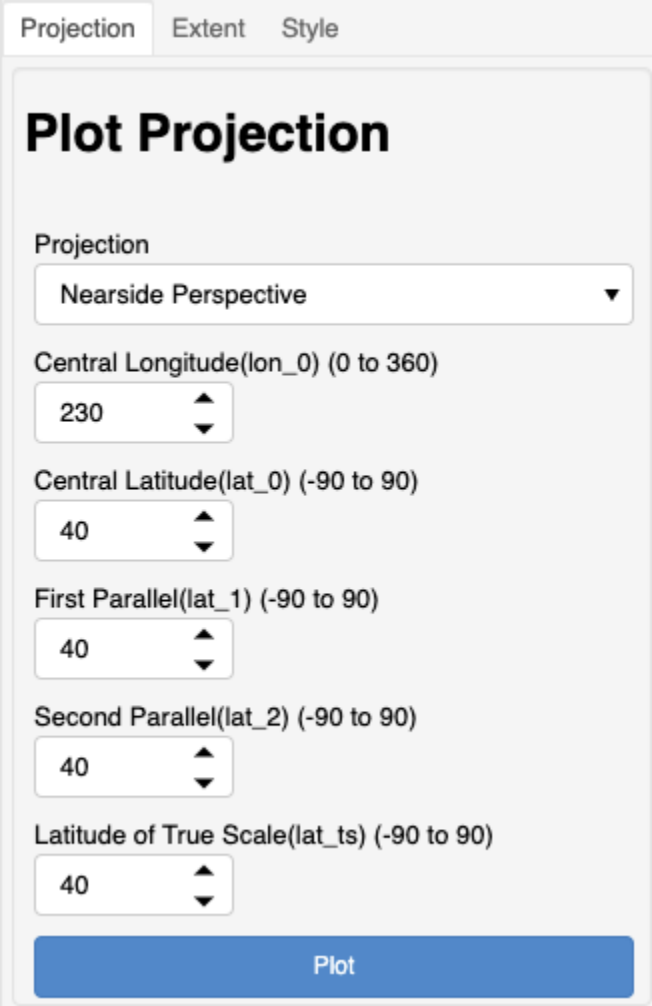


3.1.1 Plot

This section describes "*Plot*" controls.

Projection

This section describes the "*Projection*" controls for plotting.



The grid generation application supports the following projections for plotting of the model grid and other information:

- Nearside Perspective
- Mercator
- Lambert Conformal Conic
- Stereographic

Please see Cartopy projection list for more details on these projections.

The gridtools library attempts to conform to Proj terminology for setting projection parameters such as **latitude of true scale**.

Here is a comparison of **Cartopy** mapping parameters to parameters for use with **Proj** and **Gridtools**:

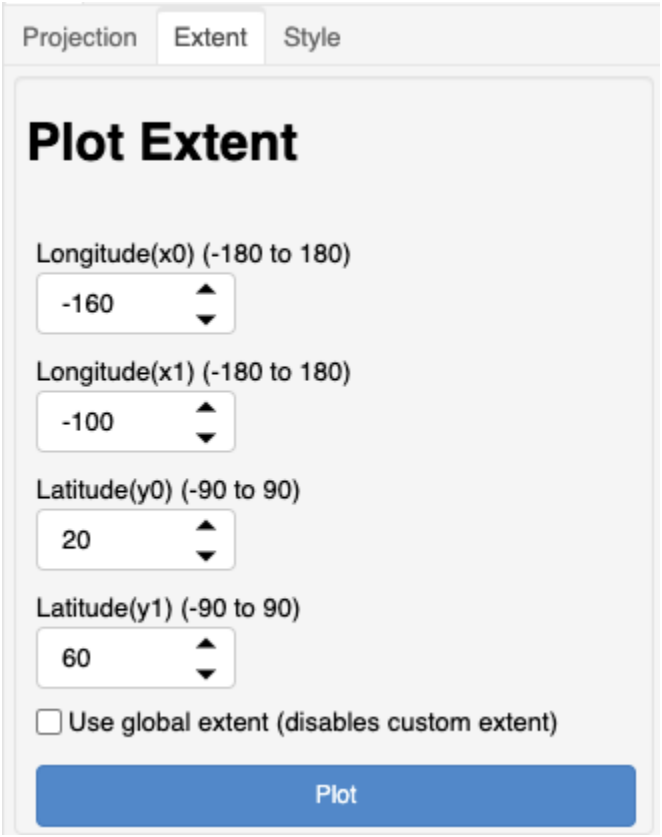
Cartopy	Proj	Gridtools
central_latitude	lat_0	lat_0
central_longitude	lon_0	lon_0
false_easting	x_0	x_0
false_northing	y_0	y_0
standard_parallels	lat_1, lat_2	lat_1, lat_2
latitude_true_scale	lat_ts	lat_ts
scale_factor	k_0	k_0
satellite_height	h	satelliteHeight

A "*Plot*" button is provided on each control to update the plot to the right after making any adjustment.

See also: `setPlotParameters()`.

Extent

This section describes the "*Extent*" controls for plotting.



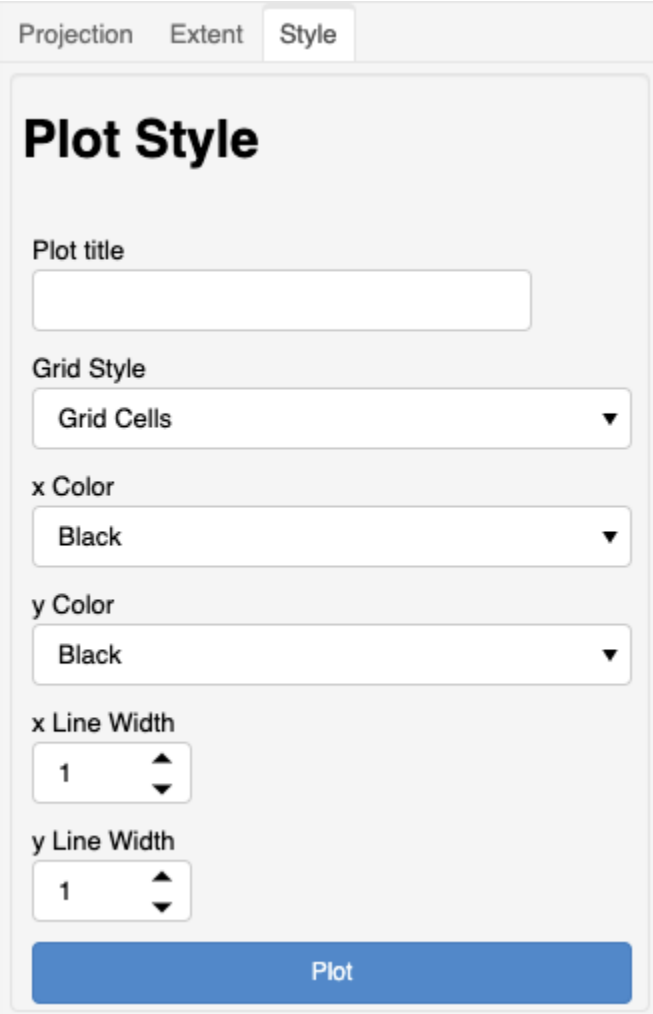
The extent controls only work in degrees (at the moment). The plot extent can be defined using the longitude and latitude controls or by checking the box to use a global extent.

These extents are set by the cartopy geoaxes object by calling *set_extent(x0, x1, y0, y1)*. When the global extent selected, the *set_global()* method is used.

See: geoaxes

Style

This section describes the "*Style*" controls for plotting.



The style controls allow for customization of the plots.

Plot Title

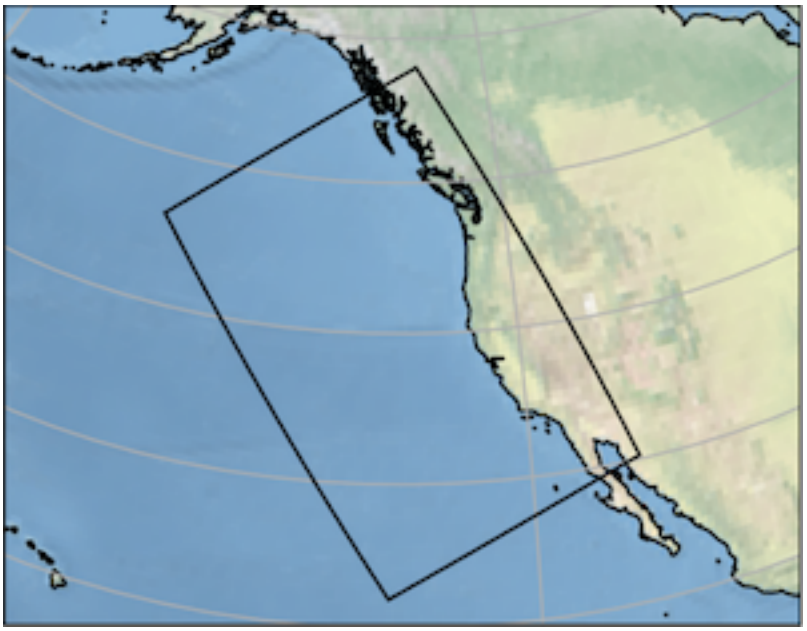
A default title is always generated for every plot. A custom plot title may be provided to replace the default title.

Grid Style

One of three styles may be selected.

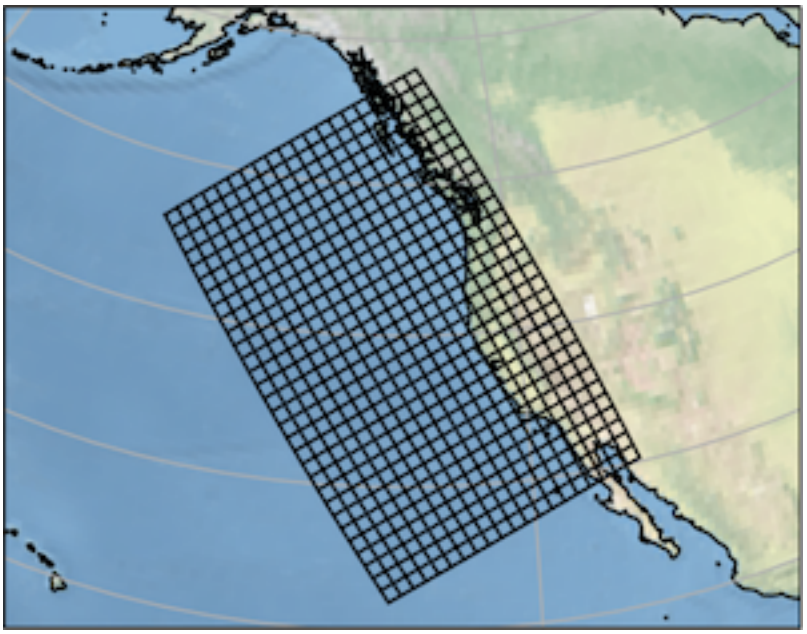
Grid Extent

This option will instruct the plot to only show the outline of the model grid.



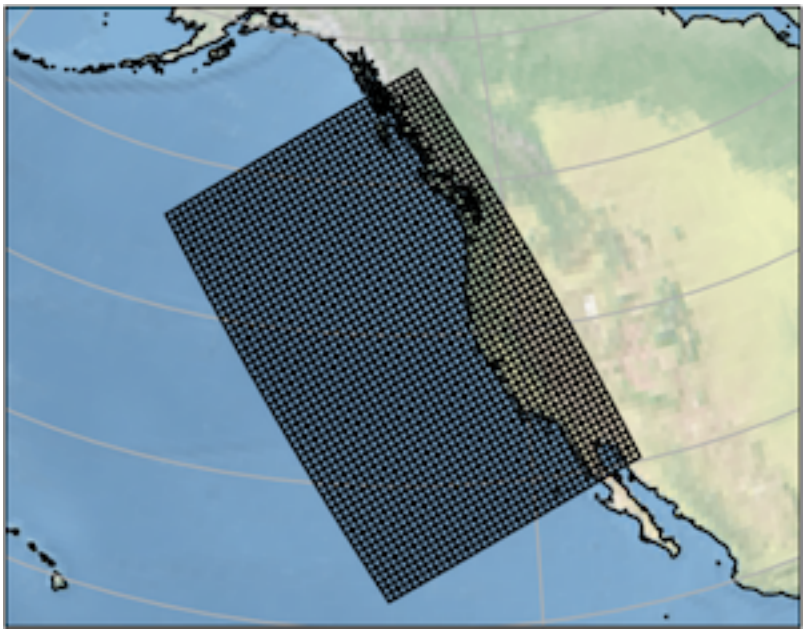
Grid Cells

This is the default style for plotting the model grid. This will show the nominal grid cells for the model grid.



Supergrid Cells

This option will instruct the plot to show the denser supergrid for the model grid. *This options supports MOM6 model grids.*

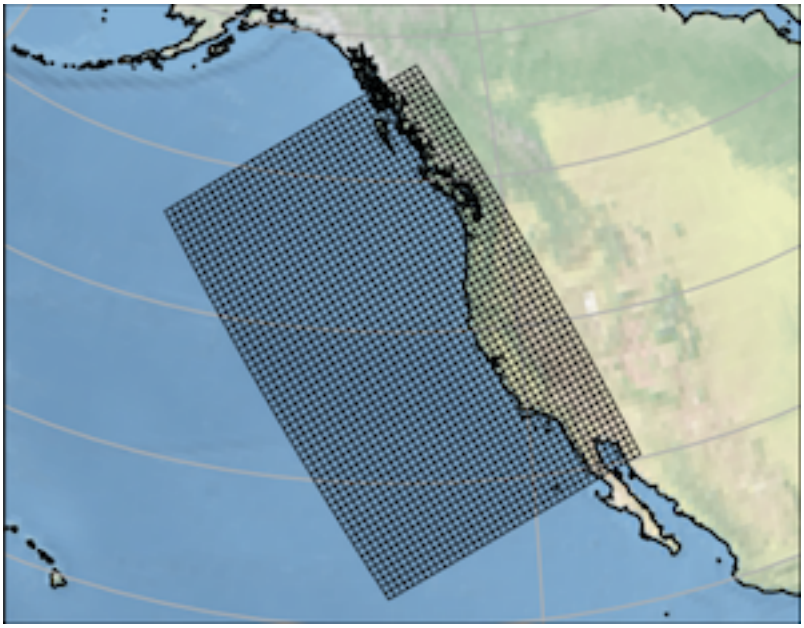


Line Width

These controls adjust the width of x and y lines drawn for the extent or grid cells. The line width can be changed from the default of one (1) dpi.

Note: For denser grids, it may be useful to change the length of the x and y line width to `0.5` or even `0.1` dpi.

Below is an example plot with x and y line width set to `0.5`.

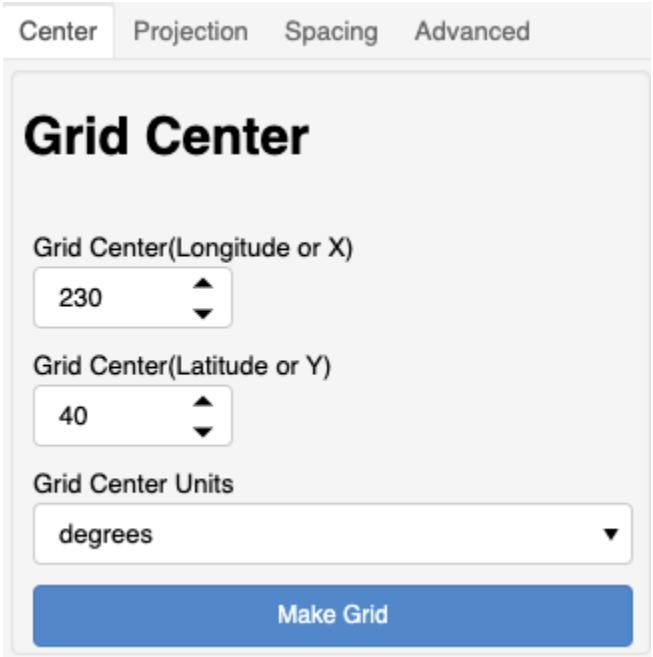


3.1.2 Grid

This section describes "*Grid*" controls.

Center

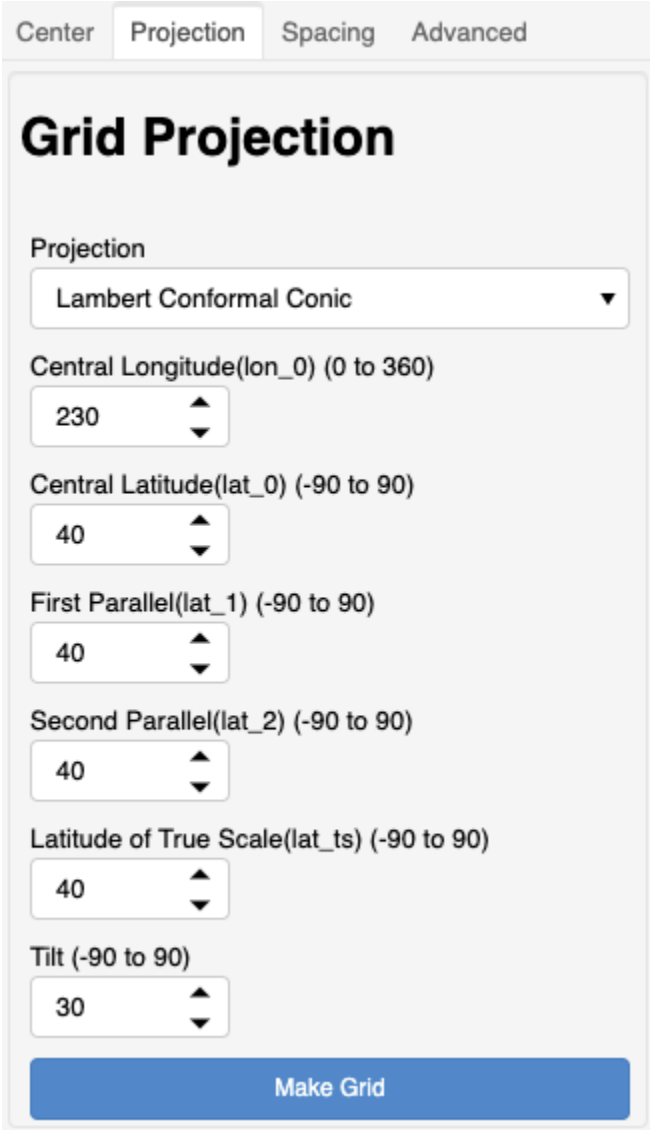
This section describes the "*Center*" controls for generation of the model grid.



This is the absolute center point of the entire grid. The center may be specified in *degrees* or *meters*.

Projection

This section describes the "*Projection*" controls for generation of the model grid.



Projection

Three grid projections are supported:

- Mercator
- Lambert Conformal Conic
- Stereographic

Please see Cartopy projection list for more details on these projections.

The gridtools library attempts to conform to Proj terminology for setting projection parameters such as **latitude of true scale**.

Here is a comparison of **Cartopy** mapping parameters to parameters for use with **Proj** and **Gridtools**:

Cartopy	Proj	Gridtools
central_latitude	lat_0	lat_0
central_longitude	lon_0	lon_0
standard_parallels	lat_1, lat_2	lat_1, lat_2
latitude_true_scale	lat_ts	lat_ts

Tilt is allowed to be specified for all projections. Conformality of Lambert Conformal Conic grids has been confirmed with usage of a tilt value that is non-zero.

Setting of **false_easting or false_northing** values is not implemented.

See also: *setGridParameters()*.

Spacing

This section describes the "*Spacing*" controls for generation of the model grid.

This controls the construction of the model grid about the center point.

Grid Extent

The *dx* and *dy* represent the **DISTANCE** of the entire model grid in the x and y directions. The **UNITS** need to be in degrees or meters.

Grid Resolution

The grid resolution in the X and Y direction in degress or meters. These represent the individual cell distances in the X and Y direction.

Number of grid points

The following equations are used to determine the number of grid points in the x and y directions.

$$nx = dx/X$$
$$ny = dy/Y$$

The supergrid will be of the size:

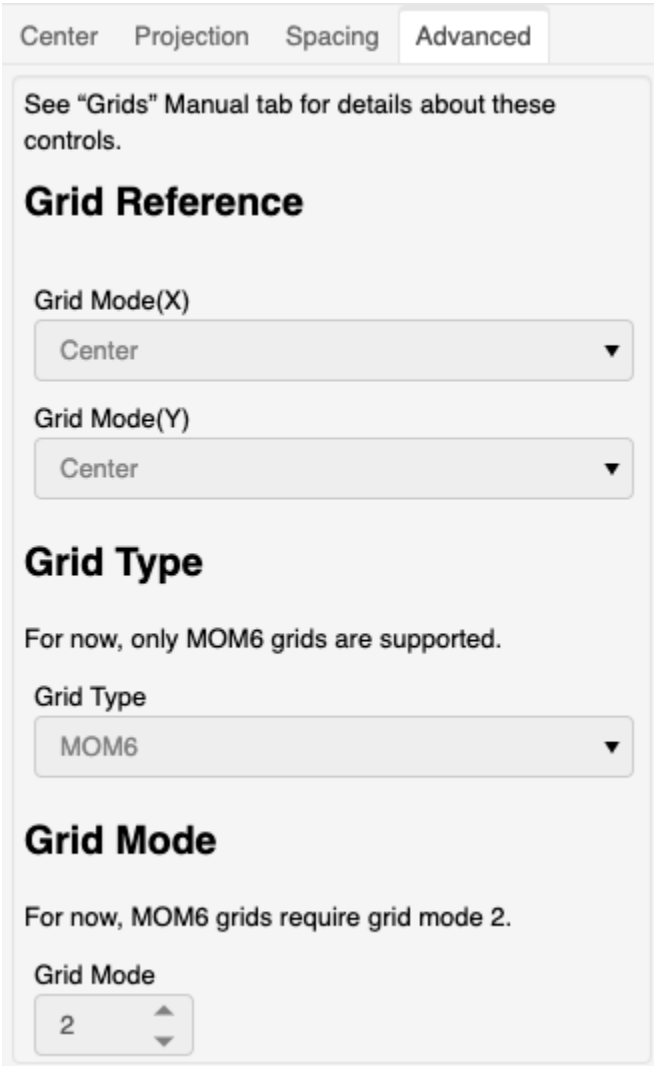
$$(ny * 2) + 1, (nx * 2) + 1$$

If the grid cell distance is *INCREASED* the number of grid cells will *DECREASE*.

Note: This is for the construction of the regular grid cells for a MOM6 model grid. A supergrid is automatically created for the regular MOM6 model grid.

Advanced

This section describes the "*Advanced*" controls for generation of the model grid.



For now, these controls are fixed to support only MOM6 model grids. There may be additional controls added to the gridtools grid generation application in the future.

Grid Reference

For now, the model grid can only be constructed from the *grid center*. In a future release, the grid may be specified in releation to other points of reference. For example, from a western point or north eastern point. In which case, the total distance and cell distances would expand from those boundaries instead of split at a center point.

Grid Type

For now MOM6, is the only model grid supported by the grid generation application.

Grid Mode

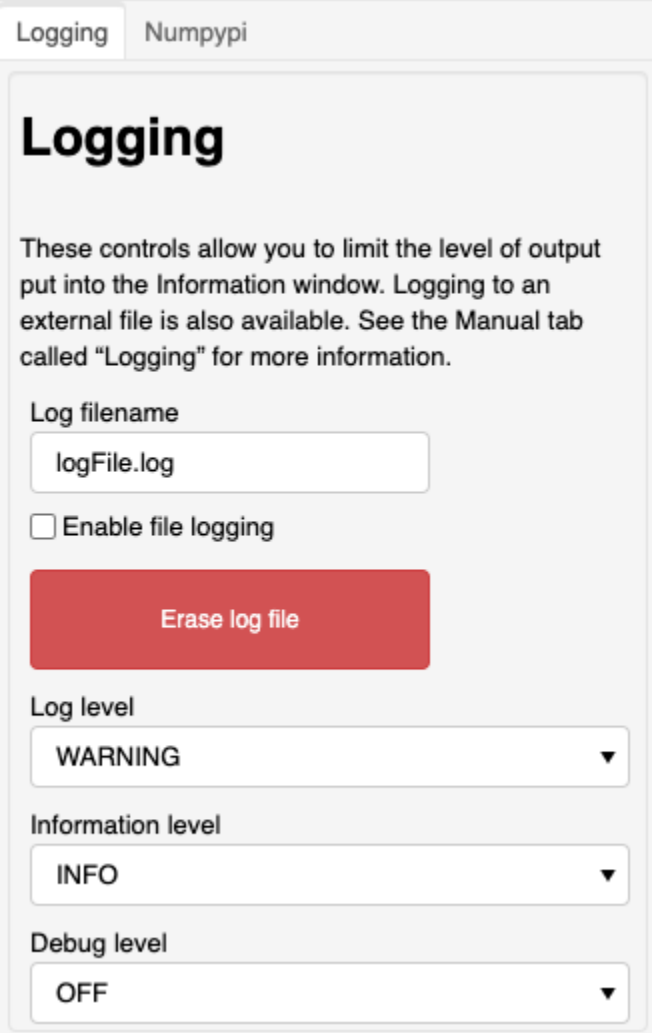
For MOM6, the grid mode is fixed at two (2). This specifies the generation of a supergrid even though the grid spacing is in reference to the regular MOM6 model grid cells. This may become clearer in future releases.

3.1.3 Setup

This section describes "*Setup*" controls.

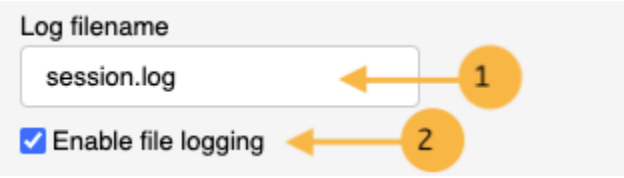
Logging

This section describes the "*Logging*" controls for logging and display of messages within the application.



This is for use with the grid generation application to assist with debugging and troubleshooting. These control the level of information provided to the information box in the application and to the log file, if enabled.

Instead of taking screen shots of the information window, messages can be logged to a file by setting a filename and checking the "Enable file logging" box. **Set the filename first, then check the checkbox.**



Note: Setting of the debug level above `MESSAGE` will not do much to help troubleshoot the grid generation application. In most cases, it will cause the application to quit operation at which point, the only way to continue is to restart the entire notebook.

For the log file to match everything that is displayed in the "*Information*" window, log level and information level should use the *SAME* setting.

See also

- *gridtools.gridutils.GridUtils.setLogLevel()*
- *gridtools.gridutils.GridUtils.setDebugLevel()*
- Additional logging information
- An example python script demonstrating the use of the gridtools logging facility: mkGridsExample02.py

Numpypi

This section describes the "*Numpypi*" controls for the application.

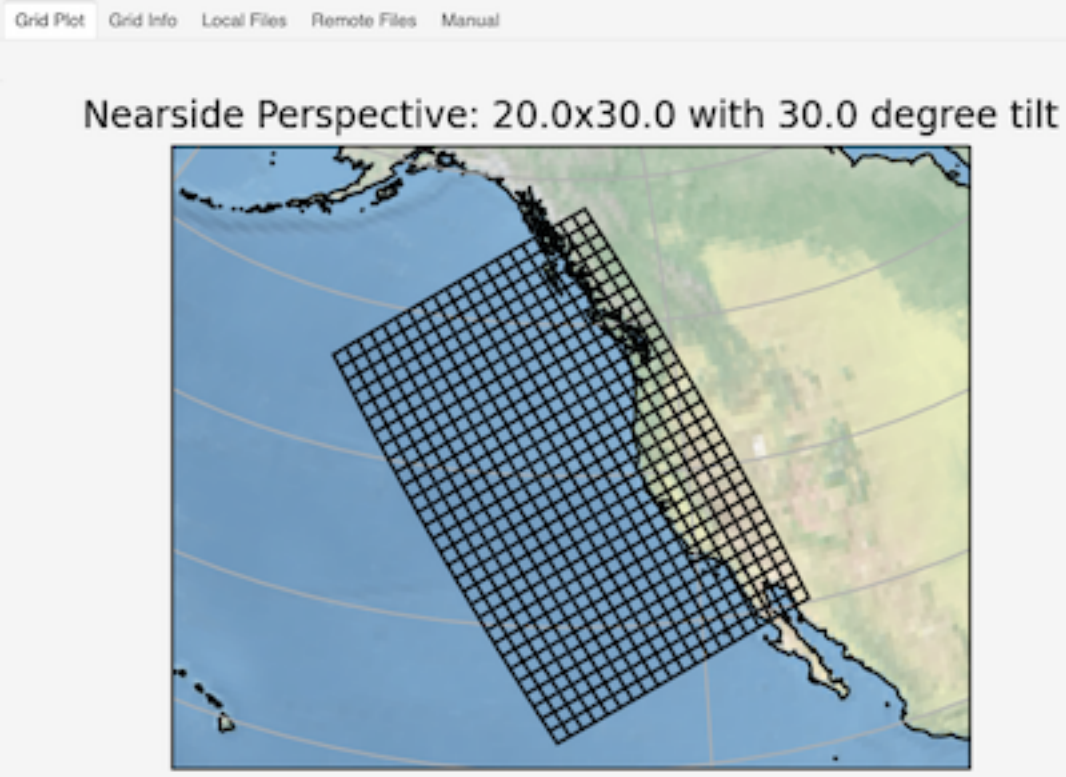
At the present time, this feature is **NOT ENABLED**. In a future release, the ability to use the `numpypi` package will be enabled to help guarantee bitwise-the-same operations between computer systems.

3.1.4 Grid Plot

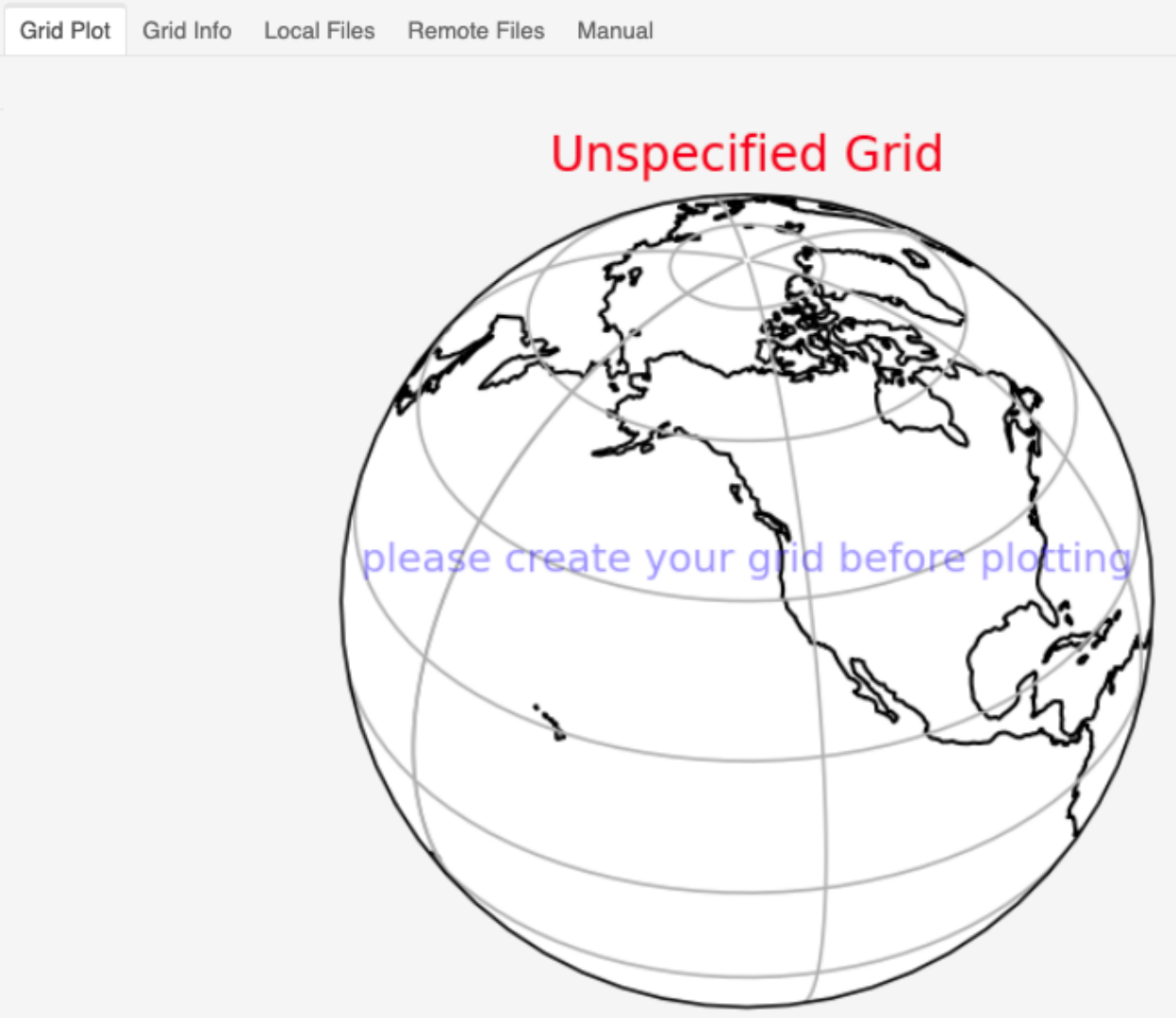
This section describes "*Grid Plot*" control tab.

This control displays the last available plot. The plot may also convey an important message.

Example plot:



Example of an important message:



3.1.5 Grid Info

This section describes "*Grid Info*" information tab.

This control displays the data structure currently stored. This should generally be an xarray Dataset object and should allow some basic interaction to look at a few values or important metadata.

Example:

xarray.Dataset

▶ Dimensions: (nx: 40, nxp: 41, ny: 60, nyp: 61)

▶ Coordinates: (0)

▼ Data variables:

x	(nyp, nxp)	float64	-131.0 -130.5 ... -129.1 -128.3
y	(nyp, nxp)	float64	22.06 22.31 22.56 ... 57.69 57.93
tile	()	<U5	'tile1'
dx	(nyp, nx)	float64	5.376e+04 5.376e+04 ... 5.376e+04
dy	(ny, nxp)	float64	5.566e+04 5.566e+04 ... 5.566e+04
angle_dx	(nyp, nxp)	float64	0.5526 0.5504 ... 0.5331 0.53
area	(ny, nx)	float64	2.996e+09 2.996e+09 ... 2.996e+09

▼ Attributes:

grid_version :	0.2
code_version :	GridTools: 0.3.0+2c15d5e
history :	2021-07-13 23:20:51: created grid with GridTools library
projection :	LambertConformalConic
grid_centerX :	230.0
grid_centerY :	40.0
grid_centerUnits :	degrees
grid_dx :	20
grid_dxUnits :	degrees
grid_dy :	30
grid_dyUnits :	degrees
grid_tilt :	30.0
software_version :	Cython 0.29.23; IPython 7.24.1; async_generator 1.10; backcall 0. 2.0; bokeh 2.3.2; cartopy 0.19.0.post1; certifi 2021.5.30; cffi 1.14.5;

Using a separate jupyter notebook cell, the grid information may also be accessed.

[1]:

```
print(grd.grid.software_version)
```

```
Cython 0.29.23; IPython 7.24.1; async_generator 1.10; backcall 0.2.0;
bokeh 2.3.2; cartopy 0.19.0.post1; certifi 2021.5.30; cffi 1.14.5;
cftime 1.5.0; chardet 4.0.0; click 7.1.2; cloudpickle 1.6.0;
colorama 0.4.4; colorcet 2.0.6; cycler 0.10.0; cython 0.29.23;
cytoolz 0.11.0; dask 2021.6.2; datashader 0.13.0; datashape 0.5.4;
decorator 5.0.9; defusedxml 0.7.1; distributed 2021.6.2;
entrypoints 0.3; fiona 1.8.18; fsspec 2021.6.1; geoviews 1.9.1;
gridtools 0.3.0+2c15d5e; holoviews 1.14.4; hvplot 0.7.2; idna 2.10;
importlib_metadata 4.5.0; ipykernel 5.5.5; ipython_genutils 0.2.0;
jedi 0.18.0; jinja2 3.0.1; jsonschema 3.2.0; jupyter_client 6.1.12;
jupyter_core 4.7.1; jupyterlab_pygments 0.1.2; kiwisolver 1.3.1;
llvmlite 0.36.0; markdown 3.3.4; markupsafe 2.0.1; matplotlib 3.4.2;
mistune 0.8.4; msgpack 1.0.2; multipledispatch 0.6.0; nbclient 0.5.3;
```

(continues on next page)

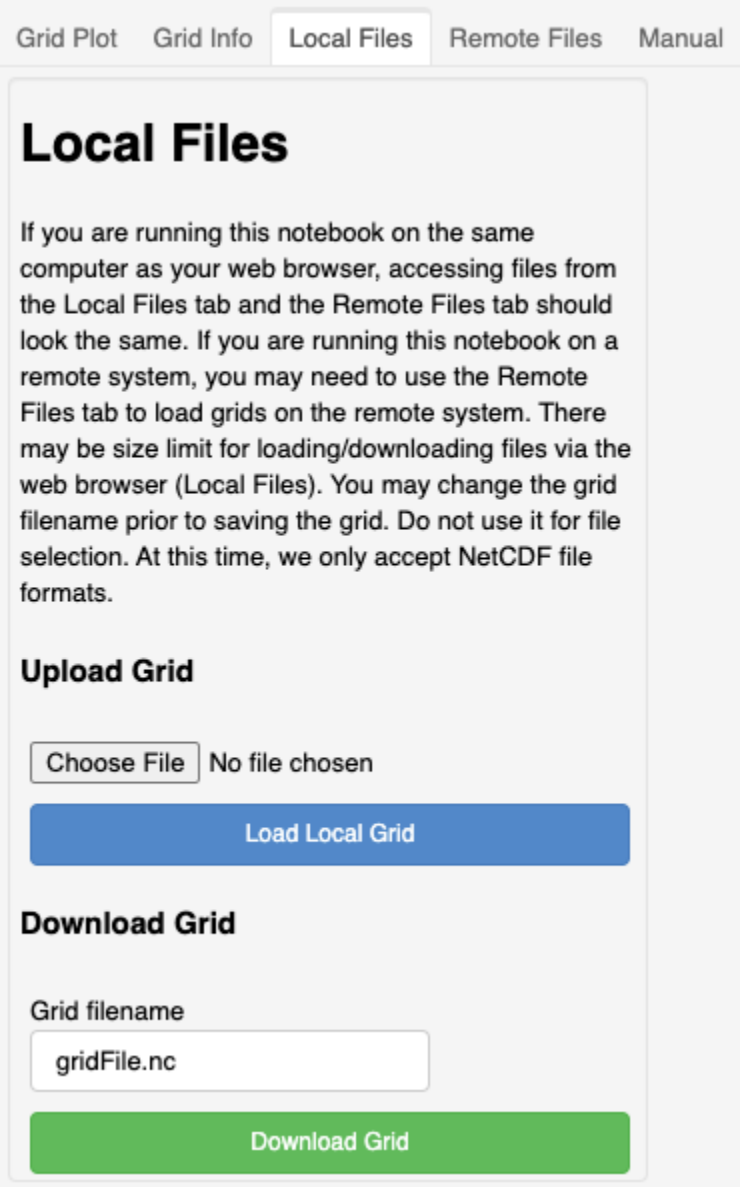
(continued from previous page)

```
nbconvert 6.1.0; nbformat 5.1.3; netCDF4 1.5.6; notebook 6.4.0;
numba 0.53.1; numpy 1.21.0; owslib 0.24.1; packaging 20.9;
pandas 1.2.5; pandocfilters 1.4.2; panel 0.11.3; param 1.10.1;
parso 0.8.2; pexpect 4.8.0; pickleshare 0.7.5; platform linux-x86_64;
prompt_toolkit 3.0.19; psutil 5.8.0; ptyprocess 0.7.0;
pycparser 2.20; pyct 0.4.8; pygments 2.9.0; pykdtree 1.3.4;
pyparsing 2.4.7; pyproj 3.1.0; pyrsistent 0.17.3; python 3.7.10;
pytz 2021.1; pyviz_comms 2.0.2; requests 2.25.1; scipy 1.6.3;
shapely 1.7.1; six 1.16.0; sortedcontainers 2.4.0; tblib 1.7.0;
testpath 0.5.0; toolz 0.11.1; tornado 6.1; traitlets 5.0.5;
typing_extensions 3.10.0.0; urllib3 1.26.5; wcwidth 0.2.5;
xarray 0.18.2; zipp 3.4.1
```

3.1.6 Local Files

This section describes "*Local Files*" control tab. Use this transfer if transferring a grid to and from a remote system. Transfer to and from a local system is also possible.

Warning: Some remote systems limit the amount of data that can be uploaded at one time.



Upload

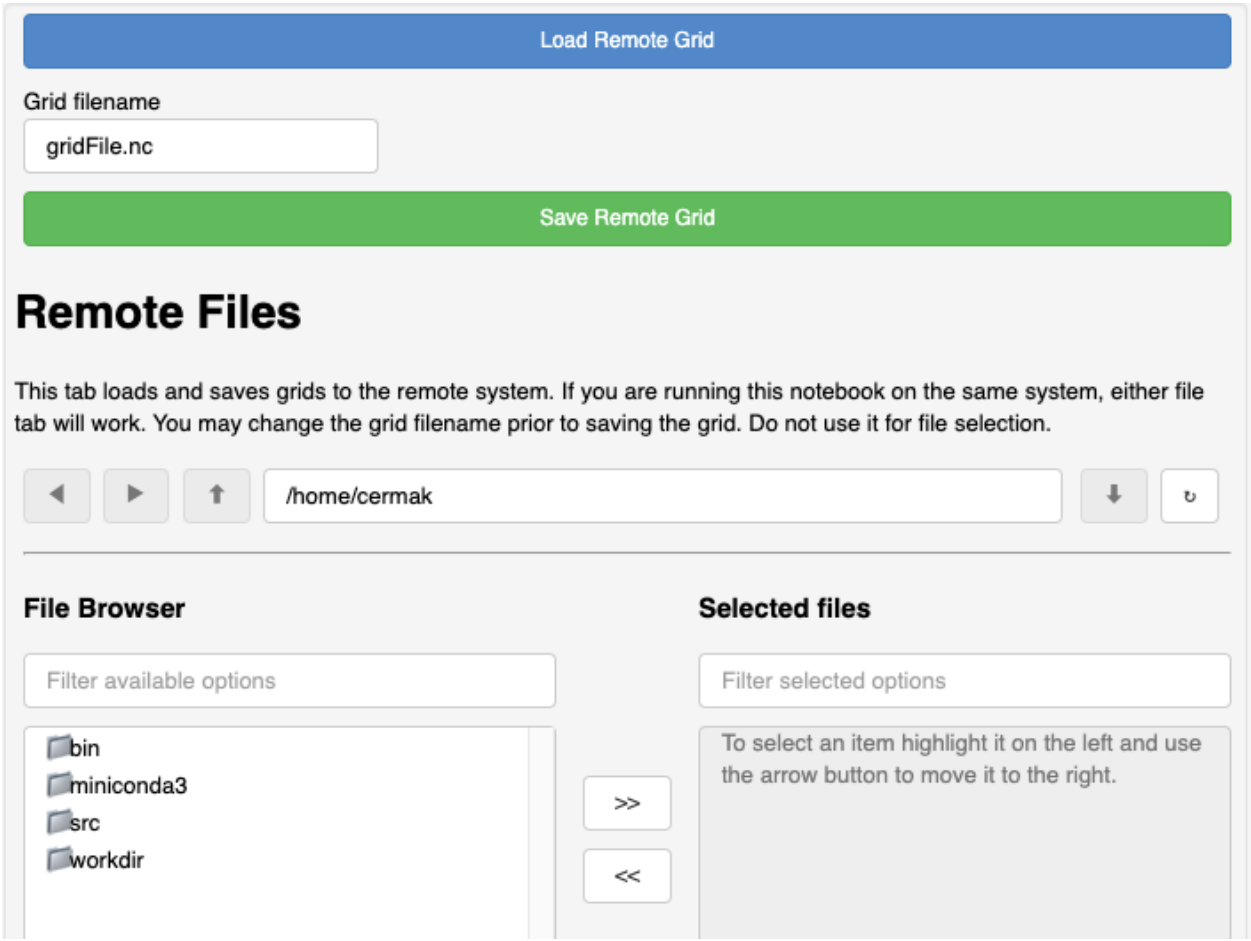
First choose the file to upload. Then click "Load Local Grid" to upload the model grid.

Download

Set the file name first. Then click the "Download Grid" button. The file should begin downloading to the directory set by the web browsers. The typical destination directory is "Downloads".

3.1.7 Remote Files

This section describes "*Remote Files*" control tab.



This control is somewhat complicated. Please read the directions carefully.

Warning: Large grids may take time to save. Until the saving is complete, the application may appear unresponsive.

A message should appear in the "Information" window indicating success in write the netCDF file.

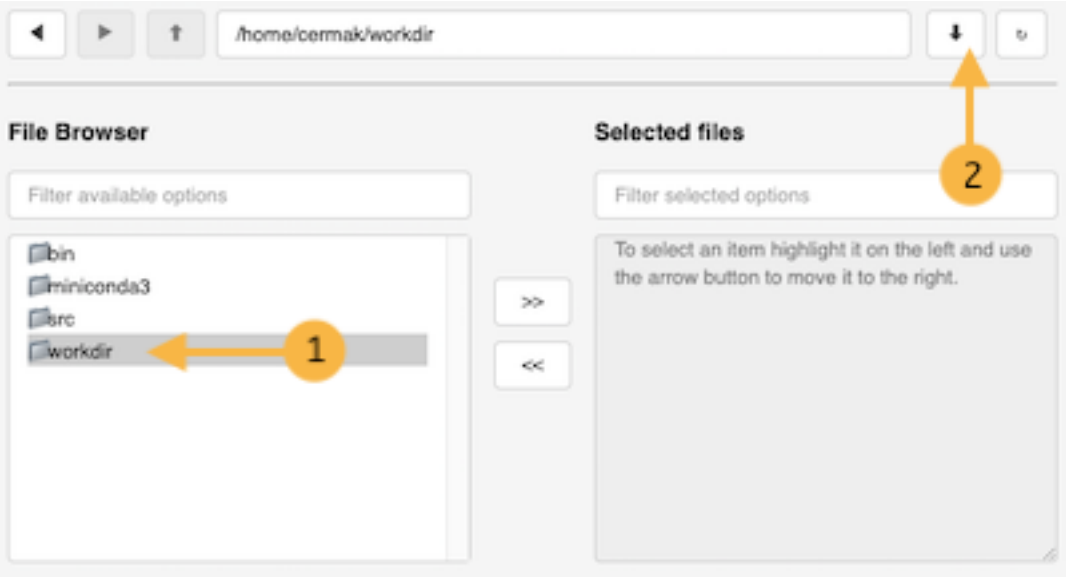
Any errors that occur when using this control may cause the application to crash and become unresponsive. They only way to restart the application is to restart the jupyter kernel and re-run all the program cells.

Load Remote Grid

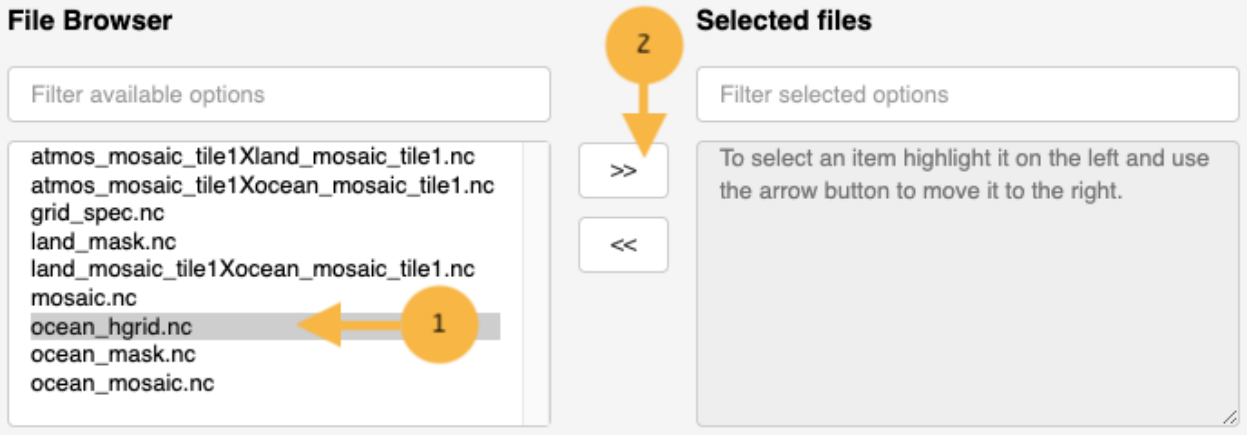
This control may be used to load any netCDF file on a remote system running the jupyter notebook. This control only works with files on the remote system running the jupyter notebook.

Navigating Directories

To decend into a directory tree, select the directory in the "File Browser" (the "left" side) by clicking once. Click the down arrow control on the right hand side to decend into that directory.



Keep decending until the netCDF file needed is shown in the "File Browser". Select the needed netCDF file and move it using the control between the "File Browser" and the "Selected Files". The controls show the direction it will move the selection between windows.



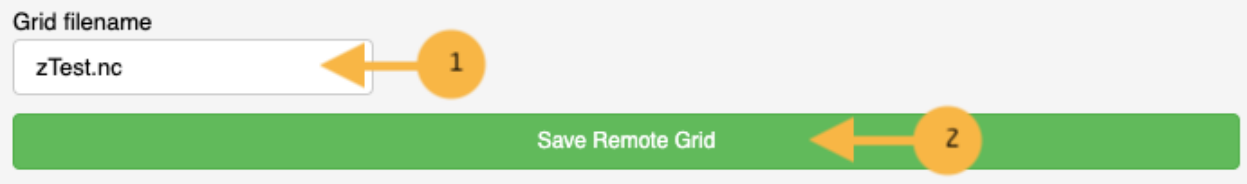
Once the netCDF file is shown in "Selected files", click on the blue button "Load Remote Grid". This will attempt to load the selected netCDF file into the grid generation application.

The "Information" window should indicate if the grid was loaded successfully.

Save Remote Grid

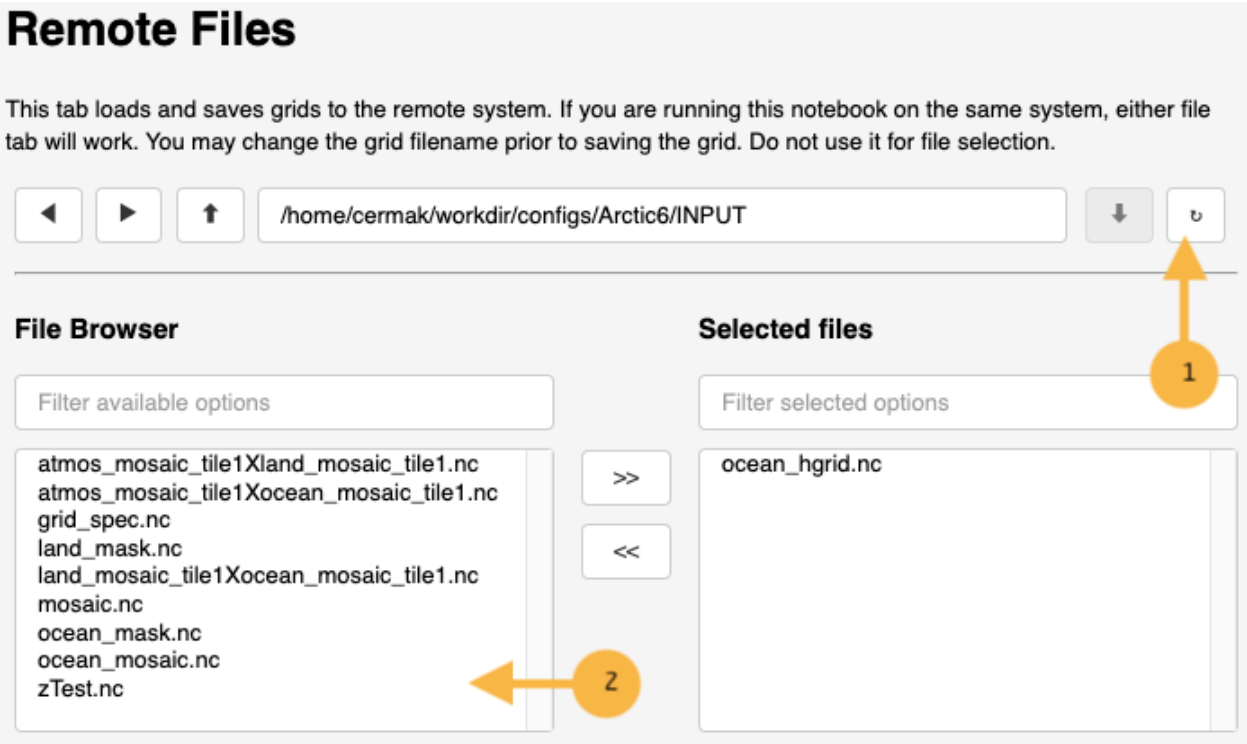
First, navigate to the appropriate directory. Make sure to decend into the directory and is shown under the "File Browser".

Update the "Grid filename" text box with the name file to use to save the model grid.



Press the "Save Remote Grid" button. For very large grids, this operation may take some time. An information message should indicate that the model grid was saved successfully.

The model grid just saved should appear in the "File Browser" **AFTER** clicking the refresh button.



3.1.8 Manual

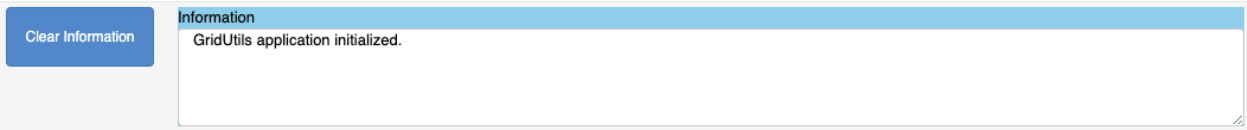
This section describes "*Manual*" information tab.

This tab has a subset of information describing the use of the grid generation application.

3.1.9 Information

This section describes the "*Information*" window and control at the top of the application.

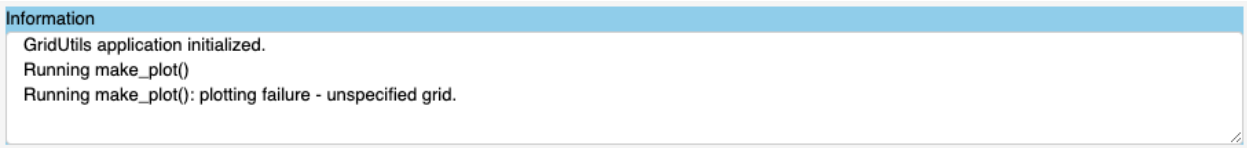
When the application is first started, the logging mechanism is attached to the "*Information*" window.



The application is currently running as a separate process and controls are weakly connected to the jupyter notebook. Any output from the application is captured and placed into the "*Information*" window.

Warning: If there is an application error, it is nearly impossible to debug since errors are not returned to the jupyter notebook.

Any errors or information captured by the application should be appended to the "*Information*" window.



At any time, the "*Clear Information*" button may be pressed to clear this window. Clearing the window will not erase previously logged information if logging to a file is enabled. See: "*Logging*" for more details on logging.

3.2 Mask Editors

The gridtools library maintains two mask editors. The first mask editor is implemented in Jupyter. The second mask editor is implemented in pylab.

3.2.1 Jupyter Mask Editor

Once a model grid is read and a map projection created, the application can be launched with these commands in a jupyter notebook cell:

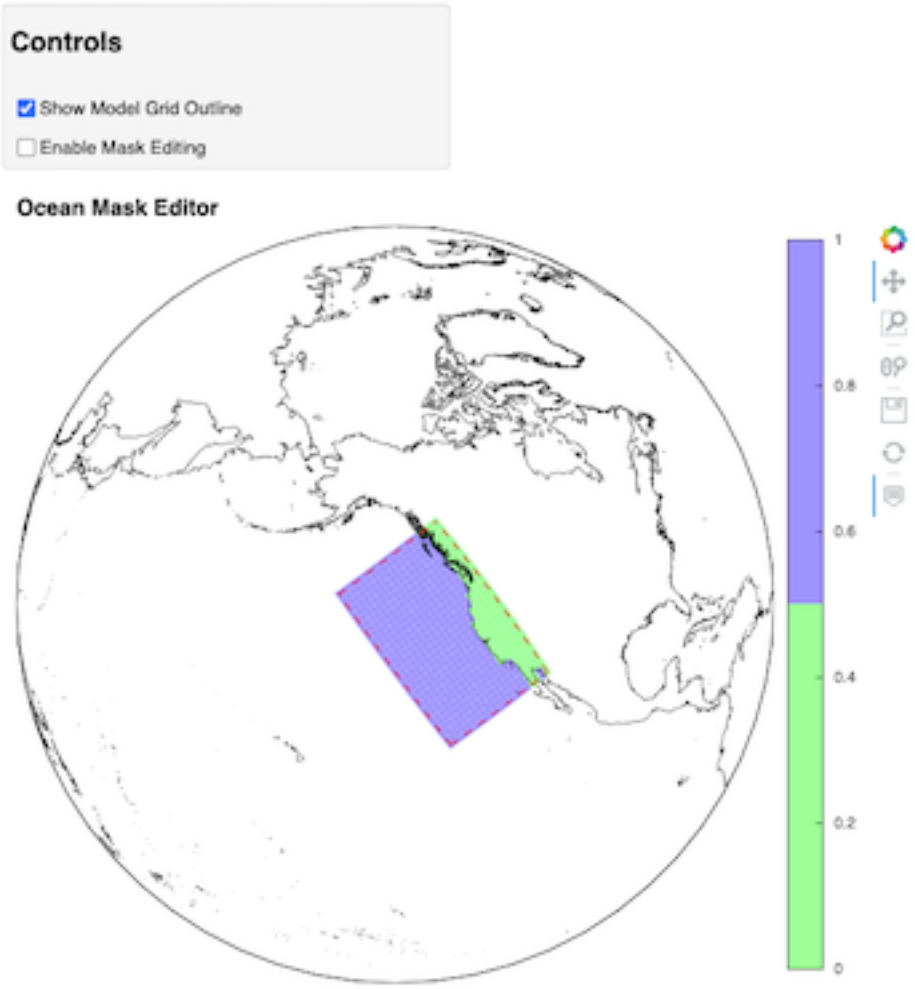
```
# Import needed python modules
import os, sys
from gridtools.gridutils import GridUtils
from gridtools.app import maskEditor
import cartopy.crs as ccrs

# Set a map projection for the mask editor
```

(continues on next page)

(continued from previous page)

```
map_crs = ccrs.Orthographic(-160, 90)

# Create the mask editor
appObj = maskEditor(crs=map_crs, ds=grd.grid['mask'])
app = appObj.createMaskEditorApp()
display(app)
```

The map editor should display after the cell is run.



For more information, see "*Edit a grid mask using Jupyter*".

Troubleshooting

If the application does not show, the jupyter lab session may need to be restarted. Errors in the gridtools library also may prevent the application from showing.

3.2.2 Pylab Mask Editor

The pylab mask editor is a port of the ROMS grid mask editor. This editor can be started in two ways.

The editor will require **X** windows to be operational and properly forwarded if starting the application on a remote system.

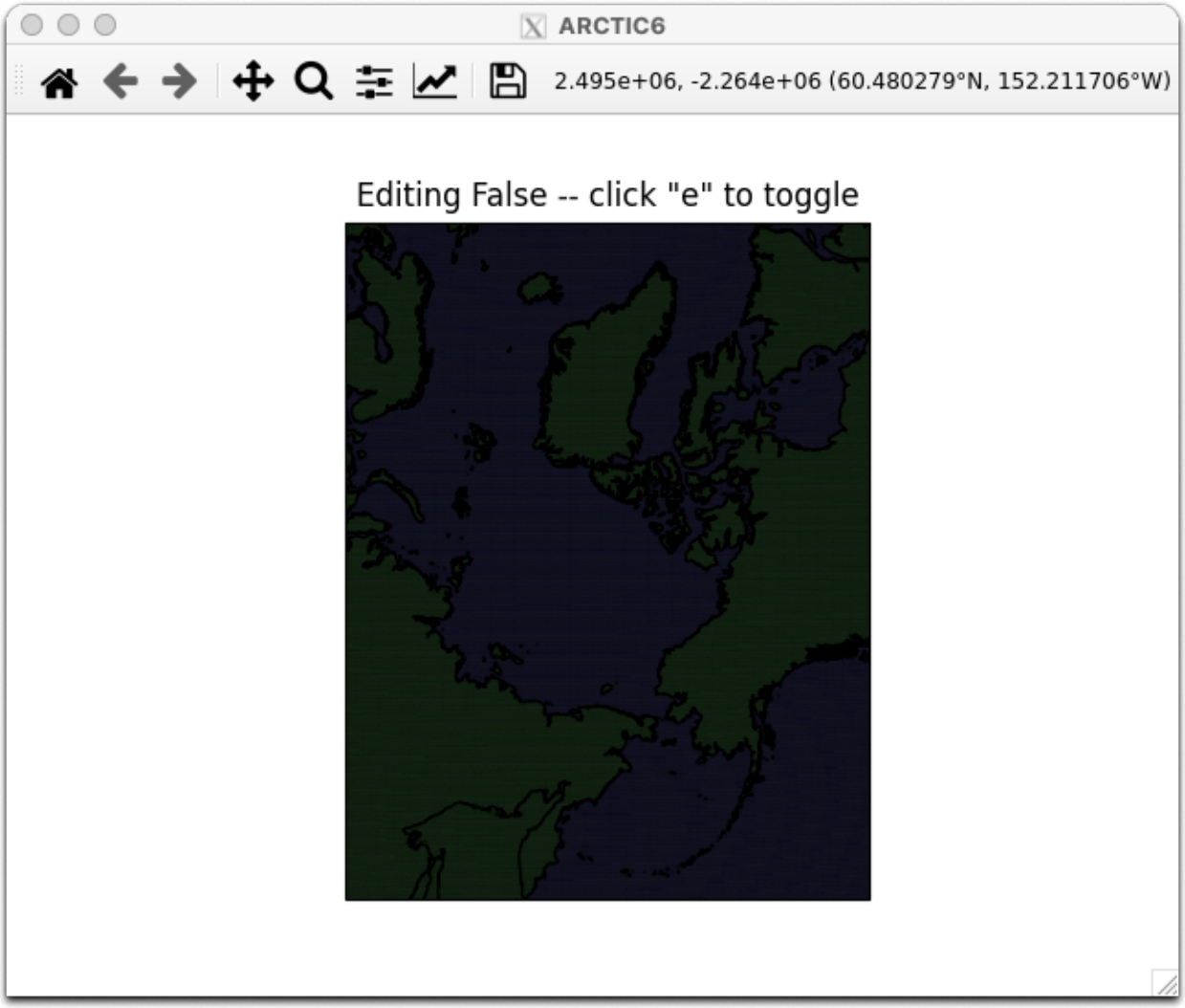
ipython

At a command prompt, start **ipython** with **pylab**.

```
% ipython --pylab

Python 3.7.10 | packaged by conda-forge | (default, Feb 19 2021, 16:07:37)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.24.1 -- An enhanced Interactive Python. Type '?' for help.
Using matplotlib backend: Qt5Agg
```

Then, type or copy and paste code into ipython to launch the editor. Changes will be retained in memory. At the conclusion of editing, use additional python commands to write out the new grid mask.

```
In [1]: import os, sys
   ...: from gridtools.gridutils import GridUtils
   ...: from gridtools.grids import roms
   ...: from pyproj import Proj
   ...: import cartopy.crs as ccrs
   ...:
   ...: # Set a place to write files
   ...: os.environ["ROMS_GRIDID_FILE"] = "/import/AKWATERS/jrcermakiii/configs/ROMS/
↪gridid.txt"
   ...: wrkDir = '/import/AKWATERS/jrcermakiii/configs/zOutput'
   ...: inputDir = os.path.join(wrkDir, 'INPUT')
   ...:
   ...: grd = GridUtils()
   ...:
   ...: romsObj = roms.ROMS()
   ...: romsGrd = romsObj.get_ROMS_grid('ARCTIC6')
   ...:
   ...: map_proj = ccrs.Stereographic(central_latitude=90.0, central_longitude=160.0)
   ...:
   ...: plotObj = romsObj.edit_mask_mesh(romsGrd.hgrid, proj=map_proj)
```




```
In [2]: #...post-editing commands...
```

jupyter

Using jupyter, use the magic function `%pylab` to activate pylab interactive support.

[]:

```
# Enable pylab
%pylab
```

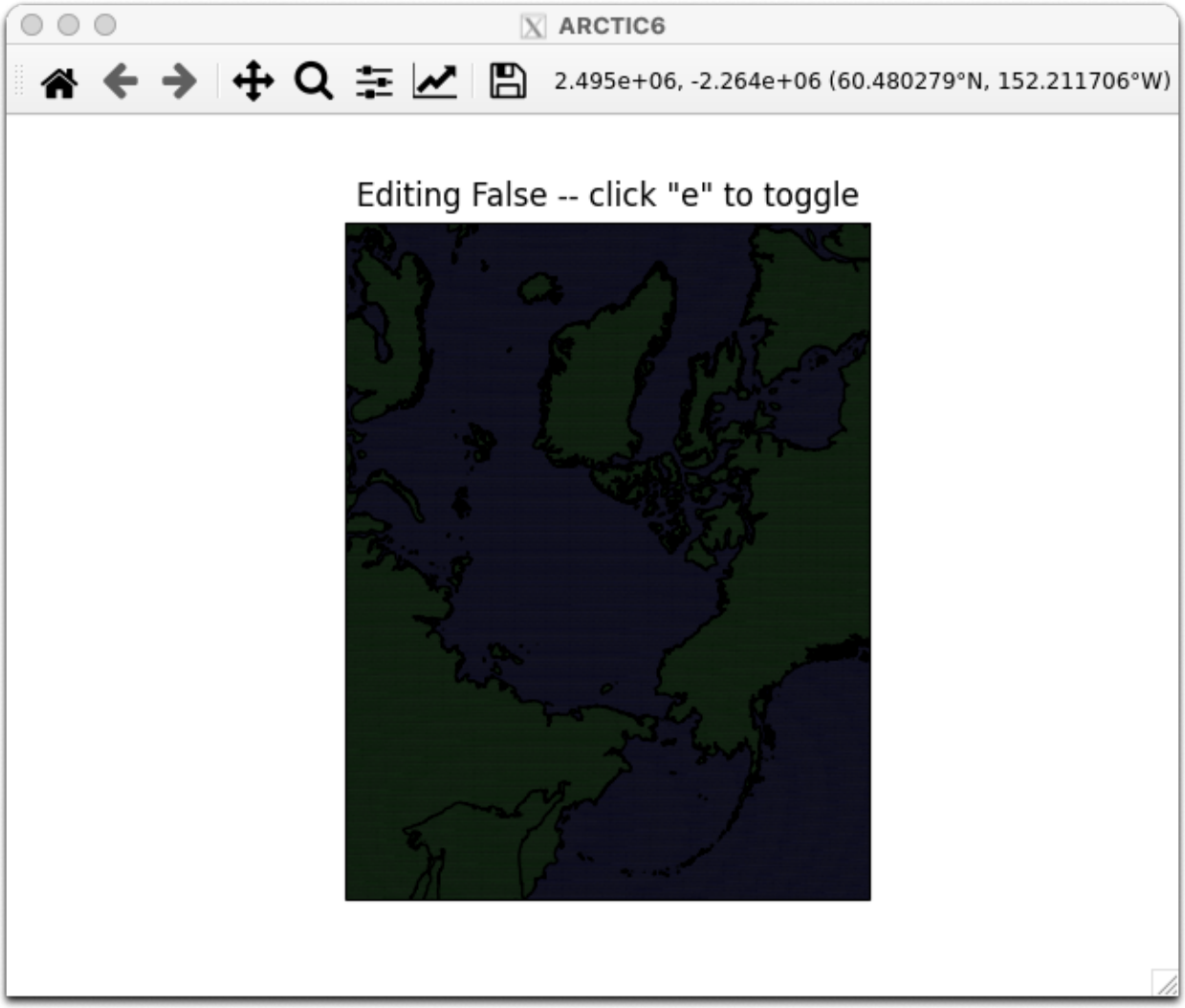
[]:

```
# Load a ROMS grid
import os
from gridtools.grids import roms
import cartopy.crs as ccrs
import xarray as xr
```

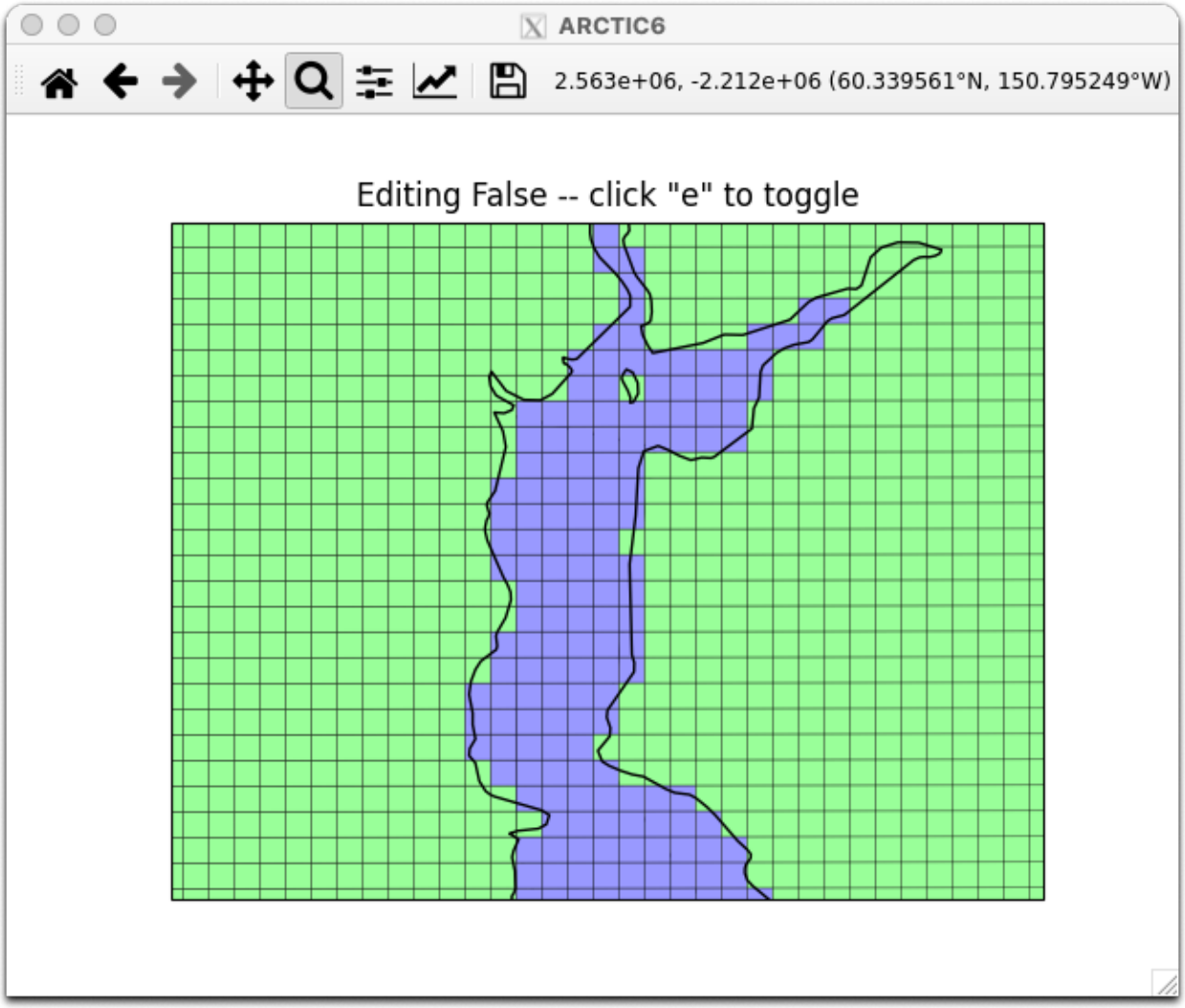
[]:

```
# Define a map projection
map_proj = ccrs.Stereographic(central_latitude=90.0, central_longitude=160.0)

# Use the gridid.txt file
os.environ["ROMS_GRIDID_FILE"] = "/import/AKWATERS/jrcermakiii/configs/ROMS/gridid.txt"

romsObj = roms.ROMS()
romsGrd = romsObj.get_ROMS_grid('ARCTIC6')

# Start the pylab editor
plotObj = romsObj.edit_mask_mesh(romsGrd.hgrid, proj=map_proj)
```

Here is the pylab mask editor application after startup:



Here is a zoomed in view of the pylab mask editor:


ARCTIC6
2.563e+06, -2.212e+06 (60.339561°N, 150.795249°W)
Editing False -- click "e" to toggle

CHAPTER

FOUR

GUIDES

The guides in this section explore additional topics beyond the scope of the gridtools library.

4.1 Binder

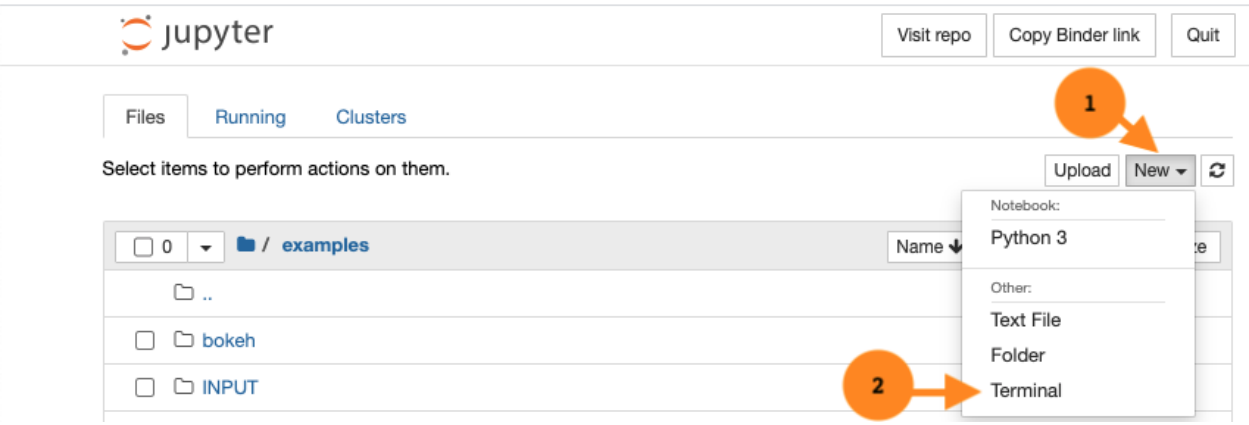
The binder website allows experimentation with python notebooks placed on github. Links to start a gridtools binder cloud service can be found on the github repository site.

Note: The binder cloud service, if not actively used, will unexpectedly terminate. **All data will be lost!** It is important to save any original information efficiently. Any grids or datasets will need to be reloaded or regenerated before they can be used again on a new cloud service session.

4.1.1 GEBCO

Downloading the GEBCO bathymetry dataset to binder can be accomplished through the terminal interface.

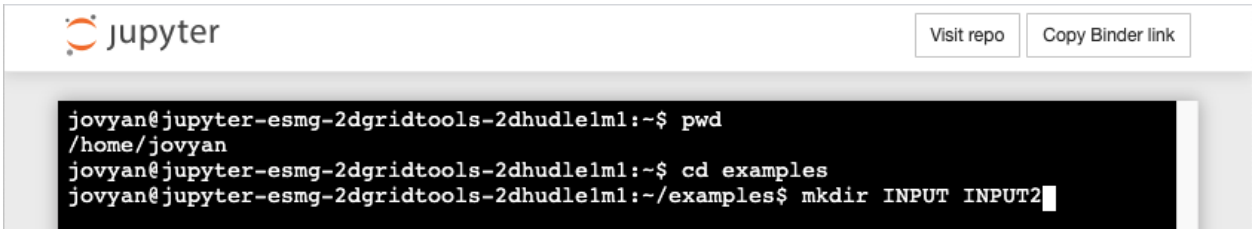
Launch a terminal from jupyter.



Change directory to the `examples` directory. If running the `NewGridMOM6.ipynb` example, then also create `INPUT` and `INPUT2` directories.

```
$ cd examples
$ mkdir INPUT INPUT2
```

This is what the terminal session should look similar to:



Download GEBCO 2020 from the GEBCO website.

```
$ wget "https://www.bodc.ac.uk/data/open_download/gebco/gebco_2020/zip/"
```

Note: This download will take approximately 20 minutes.

Rename `index.html` to `GEBCO.zip` and unzip.

```
$ mv index.html GEBCO.zip
$ unzip GEBCO.zip
```

This should finally unpack the neede `GEBCO_2020.nc` file which is now available in the `examples` directory. Switch back to the python notebook and change directory entries to `.` or reference the `GEBCO_2020.nc` file directly.

For the `NewGridMOM6.ipynb` notebook, these lines will change from:

```
# Source of GEBCO 2020 topographic grid
highResTopographyFile = "/import/AKWATERS/jrcermakiii/bathy/gebco/GEBCO_2020.nc"

# Write current model grid files
wrkDir = "/home/cermak/workdir/configs/zOutput"
inputDir = os.path.join(wrkDir, "INPUT")
input2Dir = os.path.join(wrkDir, "INPUT2")
```

to:

```
# Source of GEBCO 2020 topographic grid
highResTopographyFile = "GEBCO_2020.nc"

# Write current model grid files
wrkDir = "."
inputDir = os.path.join(wrkDir, "INPUT")
input2Dir = os.path.join(wrkDir, "INPUT2")
```

Note: The code above is in two separate notebook cells!

4.2 FRE-NCtools

4.2.1 Comparison of grid generation methods between gridtools and FRE-NCtools

This guide demonstrates two things:

- the creation of two similary constructed MOM6 model grids using gridtools and FRE-NCtools
- the creation of *ocean_topog.nc* using available functions in the gridtools library

Note: The FRE-NCtools software can be downloaded from the FRE-NCtools github website.

Grid Generation

In this section, a MOM6 model grid is created using the gridtools library and then the FRE-NCtools software.

In both examples, the MOM6 model grid is in the Mercator projection. The regular grid is 1.0 degree x 1.0 degree. The representation of this grid in the *ocean_hgrid.nc* file is typically a *supergrid*. The resolution of this grid is half the distance which is 0.5 degrees x 0.5 degrees.

The MOM6 model grid extent is from 220 East (or -140 West) to 240 East (or -120 West) and 25 to 55 North. The number of *supergrid* cells is 40 in the longitude or *j*-direction and 60 in the latitude or *i*-direction.

gridtools

A python script was written to demonstrate the creation of the MOM6 model grid using the gridtools python library. This example is currently named: mkGridsExample03.py

A MOM6 model grid is created by specifing grid parameters with respect to the regular grid. The grid center is specified and the number of grid cells evenly distributed along the *i* and *j*-direction. It is possible for a regular grid cell to become centered over the grid center in either the *i* and *j*-direction. The *supergrid* will always have an odd number of vertices which represents the number of regular grid points times two plus one.

A detailed description of the representation of discrete horizontal and vertical grids can be found in the MOM6 User Manual.

Grid parameters::

- **gridResolution**: 1.0 degree; the distance of the regular grid cell in both *i* and *j*-direction.
- **dx**: 20; the number of grid cells to use along the longitude or *j*-direction.
- **dy**: 30; the number of grid cells to use along the latitude or *i*-direction.

Specification of the grid center for longitude is specified in the total number of degrees East from the Prime Meridian.

Grid parameters::

- **centerX**: 230.0; 230.0 degrees East of the Prime Meridian
- **centerY**: 40.0; 40.0 degrees North

Once the grid paramters are set, a call to `makeGrid()` will result in construction of a regular grid that is 20x30 and a *supergrid* of 41x61 points that form vertices and pass through the center of the grid points.

FRE-NCtools

A python script was written to demonstrate the creation of the MOM6 model grid using the FRE-NCtools software package. This example is currently named: mkGridsExample03FRE.py

A MOM6 model grid is created by specifying grid parameters with respect to the *supergrid*.

The executable, *make_hgrid*, is used to create the MOM6 model grid. The program will create an output filename called *horizontal_grid.nc*.

The Mercator grid projection is specified by the program arguent *–grid_type* using *regular_lonlat_grid*.

The next two arguments specifies the number of x (or *j*) and y (or *i*) boundaries in the MOM6 model grid. Since this is a rectangle, there are two boundaries in each direction. The extents of the rectangle are specified in the next two arguments.

Grid parameter arguments::

- **–nxbnds** 2
- **–nybnds** 2
- **–xbnds** -140,-120
- **–ybnds** 25,55

Since longitude has a length of 20 degrees and the desired distance for the *supergrid* is 0.5 degrees, the number of points along the longitude is 40. Since latitude has a length of 30 degrees, the number of points needed to create 0.5 degree spacing is 60 points.

Grid parameters arguments::

- **–nlon** 40
- **–nlat** 60

This creates a regular grid that 20x30 grid points. The *supergrid* will have 41x61 points that make up the grid vertices that form the edges of the regular grid and pass through the grid centers.

The full command pulling all the above arguments togther is:

```
make_hgrid --grid_type regular_lonlat_grid --nxbnds 2 --nybnds 2\
    --xbnds -140,-120 --ybnds 25,55 --nlon 40 --nlat 60
```

CHAPTER

FIVE

API

5.1 app module

Gridtools library applications.

App()

maskEditor()

maskEditorPylab() This class is based on code from [HHT+21].

class gridtools.app.**App**(*grd=None*)

Bases: object

clearInformationWindow(*event*)

debugLevelCallback(*event*)

deleteLogfile(*event*)

This function is called as a result of pushing the "Erase logfile" button in the application. This places a call into GridTools.deleteLogfile(filename).

downloadNetCDF()

errorFigure()

Create Blank Plot to signal plotting failure. This signals a problem within the user specifications in relation to code capabilities.

errorNoGridFigure()

Creates a plot for the scenario where the user has not specified the grid before making a plot

initializeDashboard()

initializePlot()

Plot the initial image upon loading up the application. This is developed to differentiate between plot failure and the first plot.

initializeTabs()

initializeWidgets()

loadLocalGrid(*event*)

loadRemoteGrid(*event*)

logEnableCallback(*event*)

logLevelCallback(*event*)

make_grid(*event*)

make_plot()

plotRefresh(*event*)

saveRemoteGrid(*event*)
Attempt to save grid to remote filesystem using last known grid filename.

showManual()

updateDataView()

updateFilename(*newFilename*)

class gridtools.app.**maskEditor**(*ds=None*, *grd=None*, ***kwargs*)
Bases: object

This class helps launch the jupyter version of the ocean mask editor.

createMaskEditorApp()

getGridSubset(*lClickY*, *lClickX*, *grd*)

great_circle(*lon1*, *lat1*, *lon2*, *lat2*)

plotMap(*x*, *y*)

class gridtools.app.**maskEditorPylab**(*grd=None*)
Bases: object

This class helps launch the pylab version of the ocean mask editor. A jupyter cell with *%pylab* should preceed usage of this class.

Example:

[1]: import os from gridtools.grids import roms import cartopy.crs as crs import xarray as xr

Enable pylab %pylab

[2]: # Define a map projection map = ccrs.Stereographic(central_latitude=90.0, central_longitude=160.0)

[3]: # Load a ROMS grid

Use the gridid.txt file os.environ["PYROMS_GRIDID_FILE"] = "/import/AKWATERS/jrcermakiii/configs/Arctic6/roms/gridid.txt"

romsObj = roms.ROMS() romsGrd = romsObj.get_ROMS_grid('ARCTIC6')

[4]: # Start the pylab editor plotObj = romsObj.edit_mask_mesh(romsGrd.hgrid, proj=map)

[5]: # When ready to save the edited grid, uncomment the next line and run this cell # romsObj.write_ROMS_grid(romsGrd, filename='grid_py.nc')

Note: This currently only operates on ROMS grids. This method is extremely slow due to the map refresh method.

5.2 bathyutils module

gridtools.bathyutils.**applyExistingLandmask**(*grd*, *dsData*, *dsVariable*, *maskFile*, *maskVariable*, ***kwargs*)

Modify a given bathymetry using a specified land mask.

Parameters

- **grd** (GridUtils) – class object
- **dsData** (*xarray*) – data source data object
- **dsVariable** (*string*) – data source variable name
- **maskFile** (*string*) – filename
- **maskVariable** (*string*) – variable name in maskFile
- ****kwargs** – See below

Keyword arguments:

- *epsilon* (`float`) – When a point is declared an ocean point, if the depth is shallower than the masking depth, the depth is set to the minimum depth. If the masking depth is undefined or equal to the minimum depth, the new depth is set deeper by epsilon to avoid becoming masked as land. Default: 1.0e-14
- *MINIMUM_DEPTH* (`float`) – Minimum ocean depth. Default: 0.0
- *MASKING_DEPTH* (`float`) – Ocean depths equal or shallower than the masking depth are masked as land. Default: undefined
- *MAXIMUM_DEPTH* (`float`) – Maximum depth of the ocean. Defaults to maximum depth from data source if not specified.
- *TOPO_EDITS_FILE* (`string`) – Changed mask points in the MOM6 zEdits format will be recorded to the specified filename. (Unimplemented)

Note: For ocean points, if a depth is shallower than the MINIMUM_DEPTH but deeper than the MASKING_DEPTH, the depth will be set to the MINIMUM_DEPTH.

Ocean points that are to become land will be set to the MASKING_DEPTH. If MASKING_DEPTH is not defined, MINIMUM_DEPTH is used as the masking depth.

---

gridtools.bathyutils.**applyExistingOceanmask**(*grd*, *dsData*, *dsVariable*, *maskFile*, *maskVariable*, ***kwargs*)

Modify a given bathymetry using a specified ocean mask.

Parameters

- **grd** (GridUtils) – class object
- **dsData** (*xarray*) – data source data object
- **dsVariable** (*string*) – data source variable name
- **maskFile** (*string*) – filename
- **maskVariable** (*string*) – variable name in maskFile
- ****kwargs** – See below

Keyword arguments:

- *epsilon* (`float`) – When a point is declared an ocean point, if the depth is shallower than the masking depth, the depth is set to the minimum depth. If the masking depth is undefined or equal to the minimum depth, the new depth is set deeper by epsilon to avoid becoming masked as land. Default: 1.0e-14
- *MINIMUM_DEPTH* (`float`) – Minimum ocean depth. Default: 0.0
- *MASKING_DEPTH* (`float`) – Ocean depths equal or shallower than the masking depth are masked as land. Default: undefined
- *MAXIMUM_DEPTH* (`float`) – Maximum depth of the ocean. Defaults to maximum depth from data source if not specified.
- *TOPO_EDITS_FILE* (`string`) – Changed mask points in the MOM6 zEdits format will be recorded to the specified filename. (Unimplemented)

Note: For ocean points, if a depth is shallower than the MINIMUM_DEPTH but deeper than the MASKING_DEPTH, the depth will be set to the MINIMUM_DEPTH.

Ocean points that are to become land will be set to the MASKING_DEPTH. If MASKING_DEPTH is not defined, MINIMUM_DEPTH is used as the masking depth.

---

gridtools.bathyutils.**break_array_to_blocks**(*a*, *xb=4*, *yb=1*, *useOverlap=False*, *useSupergrid=False*)

gridtools.bathyutils.**computeBathymetricRoughness**(*grd*, *dsName*, ***kwargs*)

This generates h2 and other variables and returns an xarray DataSet.

Parameters

- **grd** (GridUtils) – class object
- **dsName** (*string*) – data source name
- ****kwargs** – See below

Keyword arguments:

- *maxMb* (`int`) – Memory limit for grid refinements. Default: 8000.0
- *h2Name* (`string`) – The computed bathymetric roughness grid name. Default: h2
- *depthName* (`string`) – The bathymetry grid name to use from data source. Default: depth
- *gridPoint* (`string`) – Grid placement of bathymetric roughness values. See below. Default: h
- *auxVariables* (`list`) – Specify additional variables to include with bathymetric roughness. See below. Default: []
- *superGrid* (`boolean`) – If `True`, the bathymetric roughness grid returned is a supergrid. Otherwise, the ocean roughness is the same size as the current grid. Default: False
- *useClipping* (`boolean`) – Use `True` if the current grid is periodic and should be clipped prior to computing the bathymetric roughness. Defualt: False
- *useFixByOverlapQHGridShift* (`boolean`) – When using a regular grid, use overlapping grid technique to fill in partition boundaries. See IMPLEMENTATION NOTES below. Default: True
- *useQHGridShift* (`boolean`) – For a regular grid, use the Q point values as the H values to fill missing points.
- *useOverlap* (`boolean`) – Use overlapping grid technique to fill in partition boundaries. The outer column and row will still be missing.

- *extendedGrid* (boolean) – If True, the grid provided by this routine has been extended and should use the overlap technique on h-points only and not q-points which are then shifted back to h-points. See IMPLEMENTATION NOTES below. Default: False

This routine is based on a paper by Adcroft [Adc03] and python code from *OMtopogen/create_topog_refinedSampling.py* [Zad20b].

Note:

maxMB The memory limit for successive grid refinements. This should be maximized for the memory footprint of the compute node. If the program crashes, use lower amounts of memory.

auxVariables h_std, h_min, h_max, and height variables. Ask for one or more additional variables by python list [] to auxVariables. Default is an empty list: []

gridPoint By default, this generates bathymetric roughness grids on Arakawa C h-points but may be specified via gridPoint.

For gridPoint, a diagram of a grid cells is:

```
q-----v-----q
|           |
u     h     u
|           |
q-----v-----q
```

Valid values for gridPoint are:

- 'h' h-point; grid center
- 'q' q-point; grid corners
- 'uv' u- and v-point; grid faces

superGrid If you want all supergrid points, set superGrid=True. Setting tells this routine to use the supergrid when calculating the bathymetric roughness. This will require more RAM. Default: False

---

IMPLEMENTATION NOTES:

- For the supergrid, four zero bands along longitude are returned representing the four grid partitions. This needs to be fixed in the future.
- **useFixByOverlapQHGridShift** by default is True. Roughness (h2) is diagnosed on the q-points and shifted by 1/2 a grid cell back to the h-points. Accuracy of the roughness and other resultant variables are off by a 1/2 grid cell. This only works for the regular grid, not the supergrid. If the grid is extended, setting **extendedGrid** to True, tells this routine to attempt to diagnose h2 on the h-points without shifting the grid. The extended grid should be extended by two grid points on the supergrid.
- Support for 'q' and 'uv' grid points are not supported.
- If the program is running out of memory, reduce the maxMb value. This reduces the available memory footprint available to this routine. This will also reduce the number of available refinements against the bathymetry data source.

gridtools.bathyutils.**do_block**(*grd*, *part*, *lon*, *lat*, *topo_lons*, *topo_lats*, *topo_elvs*, *max_mb=500*)

gridtools.bathyutils.**extend_by_zeros**(*x*, *shape*)

gridtools.bathyutils.**get_indices1D**(*lon_grid*, *lat_grid*, *x*, *y*)
This function returns the j,i indices for the grid point closest to the input lon,lat coordinates.

gridtools.bathyutils.**get_indices1D_old**(*lon_grid*, *lat_grid*, *x*, *y*)
This function returns the j,i indices for the grid point closest to the input lon,lat coordinates.

gridtools.bathyutils.**get_indices2D**(*lon_grid*, *lat_grid*, *x*, *y*)
This function returns the j,i indices for the grid point closest to the input lon,lat coordinates.

gridtools.bathyutils.**ice9**(*grd*, ***kwargs*)
This is a implementation of the ice-9 algorithm for filling in disconnected ocean bodies (lakes) for a given ocean grid. Be sure to specify anticipated MINIMUM_DEPTH, MASKING_DEPTH and MAXIMUM_DEPTH that will be used for the model run.

Parameters

- **grd** (GridUtils) – class object
- ****kwargs** – See below

Keyword arguments:

- *ocean_seeds* ([(int, int)]) – Provide ice-9 algorithm with one or more (j,i) ocean body grid points to designate as areas that should be treated as areas of continuous ocean points. If more than one seed is given, all the discovered wet points will be merged together to form the final grid minus any detached ocean points (lakes). NOTE: Only the first seed is used! Future releases will allow multiple seeds.
- *depth* (grid) – The depth grid to use for ice-9 algorithm. This should be the same size as the provided latitude and logitude points defining the grid. Values are positive for depth (water) and negative for height (land).
- *periodic* (boolean) – Tells the algorithm that the grid is periodic and should check wrap points. Default: False NOTE: Not implemented!
- *returnFields* (list) – List of fields to be returned in the grd object. Default: ['wetMask']
- *MINIMUM_DEPTH* (float) – minimum depth of ocean in meters. Default: 0.0
- *MASKING_DEPTH* (float) – masking depth of ocean in meters. Default: 0.0
- *MAXIMUM_DEPTH* (float) – maximum depth of ocean in meters. Default: -99999.0
- *zEdits* (boolean) – Utilize zEdits for the provided depth field. Store any updates to the depth field in zEdits as well. Default: False NOTE: Not implemented!

The ice-9 algorithm was first mentioned in a program written by Niki Zadeh in the *ocean_model_topog_generator* repository. [Zad20b]

Another reference to the ice-9 algorithm is metioned in the repository *regrid_runoff* by Alistair Adcroft. [Adc20].

gridtools.bathyutils.**mdist**(*x1*, *x2*)
Returns positive distance modulo 360.

gridtools.bathyutils.**refine_by_repeat**(*x*, *rf*)

gridtools.bathyutils.**undo_break_array_to_blocks**(*a*, *xb=4*, *yb=1*, *useOverlap=False*, *extendedGrid=False*, *useSupergrid=False*, *useQHGridShift=False*)

5.3 datasource module

class gridtools.datasource.**DataSource**
Bases: object

addDataSource(*newDataSource*, *delete=False*)
Add a new dataset to the catalog. This will not delete an exisiting dataset unless the delete flag is True. A complete description must be provided as the named dataset will completely replace any existing map if the delete flag is True.

cleanCatalog(*catalog*)
This removes any empty/null values from the catalog prior to saving or printing the values.

clearCatalog()
This clears the current catalog.

loadCatalog(*inFile*, *append=True*, *overwrite=False*)
Loads catalog entries from a json or yaml file. Default behavior is only append entries (append=True). Any duplicate entries are ignored (overwrite=False). If you wish to only overwrite existing entries use append=False and overwrite=True. Using append=False and overwrite=False will do nothing. To append and/or replace entries use append=True and overwrite=True.

saveCatalog(*outFile*)
Save currently stored catalog to a file in the chosen format by the extension. Supported extensions are: json, yaml

5.4 fileutils module

gridtools.fileutils.**resolveDataSource**(*grd*, *dsName*)
This returns a final filename to gridtools to handle cases where the user may or may not provide a file:// prefix to filenames.

This routine does not check if the resolved filename exists.

This may also be a data source via ds:// in which we translate to a file:// spec.

http://, https:// and gql:// are not supported at this time.

5.5 gridutils module

class gridtools.gridutils.**GridUtils**(*app={}*)
Bases: object

addDataSource(*dataSource*, *delete=False*)
Add a data source to the catalog. See: datasource.addDataSource()

addMessage(*msg*)
Append new message to message buffer.

adjustExternallLoggers()
This adjusts some noisy loggers from other python modules.

app()
By calling this function, the user is requesting the application functionality of GridUtils(). return the dashboard, but GridUtils() also has an internal pointer to the application.

application(*app={}*)
Convienence function to attach application items to GridUtils() so it can update certain portions of the application:

```
app = {
    'messages': panel.widget.TextBox     # Generally a pointer to a panel␣
↪widget for display of text
    'defaultFigureSize': (8,6)           # Default figure size to return from␣
↪matplotlib
    'usePaneMatplotlib': True/False      # Instructs GridUtils to use panel.
↪pane.Matplotlib for plot objects
}
```

applyEvalMap(*dsName*, *dsData*)
Apply constructed equations through python eval() to manipulate data source fields. Data source catalog entries must be prefixed with ds:. If GEBCO is defined as a data source in the catalog, use: ds:GEBCO. All catalog entries start with a slash.

applyExistingLandmask(*dsData*, *dsVariable*, *maskFile*, *maskVariable*, ***kwargs*)
This modifies a given bathymetry using an existing land mask. See *applyExistingLandmask()*.

applyExistingOceanmask(*dsData*, *dsVariable*, *maskFile*, *maskVariable*, ***kwargs*)
This modifies a given bathymetry using an existing ocean mask. See *applyExistingOceanmask()*.

cart2pol(*x*, *y*, *z*)
Transform a point on globe from Cartesian (x,y,z) to polar coordinates.

checkAvailableVariables(*dsData*, *varList*)
Check for available variables in a data source. If any variable is missing, issue a warning and return False. If all variables are available, return True.

checkGridMetadata(*kwargs*, *varKey*, *defaultValue*, *append=False*)
Update or overwrite global metadata values based on keyword arguments.

Keyword arguments

- *append* (`boolean`) – Updated metadata is appended to any existing metadata. If the attribute does not exist, it is added.

clearGrid()
Call this when you want to erase the current grid. This also clobbers any current grid and plot parameters. Do not call this method between plots of the same grid in different projections.

clearGridParameters()
Clear grid parameters. This does not erase any grid data.

clearMessage()
This clears the message buffer of messages.

closeGrid()
Closes and open dataset file pointer.

computeBathymetricRoughness(*dsName*, ***kwargs*)
This generates h2 and other fields. See: *gridtools.bathyutils.computeBathymetricRoughness()*

computeGridMetricsCartesian()
Compute MOM6 grid metrics: angle_dx, dx, dy and area for a grid in cartesian coordinates.

computeGridMetricsSpherical(***kwargs*)
Compute MOM6 grid metrics: angle_dx, dx, dy and area for a grid in spherical coordinates.

Keyword arguments

- *history* (`string`) – optional message to append to the global `history` attribute. A default message is provided if one is not specified.

`convertGrid`(*target*, ***kwargs*)

Convert current grid to another grid type.

Parameters `target` (*string*) – name of new grid format

Returns none

Return type none

Supported grid conversions:

SOURCE	TARGET	CODE CITATIONS
ROMS	MOM6	convert_ROMS_grid_to_MOM6.py [IAH20]

Keyword arguments:

- *writeTopography* (`boolean`) – set True to write topographic grid to a file. Default: False
- *topographyFilename* (`string`) – filename used to write topographic grid. Default: "ocean_topog.nc"
- *writeMosaic* (`boolean`) – set True to write the mosaic file. Default: False
- *mosaicFilename* (`string`) – filename for mosaic file. Default: "ocean_mosaic.nc"
- *oceanGridFilename* (`string`) – filename for ocean grid file. Default: "ocean_hgrid.nc"
- *writeLandmask* (`boolean`) – set True to write land mask file. Default: False
- *landmaskFilename* (`string`) – filename used to write the land mask. Default: "land_mask.nc"
- *writeOceanmask* (`boolean`) – set True to write ocean mask file. Default: False
- *oceanmaskFilename* (`string`) – filename used to write the ocean mask. Default: "ocean_mask.nc"
- *tileName* (`string`) – name to assign to the solo tile. Default: "tile1"
- *MINIMUM_DEPTH* (`float`) – minimum depth of ocean in meters. Default: 0.0
- *MASKING_DEPTH* (`float`) – masking depth of ocean in meters. Default: 0.0
- *MAXIMUM_DEPTH* (`float`) – maximum depth of ocean in meters. Default: -99999.0
- *writeCouplerMosaic* (`boolean`) – set False to skip creation of coupler mosaic file. Default: False
- *couplerMosaicFilename* (`string`) – set False to skip creation of coupler mosaic file. Default: "mosaic.nc"
- *writeExchangeGrids* (`boolean`) – set False to skip creation of exchange grids. Default: False
- *overwrite* (`boolean`) – set True to overwrite existing files. Default: False
- *inputDirectory* (`string`) – absolute or relative path to write model input files. Default: "INPUT"
- *relativeToINPUTDir* (`string`) – absolute or relative path for mosaic files to the INPUT directory. Default: "./"

Keyword arguments (ROMS):

- *topographyVariable* (`string`) – grid name for ROMS topography containing depths. Some ROMS grids also contain a *hraw* which might be useful. Default: h

Note: In the original ROMS to MOM6 conversion script, any land mask points are masked to a depth of zero(0). This version sets the depth to the *MASKING_DEPTH*. If the mask or clamping depth is known from the ROMS grid, please set *MASKING_DEPTH* appropriately or the land mask depths will be set to zero(0) which is the default.

convertToMathExpression(*sourceExpression*)
Convert a source expression to a python expression for evaluation.

Ex: "-[elevation]" => "-dsData['elevation']"

debugMsg(*msg*, *level=- 1*)
This function has a specific purpose to aid in debugging and activating pdb breakpoints. NOTE: pdb breakpoints tend not to work very well when running under the gridtools application. It tends to terminate the bokeh/tornado server.

The debug level can be zero(0) and you can forcibly add a break point in the code by using *debugMsg(msg, level=3)* anywhere in the code.

Note:

Currently defined debug levels: 0=off 1=log debugging messages 2=raise an exception 3=stop with a pdb breakpoint after logging message

deleteGridParameters(*gList*, *subKey=None*)
This deletes a given list of grid parameters.

deleteLogfile(*logFilename*)
Delete a log file. Logging must be off.

deletePlotParameters(*pList*, *subKey=None*)
This deletes a given list of plot parameters.

detachLoggingFromApplication()
Detach logging from application so messages are shown in the script and/or jupyter.

disableLogging()
Disable logging of messages to a file

enableLogging(*logFilename*)
Enable logging of messages to a file

extendGrid(*jStart*, *jEnd*, *iStart*, *iEnd*, *gridMethod='auto'*, *gridProj='auto'*)
Extend the current grid by jStart, jEnd, iStart, iEnd points using the specified method. The grid can be extended using 'spherical' space or 'latlon' space. If the method is 'auto', the routine tries to determine which to use. The grid is extended in all directions using the maximum of provided parameters and then clipped to the proper dimensions. Grids extended using 'spherical' space require the projection used when the grid was created to perform forward and reverse transformation of coordinates.

This function assumes an existing grid is present and that the variables x and y are longitude and latitude.

This function returns the extended grid.

Note: Since MOM6 grids are defined with a supergrid, to extend the regular grid one point in all directions, this routine should specify to extend the grid two points in all directions.

Grid Extension Technique:

```
This description shows the extension of a grid by one point.  This
also applies to any requested grid size.

Process

# Original grid (o); Points to fill (.)
#  . . . . .
#  . o o o .
#  . o o o .
#  . o o o .
#  . . . . .

# Step 1: Extend grid along j-direction
# New points shown by (A)
#  . . . . .
#  A o o o A
#  A o o o A
#  A o o o A
#  . . . . .
#

# Step 2: Extend grid along the i-direction
# New points shown by (B)
#  B B B B B
#  A o o o A
#  A o o o A
#  A o o o A
#  B B B B B

# Step 3: Clip grid back to requested size
# given by iStart, iEnd, jStart, jEnd.

# To prepend one column of points,
# please specify extendGrid(0,0,1,0).
# The grid returned should look like:
#  A o o o
#  A o o o
#  A o o o

# To prepend one column of points and
# append a row of points to the end,
# please specify extendGrid(0,1,1,0).
# The grid returned should look like:
#  B B B B
#  A o o o
#  A o o o
#  A o o o
```

extendGridDetectMethod()

This function looks at the grid metadata and searches for hints to see how the grid should be extended.

Sources probed for information:

- Global attribute: projection

- Variable 'tile' attribute: geometry

extendGridLatLon(*inputGrid*, *maxIncrease*)
This uniformly extends the input grid by maxIncrease points using latitude and longitude coordinates in degrees.

To increase the grid size we need maxIncrease points on either size (twice as big).

extendGridSpherical(*inputGrid*, *maxIncrease*, *gridProj*)
This uniformly extends the input grid by maxIncrease points using spherical coordinates in meters. This function requires a grid projection to accurately perform the forward and reverse transformation of grid points.

To increase the grid size we need maxIncrease points on either size (twice as big).

filterLogMessages(*record*)
This may not be needed after all.

findLineFromPoints(*ptsY*, *ptsX*, *nY*, *nX*)
Find the extension points at the end of given set of points. This routine assumes a nearly linear regularly spaced array of points is provided.

Returned are the new points on the given line.

([y1, y2], [x1, x2]) where (y1, x1) is the head of the line and (y2, x2) is the tail of the line.

NOTE: Number of points to extend should be the same nY = nX. If the points are not regularly spaced, extension of a line with a large number of points is not going to work very well.

formProjString(*param*)
Create a projection string from parameter attributes.

generateGridByRefinement(*dsName*, ***kwargs*)
Generates a grid from a data source using refinement regridding.

generate_latlon_mesh_centered(*lni*, *lnj*, *llon0*, *llen_lon*, *llat0*, *llen_lat*, ***kwargs*)
Generate a regular lat-lon grid

generate_regional_mercator(*cUnits*, *cX*, *cY*, *dx*, *dxU*, *dy*, *dyU*, *tilt*, *grX*, *grXU*, *grY*, *grYU*, *pD*, ***kwargs*)
Generate a regional mercator grid centered at (cX, cY) with spans of (dx, dy) with resolution (grX, grY) and tilted by angle tilt

generate_regional_spherical(*lon0*, *lon_span*, *lat0*, *lat_span*, *tilt*, *gRes*, *gMode*)
Generate a regional grid centered at (lon0,lat0) with spans of (lon_span,lat_span) and tilted by angle tilt

generate_regional_spherical_degrees(*cUnits*, *cX*, *cY*, *dx*, *dxU*, *dy*, *dyU*, *tilt*, *grX*, *grXU*, *grY*, *grYU*, *pD*, ***kwargs*)
Create a grid in the spherical projection using grid distances in degrees.

generate_regional_spherical_meters(*cUnits*, *cX*, *cY*, *dx*, *dxU*, *dy*, *dyU*, *tilt*, *grX*, *grXU*, *grY*, *grYU*, *pD*, ***kwargs*)
Create a grid in the spherical projection using grid distances in meters.

This function uses code that performs projection transformation of 2D grids. [Dus20]

generate_rotated_grid(*lon0*, *lon_span*, *lat0*, *lat_span*, *tilt*, *refine*)

getDebugLevel()
Get the current debug level for GridUtils(). See setDebugLevel() for available levels.

getGridParameter(*gkey*, *subKey=None*, *default=None*, *inform=True*)
Return the requested grid parameter or the default if none is available. The routine will emit a message by default. Use inform=False to suppress messages emitted by this function.

getLogLevel()
Get the current debug level for GridUtils(). See setLogLevel() for available levels.

getPlotParameter(*pkey*, *subKey=None*, *default=None*, *inform=True*)
Return the requested plot parameter or the default if none is available.

To access dictionary values in projection, use the subKey argument.

This function will emit an informational message when a default parameter is being used. Use inform=False to suppress the messages.

getRadius(*param*)
Return a radius based on projection string. If parsing the projection string fails, the radius from GRS80/WGS84 is used.

getVerboseLevel()
Get the current verbose level for GridUtils()

getVersion()
Return the version number of this library

getXYDist(*x1*, *y1*, *x2*, *y2*)
Compute distance between two points in x/y space.

ice9(***kwargs*)
This calls the ice-9 algorithm from bathyutils. See: *gridtools.bathyutils.ice9()*

isAvailable(*utilName*)
Check the availablilty of a runnable executable in the system PATH. Returns True if an executable appears to be available in the system PATH. Otherwise, return False.

makeGrid(*setFilename=None*)
Using supplied grid parameters, populate a grid in memory.

makeSoloMosaic(***kwargs*)
This replicates some of the processes of `make_solo_mosaic` from [GeophysicalFDLGFDLMSGroup21]. This routine is also based on code from [IAH20].

A grid must be created or read. This function has the following keyword arguments and default values. This is a MOM6 specific function to write out various files depending on arguments passed.

Keyword arguments:

- *topographyGrid* (`xarray`) – topographic grid to be used with the model grid. REQUIRED. Default: None
- *topographyFilename* (`string`) – filename used to write topographic grid. Default: "ocean_topog.nc"
- *mosaicFilename* (`string`) – filename for mosaic file. Default: "ocean_mosaic.nc"
- *oceanGridFilename* (`string`) – filename for ocean grid file. Default: "ocean_hgrid.nc"
- *writeLandmask* (`boolean`) – set True to write land mask file. Default: False
- *landmaskFilename* (`string`) – filename used to write the land mask. Default: "land_mask.nc"
- *writeOceanmask* (`boolean`) – set True to write ocean mask file. Default: False
- *oceanmaskFilename* (`string`) – filename used to write the ocean mask. Default: "ocean_mask.nc"
- *tileName* (`string`) – name to assign to the solo tile. Default: "tile1"
- *MINIMUM_DEPTH* (`float`) – minimum depth of ocean in meters. Default: 0.0
- *MASKING_DEPTH* (`float`) – masking depth of ocean in meters. Default: 0.0

- *MAXIMUM_DEPTH* (`float`) – maximum depth of ocean in meters. Default: -99999.0
- *writeCouplerMosaic* (`boolean`) – set False to skip creation of coupler mosaic file. Default: True
- *couplerMosaicFilename* (`string`) – set False to skip creation of coupler mosaic file. Default: "mosaic.nc"
- *writeExchangeGrids* (`boolean`) – set False to skip creation of exchange grids. Default: True
- *overwrite* (`boolean`) – set True to overwrite existing files. Default: False
- *inputDirectory* (`string`) – absolute or relative path to write model input files. Default: "INPUT"
- *relativeToINPUTDir* (`string`) – absolute or relative path for mosaic files to the INPUT directory. Default: "./"

Note: Integers stored in the mosaic tiles must be 32 bit integers. If 64 bit integers are used, the FMS coupler will die with invalid netcdf variable type. Using the defaults, this routine will write the following files to the `INPUT` directory with one tile named `tile1`:

- `mosaic.nc`
- `ocean_mosaic.nc`
- `ocean_topog.nc`
- `atmos_mosaic_tile1Xland_mosaic_tile1.nc`
- `atmos_mosaic_tile1Xocean_mosaic_tile1.nc`
- `land_mosaic_tile1Xocean_mosaic_tile1.nc`

If any of these files exist, the file will **NOT** be replaced and a warning will be issued.

For *oceanGridFile*, if the filename is not provided, the routine will attempt to use the name provided when the grid was read. The filename may need to be set if the grid was just constructed using the library.

---

mesh_plot(*lon*, *lat*, *lon0=0.0*, *lat0=90.0*)
Plot a given mesh with a perspective centered at (lon0,lat0)

newFigure(*figsize=None*, *dpi=None*)
Establish a new matplotlib figure.

openDataset(*dsName*, ***kwargs*)
Open a dataset using the data source catalog or url that can point to a raw dataset using a prefix file: in the data source name. The url can be an OpenDAP dataset: e.g. https://opendap.jpl.nasa.gov/opendap/allData/ghrsst/data/L4/GLOB/NCDC/AVHRR_AMSR_OI/2011/001/20110101-NCDC-L4LRblend-GLOB-v01-fv02_0-AVHRR_AMSR_OI.nc.bz2

Keyword arguments

- *chunks* (`int, tuple of int or mapping of hashable to int`) – xarray chunk description.
- *gridid* (`ROMS model grid ID`) – Model grid identification using the gridid.txt file. The environment variable *ROMS_GRIDID_FILE* must be set.

openGrid(*inputUrl*, ***kwargs*)
Open a grid file. This creates and open file pointer which is local to the gridtools object.

Specify with file: or OpenDAP (http://, https://) or ds:.

To access the opened grid, use: *obj.xrDS*. This grid now needs to be formally read by *readGrid()* and the grid will be available to *obj.grid* and *obj.nativeGrid*.

plotGrid(***kwargs*)

Perform a plot operation. This function supports plotting a variable from a data source or the model grid that was loaded or created. A plot may contain both.

Returns Returns a tuple of matplotlib objects (figure, axes)

Return type tuple

In general, a projection is necessary to plot a variable or model grid.

Example

```
>>> grd = gridUtils()
>>> grd.setPlotParameters(
        {
            ...other grid options...,
            'projection': {
                'name': 'Mercator',
                ...other projection options...,
            },
        })
>>> grd.plotGrid()
```

Keyword arguments:

- *showModelGrid* (`boolean`) – set False to hide the model grid. Default: True
- *plotVariables* (`dict()`) – one or more variables and plot parameters. Default: None
- *showGridPoints* (`list()`) – list of grid points to show on the plot. The list contains tuples of *(j,i)*, *(j,i,c)* or *(j,i,c,s)*. The default color (c) is *red* with a size (s) of 5. To change just the size of the point, the color must also be specified. NOTE: For MOM6 grids, the (j,i) refer to points on the supergrid. Points on the regular grid should be multiples of two.

Keyword arguments for plotVariables:

- *vals* (`xarray`) – one dataset, variable or grid of information.
- *cmap* (`str or Colormap`) – A Colormap instance or registered colormap name. Default: `rcParams["image.cmap"]` or `viridis`
- *norm* (`Normalize`) – A Normalize instance. Default: The data range is mapped to the colorbar range using linear scaling.
- *levels* (`list()`) – Discrete levels for plotting. Default: None
- *cbar_kwargs* (`dict()`) – Keyword arguments that are applied to the colorbar. See: matplotlib.figure.Figure.colorbar() Default: dict()

Useful information on plot shading for grid cell or centered over a grid point. Useful example for adjusting the colorbar and using a different colormap.

pol2cart(*lam*, *phi*)

Transform a point on globe from Polar (lam,phi) to Cartesian coordinates.

printMsg(*msg*, *level=20*)

The verboseLevel and msgLogLevel can be set separately to any level. If this is attached to a panel application with a message box, the output is sent to that object. Messages omitting the level argument will default to INFO.

readGrid(***kwargs*)

Read a grid. This is more of a convenience function for applications that need to pass grids to gridtools instead of using the openGrid function.

This can be generalized to work with "other" grids if we desired? (ROMS, HyCOM, etc). This routine does not verify the read grid vs. type of grid specified.

Note: A copy of the native grid native structure can be found by using *obj.nativeGrid*. The *obj.grid* variable may be modified by the gridtools library.

---

regridTopo(*dsData*, *gridGeoLoc='corner'*, *topoVarName='elevation'*, *coarsenInt=10*, *method='conservative'*, *superGrid=True*, *periodic=True*, *gridDimX=None*, *gridDimY=None*, *gridLatName=None*, *gridLonName=None*, *topoDimX=None*, *topoDimY=None*, *topoLatName=None*, *topoLonName=None*, *convert_to_depth=True*)

Generate a bathymetry and ocean mask for a given data source topography or bathymetry. See `gridtools.topoutils.TopoUtils.regridTopo()`.

removeFillValueAttributes(*data=None*, *stringVars=None*)

Helper function to format the netCDF file to emulate the current styles.

Parameters

- **data** (*xarray*) – variables to format
- **stringVars** (*dict()*) – dictionary of string variables and lengths

Returns netCDF encoding

Return type dict()

For *data*, the `_FillValue` is masked.

For *stringVars*, supply a string length. e.g. `{'tile': 255}` This will result in an encoding of `{'dtype': 'S255', 'char_dim_name': 'string'}`.

resetPlotParameters()

Resets plot parameters for a grid.

rotate_u(*x*, *y*, *z*, *ux*, *uy*, *uz*, *theta*)

Rotate by angle θ around a general axis (ux,uy,uz).

rotate_u_mesh(*lam*, *phi*, *ux*, *uy*, *uz*, *theta*)

rotate_x(*x*, *y*, *z*, *theta*)

Rotate vector (x,y,z) by angle theta around x axis.

rotate_x_mesh(*lam*, *phi*, *theta*)

Rotate the whole mesh on globe by angle theta around x axis (passing through equator and prime meridian.).

rotate_y(*x*, *y*, *z*, *theta*)

Rotate vector (x,y,z) by angle theta around y axis.

rotate_y_mesh(*lam*, *phi*, *theta*)

Rotate the whole mesh on globe by angle theta around y axis (passing through equator and prime meridian+90.).

rotate_z(*x*, *y*, *z*, *theta*)

Rotate vector (x,y,z) by angle theta around z axis.

rotate_z_mesh(*lam*, *phi*, *theta*)

Rotate the whole mesh on globe by angle theta around z axis (globe polar axis).

saveDataset(*dsName*, *dsData*, ***kwargs*)

This allows saving variables to a file.

Parameters

- **dsName** (*string*) – data source name or filename
- **dsData** (*xarray object*) – dataset, data array or data object

Returns none

Return type none

Keyword arguments

- *overwrite* (boolean) – set to True to allow overwriting. Default: False
- *hashVariables* (list()) – names of variables to add a sha256sum attribute
- *mapVariables* (dict()) – map variable names to names stored in the output file. This argument takes precidence over all arguments.

Note: (Unimplemented) If the *dsName* is a data source (*ds:*) any variable map specified will be reversed before writing the final file.

---

saveGrid(*filename=None*, *directory=None*, *enc=None*)

This operation is destructive using the last known filename which can be overridden.

setDebugLevel(*newLevel*)

Set a new debug level.

Parameters **newLevel** (*integer*) – debug level to set or update

Returns none

Return type none

Note: Areas of code that typically cause errors have try/except blocks. Some of these have python debugging breakpoints that are active when the debug level is set to a positive number.

Currently defined debug levels: 0=off 1=extra messages 2=raise an exception 3=stop at breakpoints

---

setGridParameters(*gridParameters*, *subKey=None*)

Generic method for setting gridding parameters using dictionary arguments.

Parameters

- **gridParameters** (*dictionary*) – grid parameters to set or update
- **subKey** (*string*) – an entry in gridParameters that contains a dictionary of information to set or update

Returns none

Return type none

Note: Core gridParameter list. See other grid functions for other potential options. Defaults are **bold**. See the user manual for more details.

Primary keys

- *centerUnits* (string) – Grid center point units [**'degrees'**, 'meters']
- *centerX* (float) – Grid center in the j direction
- *centerY* (float) – Grid center in the i direction

- *dx* (`float`) – grid length in the j direction
- *dy* (`float`) – grid length in the i direction
- *dxUnits* (`string`) – grid length units [**'degrees'**, 'meters']
- *dyUnits* (`string`) – grid length units [**'degrees'**, 'meters']
- *nx* (`integer`) – number of grid points along the j direction
- *ny* (`integer`) – number of grid points along the i direction
- *tilt* (`float`) – degrees to rotate the grid (*LambertConformalConic only*)
- *gridResolution* (`float`) – grid cell size in i and j direction
- *gridResolutionX* (`float`) – grid cell size in the j direction
- *gridResolutionY* (`float`) – grid cell size in the i direction
- *gridResoultionUnits* (`string`) – grid cell units in the i and j direction [**'degrees'**, 'meters']
- *gridResoultionXUnits* (`string`) – grid cell units in the j direction [**'degrees'**, 'meters']
- *gridResoultionYUnits* (`string`) – grid cell units in the i direction [**'degrees'**, 'meters']

subKey 'projection'

- *name* (`string`) – Grid projection ['LambertConformalConic','Mercator','Stereographic']
- *lat_0* (`float degrees`) – Latitude of projection center [**0.0**]
- *lat_1* (`float degrees`) – First standard parallel (latitude) [**0.0**]
- *lat_2* (`float degrees`) – Second standard parallel (latitude) [**0.0**]
- *lat_ts* (`float degrees`) – Latitude of true scale. Defines the latitude where scale is not distorted. Takes precedence over `k_0` if both options are used together. For stereographic, if not set, will default to `lat_0`.
- *lon_0* (`float degrees`) – Longitude of projection center [**0.0**]
- *ellps* (`string`) – See `proj -le` for a list of available ellipsoids [**'GRS80'**]
- *R'* (`float meters`) – Radius of the sphere given in meters. If both R and ellps are given, R takes precedence.
- *x_0* (`float meters`) – False easting [**0.0**]
- *y_0* (`float meters`) – False northing [**0.0**]
- *k_0* (`float`) – Depending on projection, this value determines the scale factor for natural origin or the ellipsoid [**1.0**]

The subkey 'projection' mostly follows proj.org terminology for any giving projection. See: Lambert Conformal Conic, Mercator and Stereographic for more details.

MOM6 specific options

- *gridMode* (`integer`) – 2 = supergrid(*); 1 = actual grid [1 or **2**]

Warning: These options are similar to *`setPlotParameters()`* which control how this grid or other information is plotted. For instance, the **grid** projection and the **plot** projection can be in *different* projections.

`setLogLevel`(*newLevel*)
Set a new logging level.

Parameters **newLevel** (*integer or string*) – logging level to set or update

Returns none

Return type none

Note: Setting this to a positive number will increase the feedback from this module.

The available levels are:

Level	Numeric value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10

`setPlotCMap`(*varArgs*)
Set user defined color map or use matplotlib default.

`setPlotCbarkwargs`(*varArgs*)

`setPlotLevels`(*varArgs*)

`setPlotNorm`(*varArgs*)

`setPlotParameters`(*plotParameters*, *subKey=None*)
A generic method for setting plotting parameters using dictionary arguments. These parameters are applied to plots using the grid or any requested field. The **grid** projection may be different than the requested **plot** projection.

Parameters

- **plotParameters** (*dictionary*) – plot parameters to set or update
- **subkey** – an entry in plotParameters that contains a dictionary of information to set or update

Returns none

Return type none

Note: Plot parameters persist for as long as the python *GridUtils* object exists.

See the user manual for additional details.

Primary keys

- *figsize* (`(float inches, float inches)`) – matplotlib figure size (width, height) Default: **(5.0, 3.75)**
- *extent* (`[x0, x1, y0, y1]`) – map extent of given coordinate system (see *extentCRS*) If no extent is given, **[]**, then the global extent, `set_global()`, is used. See matplotlib geoaxes. Default: **[]**
- *extentCRS* (`cartopy.crs method`) – cartopy crs You must have the cartopy.crs module loaded to change this setting. See Cartopy projection list. Default: **cartopy.crs.PlateCarree()**
- *showGrid* (`boolean`) – show the grid outline. Default: **True**

- *showGridCells* (`boolean`) – show the grid cells. Default: **False**
- *showSupergrid* (`boolean`) – show the MOM6 supergrid cells. Default: **False**
- *title* (`string`) – set plot title. Default: **None**
- *iColor* (`string`) – matplotlib color for i vertices. Default: **'k'** (black)
- *jColor* (`string`) – matplotlib color for j vertices. Default: **'k'** (black)
- *iLinewidth* (`float points`) – matplotlib linewidth for i vertices. Default: **1.0**
- *jLinewidth* (`float points`) – matplotlib linewidth for j vertices. Default: **1.0**

Line width in matplotlib is generally defined by a numerical value over the default dots per inch (dpi). The nominal dpi value is 72 dots per inch. A line width of one (1.0) is 1/72nd of an inch at 72 dpi. A Stack Overflow post discusses dpi and figure size in good detail.

subKey 'projection'

- *name* (`string`) – Grid projection ['LambertConformalConic','Mercator','Stereographic']
- *lat_0* (`float degrees`) – Latitude of projection center [**0.0**]
- *lat_1* (`float degrees`) – First standard parallel (latitude) [**0.0**]
- *lat_2* (`float degrees`) – Second standard parallel (latitude) [**0.0**]
- *lat_ts* (`float degrees`) – Latitude of true scale. Defines the latitude where scale is not distorted. Takes precedence over `k_0` if both options are used together. For stereographic, if not set, will default to `lat_0`.
- *lon_0* (`float degrees`) – Longitude of projection center [**0.0**]
- *ellps* (`string`) – See `proj -le` for a list of available ellipsoids [**'GRS80'**]
- *R* (`float meters`) – Radius of the sphere given in meters. If both R and ellps are given, R takes precedence.
- *x_0* (`float meters`) – False easting [**0.0**]
- *y_0* (`float meters`) – False northing [**0.0**]
- *k_0* (`float`) – Depending on projection, this value determines the scale factor for natural origin or the ellipsoid [**1.0**]

The subkey 'projection' mostly follows proj.org terminology for any giving projection. See: Lambert Conformal Conic, Mercator and Stereographic for more details.

Warning: These options are similar to *`setGridParameters()`* which controls the representation of the grid. In this library, it is possible that the **grid** projection and the **plot** projection can be in *different* projections.

`setVerboseLevel`(*newLevel*)

Set a new verbose level.

Parameters `newLevel` (*`integer or string`*) – verbose level to set or update

Returns none

Return type none

Note: Setting this to a positive number will increase the feedback from this module.

The available levels are:

Level	Numeric value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10

---

showGridMetadata()
Show current grid metadata.

showGridParameters()
Show current grid parameters.

showLoggers()
Display an alphabetical list of loggers. Messages sent at the INFO level.

showMessages()
This converts the message buffer to text with linefeeds.

showPlotParameters()
Show current plot parameters.

subsetGrid(*scaleFactor*)
Subsets current grid by the specified scale factor. Scale factor must be an integer and be evenly divisble into the regular grid. A subsetted grid is returned or None on any error.

updateAxes(*ax*, *varArgs*)
Apply figure options to axes based on parameters passed to variable.

updateGridMetadata(***kwargs*)
Generic routine to apply metadata to appropriate entries to the existing grid loaded into the gridutils object. This also checks any existing gridParameters. This attempts to set several attributes. If only setting one or two attributes, consider using `gridtools.gridutils.checkGridMetadata()`.

Keyword arguments:

- *updateMetadata* (`boolean`) – Allow updates to metadata global attributes. If False, existing metadata is not overwritten. If an attribute is missing, the global attribute is added. Default: True

useDataSource(*dsObj*)

writeLandmask(*dsData*, *dsVariable*, *outVariable*, *outFile*, ***kwargs*)
Write a land mask based on provided information.

writeOceanmask(*dsData*, *dsVariable*, *outVariable*, *outFile*, ***kwargs*)
Write a ocean mask based on provided information.

5.6 meshrefinement module

This was originally GMesh.py from Niki Zahdah. [Zad20b]

class gridtools.meshrefinement.**MeshRefinement**(*shape=None*, *lon=None*, *lat=None*, *area=None*, *lon0=-180.0*, *from_cell_center=False*, *rfl=0*)

Bases: object

Describes 2D meshes for ESMs.

Meshes have shape=(nj,ni) cells with (nj+1,ni+1) vertices with coordinates (lon,lat).

When constructing, either provide 1d or 2d coordinates (lon,lat), or assume a uniform spherical grid with 'shape' cells covering the whole sphere with longitudes starting at lon0.

Attributes:

shape - (nj,ni) ni - number of cells in i-direction (last) nj - number of cells in j-direction (first) lon - longitude of mesh (cell corners), shape (nj+1,ni=1) lat - latitude of mesh (cell corners), shape (nj+1,ni=1) area - area of cells, shape (nj,ni)

coarsenby2(*coarser_mesh*)
Set the height for lower level Mesh by coarsening

dump()
Dump Mesh to tty.

find_nn_uniform_source(*lon*, *lat*)
Returns the i,j arrays for the indexes of the nearest neighbor point to grid (lon,lat)

least_square_plane_estimate(*xs*, *ys*, *zs*)
This function returns the estimates for h2 and h and also mean,min,max of the date in each grid cell.

mdist(*x2*)
Returns positive distance modulo 360.

pcolormesh(*axis*, *data*, ***kwargs*)

plot(*axis*, *subsample=1*, *linecolor='k'*, ***kwargs*)

refine_loop(*src_lon*, *src_lat*, *max_stages=32*, *max_mb=500*, *verbose=True*, *singularity_radius=0.25*)
Repeatedly refines the mesh until all cells in the source grid are intercepted by mesh nodes. Returns a list of the refined meshes starting with parent mesh.

refineby2(*work_in_3d=True*)
Returns new Mesh instance with twice the resolution

rotate(*y_rot=0*, *z_rot=0*)
Sequentially apply a rotation about the Y-axis and then the Z-axis.

sample_source_data_on_target_mesh(*xs*, *ys*, *zs*)
Returns the array on target mesh with values equal to the nearest-neighbor source point data

source_hits(*xs*, *ys*, *singularity_radius=0.25*)
Returns an mask array of 1's if a cell with center (xs,ys) is intercepted by a node on the mesh, 0 if no node falls in a cell

gridtools.meshrefinement.**fourPointAve**(*x*)

gridtools.meshrefinement.**is_mesh_uniform**(*lon*, *lat*)
Returns True if the input grid (lon,lat) is uniform and False otherwise

5.7 meshutils module

gridtools.meshutils.**writeLandmask**(*grd*, *dsData*, *dsVariable*, *outVariable*, *outFile*, ***kwargs*)
Write a land mask based on provided information. This routine assumes a depth field is being passed as an argument to create the mask.

MOM6 h-points are masked at and above the MASKING_DEPTH. A depth equal to the MASKING_DEPTH is masked as land. MASKING_DEPTH is ignored if negative. The default depth is zero (0.0).

gridtools.meshutils.**writeOceanmask**(*grd*, *dsData*, *dsVariable*, *outVariable*, *outFile*, ***kwargs*)
Write a ocean mask based on provided information. This routine assumes a depth field is being passed as an argument to create the mask.

MOM6 h-points are masked at and above the MASKING_DEPTH. A depth equal to the MASKING_DEPTH is masked as land. MASKING_DEPTH is ignored if negative. The default depth is zero (0.0).

5.8 sanity module

Generic sanity check routines.

gridtools.sanity.**saneDepthOptions**(***kwargs*)
Sanity check **_DEPTH* options for MOM6.

The following equation must be true for this routine to return **True**:

```
MAXIMUM_DEPTH >= MINIMUM_DEPTH >= MASKING_DEPTH
```

A land mask is applied if a depth is equal to or shallower than the `MASKING_DEPTH`. If `MASKING_DEPTH` is undefined or deeper than `MINIMUM_DEPTH`, the land mask will use the `MINIMUM_DEPTH` to apply the land mask.

Returns True of depth options are sane, False otherwise.

Return type boolean

As the model is initialized, if this parameter is undefined, it is set to the maximum as seen from the supplied topography grid.

References

- **MOM_shared_initialization.F90**
  - *MINIMUM_DEPTH*, *MASKING_DEPTH*
- **MOM_fixed_initialization.F90**
  - *MAXIMUM_DEPTH*
- diagnoseMaximumDepth

5.9 spherical module

gridtools.spherical.**angle_between**(*v1*, *v2*, *v3*)
Returns angle v2-v1-v3 i.e betweeen v1-v2 and v1-v3.

gridtools.spherical.**angle_through_center**(*p1*, *p2*)
Angle at center of sphere between two points on the surface of the sphere. Positions are given as (latitude,longitude) tuples measured in degrees.

gridtools.spherical.**quad_area**(*lat*, *lon*)
Returns area of spherical quad (bounded by great arcs).

5.10 sysinfo module

class gridtools.sysinfo.**SysInfo**(*grd=None*)
Bases: `object`

A small helper class to provide basic system information and misc operating system utilities. Useful for debugging module or system conflicts. Simple core calls should be placed into the utils module. More complex functions that depend on results from other functions should go into this module. For example, tracking or detecting module versions are in this module.

detectPythonEnvironment()
This attempts to detect a python virtual environment that is running the python script.

This attempts to guess if the script is running under conda or venv. If the answer is neither, then native is assumed.

Other possible environments are covered in an article: https://towardsdatascience.com/comparing-python-virtual-environment-tools-9a6543643a44

- virtualenv/virtualenvwrapper
- pew (unmaintained ~ March 2018)
- venv (recommended since Python 3.5)
- pipenv

dumpVersionData()
Return a string for use in netCDF attributes

get_jupyterlab_modules()

get_python_loaded_modules()

get_python_module_version(*mdlName*)
Try to obtain the python module version. This returns the version numer if the module was found or None if nothing was found or an error was encountered.

isAvailable(*pkgReq*, *verReq*, *orLower=False*)

is_lab_notebook()
Detects if current process is a jupyter-lab process.

loadVersionData(***kwargs*)
This function loads module version data in a variety of ways. The default is for modules currently loaded by the python executable.

Keyword arguments

- *usePackageManager* (`boolean`) Use a package manager to discover module versions. Default: False
- *packageManager* (`string`) Specify *conda* or *venv* as the package manager to use to show versions of installed software. Default: auto

---

Note: This function by default only obtains version information for modules already loaded into the python environment. Switching to a package manager will display all installed software which may be more information than desired.

For the *conda* package manager, the conda python module may be installed or the *conda* executable is available.

The conda python module may be installed within the environment: *conda install -c conda-forge conda*

---

printInfo(*k*, *kd*, *msg*)

resetVersionData()
This resets any loaded module version data.

runCommand(*cmdString*)
Generic function to run a command string and return the results as an array of three items. Return: (stdout, stderr, returncode)

show(*showOnly=[]*, *vList=[]*)

showAll(*vList=[]*)

showEnvironment()

showPackageVersions(*vList=[]*)

showSystem()

5.11 topoutils module

5.12 utils module

Generic utility functions

gridtools.utils.**checkArgument**(*vDict*, *vKey*, *vVal*)
This checks to see if there is a key in the passed dictionary. If the key does not exist, create the key with the specified value. No logging of any kind is performed. Any errors are silently ignored.

Parameters

- **vDict** (*dict*) – python dictionary; typically kwargs
- **vKey** (*string*) – dictionary key to check
- **vVal** (*value*) – value to use for dictionary key if it does not exist

Returns none

Return type none

gridtools.utils.**get_architecture**()
Returns the executable architecture.

gridtools.utils.**get_git_repo_version_info**(*grd=None*)
Describe the current version of this script as known by Git.

This function is based on code from [IAH20].

gridtools.utils.**get_history_entry**(*argv*)
Construct an entry for the global 'history' attribute of a NetCDF file, which is a date and the command used.

This function is copied from [IAH20].

gridtools.utils.**get_machine**()
Returns the machine type: i386, x86_64, etc.

gridtools.utils.**get_processor**()
Returns the real processor name.

gridtools.utils.**get_python_version**()
Returns the python version.

gridtools.utils.**get_release**()
Returns the system' release version.

gridtools.utils.**get_system**()
Returns the system/OS name: Linux, Darwin, etc.

gridtools.utils.**runningHow**()
Utility function to determine how a python script is being run.

Detected Value	Returned Value
ZMQInteractiveShell	jupyter
TerminalInteractiveShell	ipython
<other>	<other>
on error/exception	python

gridtools.utils.**sha256sum**(*xrData*)
Utility function that returns a hash of the data provided.

5.13 grids

5.13.1 mom6 module

Generic class and utility funtions for handling MOM6 grids.

class gridtools.grids.mom6.**MOM6**
Bases: object

approximate_MOM6_grid_metrics()
Fill in missing MOM6 supergrid metrics by computing best guess values.

This function is based on code from [IAH20].

convert_ROMS_to_MOM6(*roms_grid*, ***kwargs*)
Convert the ROMS grid data into a skeleton MOM6 grid, mainly by merging the four sets of point locations from the ROMS grid into a single supergrid for MOM6.

This function is based on code from [IAH20].

`getGrid()`
Returns the converted grid as an xarray data structure.

This function is based on code from [IAH20].

`setup_MOM6_grid(**kwargs)`
Create an empty MOM6 data structure.

This function is based on code from [IAH20].

Keyword arguments:

- *tileName* (`string`) – name to assign to the solo tile. Default: "tile1"
- *inputDirectory* (`string`) – absolute or relative path to write model input files. Default: "INPUT"
- *relativeToINPUTDir* (`string`) – absolute or relative path for mosaic files to the INPUT directory. Default: "./" which refers to the "INPUT" directory.
- *supergridName* (`string`) – name to assign to the supergrid filename. Default: "ocean_hgrid.nc"
- *topographyFilename* (`string`) – filename used to write topographic grid. Default: "ocean_topog.nc"
- *mosaicFilename* (`string`) – filename for mosaic file. Default: "ocean_mosaic.nc"
- *landmaskFilename* (`string`) – filename used to write the land mask. Default: "land_mask.nc"
- *oceanmaskFilename* (`string`) – filename used to write the ocean mask. Default: "ocean_mask.nc"

Returns none

Return type none

`write_MOM6_coupler_mosaic_file(grd, **kwargs)`
Write the compler mosaic file, which references all the rest. Based on 'make_quick_mosaic' tool in version 5 of MOM (http://www.mom-ocean.org/).

This function is based on code from [IAH20].

`write_MOM6_exchange_grid_files(grd, **kwargs)`
Write three exchange grid files. Based on 'make_quick_mosaic' tool in version 5 of MOM (http://www.mom-ocean.org/).

This function is based on code from [IAH20].

`write_MOM6_land_mask_file(grd, **kwargs)`
Write the land mask file. Based on 'make_quick_mosaic' tool in version 5 of MOM (http://www.mom-ocean.org/).

This function is based on code from [IAH20].

`write_MOM6_ocean_mask_file(grd, **kwargs)`
Write the ocean mask file. Based on 'make_quick_mosaic' tool in version 5 of MOM (http://www.mom-ocean.org/).

This function is based on code from [IAH20].

`write_MOM6_solo_mosaic_file(grd, **kwargs)`

Write the "solo mosaic" file, which describes to the FMS infrastructure where to find the grid file(s). Based on tools in version 5 of MOM (http://www.mom-ocean.org/).

Keyword arguments:

- *intType* (`str`) – The default type to be written out to for the FMS coupler tile files. Default: int32

This function is based on code from [IAH20].

write_MOM6_topography_file(*grd*, ***kwargs*)
Save the MOM6 ocean topography grid in a separate file.

This function is based on code from [IAH20].

5.13.2 roms module

Generic class and utility functions for handling ROMS grids.

class gridtools.grids.roms.**ROMS**(*grd=None*)
Bases: object

edit_mask_mesh(*hgrid*, *proj=None*, ***kwargs*)

getGrid(*variable=None*)
Return the ROMS grid to the caller.

get_ROMS_grid(*gridid*, *zeta=None*, *hist_file=None*, *grid_file=None*)
grd = get_ROMS_grid(self, gridid, hist_file=None, grid_file=None)

Load ROMS grid object.

gridid is a string with the name of the grid in it. If hist_file and grid_file are not passed into the function, or are set to None, then gridid is used to get the grid data from the gridid.txt file.

if hist_file and grid_file are given, and they are the file paths to a ROMS history file and grid file respectively, the grid information will be extracted from those files, and gridid will be used to name that grid for the rest of the python session.

grd.vgrid is a s_coordinate or a z_coordinate object, depending on gridid.grdtype. grd.vgrid.z_r and grd.vgrid.z_w (grd.vgrid.z for a z_coordinate object) can be indexed in order to retreive the actual depths. The free surface time serie zeta can be provided as an optional argument. Note that the values of zeta are not calculated until z is indexed, so a netCDF variable for zeta may be passed, even if the file is large, as only the values that are required will be retrieved from the file.

This function is based on code from [HHT+21].

get_ROMS_hgrid(*gridid*)
hgrid = get_ROMS_hgrid(gridid)

Load ROMS horizontal grid object

get_ROMS_vgrid(*gridid*, *zeta=None*)
vgrid = get_ROMS_vgrid(gridid)

Load ROMS vertical grid object. vgrid is a s_coordinate or a z_coordinate object, depending on gridid.grdtype. vgrid.z_r and vgrid.z_w (vgrid.z for a z_coordinate object) can be indexed in order to retreive the actual depths. The free surface time serie zeta can be provided as an optional argument. Note that the values of zeta are not calculated until z is indexed, so a netCDF variable for zeta may be passed, even if the file is large, as only the values that are required will be retrieved from the file.

read_ROMS_grid(*grd*)
Load the ROMS grid previous read by GridUtils().

This function is based on code from [IAH20].

trim_ROMS_grid()
Remove extraneous points on the outside of the ROMS grid.

This function is based on code from [IAH20].

write_ROMS_grid(*grd*, *filename*)
Write ROMS_CGrid class on a NetCDF file.

This function is based on code from [HHT+21].

class gridtools.grids.roms.**ROMS_Grid**(*name*, *hgrid=<class 'gridtools.grids.roms_hgrid.CGrid'>*, *vgrid=<class 'gridtools.grids.roms_vgrid.s_coordinate'>*)

Bases: object

grd = ROMS_Grid(hgrid, vgrid)

ROMS Grid object combining horizontal and vertical grid

This class is based on code from [HHT+21].

class gridtools.grids.roms.**ROMS_gridinfo**(*gridid*, *grid_file=None*, *hist_file=None*)
Bases: object

gridinfo = ROMS_gridinfo(gridid,grid_file=None,hist_file=None)

Return an object with grid information for gridid.

There are two ways to define the grid information. If grid_file and hist_file are not passed to the object when it is created, the information is retrieved from gridid.txt. To add new grid please edit your gridid.txt. You need to define an environment variable ROMS_GRIDID_FILE pointing to your gridid.txt file. Just copy an existing grid and modify the definition accordingly to your case (Be carefull with space and blank line).

If grid_file is the path to a ROMS grid file, and hist_file is the path to a ROMS history file, then the grid information will be read from those files. Gridid can then be used to refer to this grid information so that the grid and history files do not be included in subsequent calls.

This class is based on code from [HHT+21].

gridid_dictionary = {}

5.13.3 roms_hgrid module

Tools for creating and working with Arakawa C-Grids

This module is a modified copy from [HHT+21].

class gridtools.grids.roms_hgrid.**BoundaryInteractor**(*x*, *y=None*, *beta=None*, *ax=None*, *proj=None*, ***gridgen_options*)

Bases: object

Interactive grid creation

*bry = BoundaryClick(x=[], y=[], beta=None, ax=gca(), **gridgen_options)*

The initial boundary polygon points (x and y) are counterclockwise, starting in the upper left corner of the boundary.

Key commands:

t : toggle visibility of verticies d : delete a vertex i : insert a vertex at a point on the polygon line

p : set vertex as beta=1 (a Positive turn, marked with green triangle) m : set vertex as beta=1 (a Negative turn, marked with red triangle) z : set vertex as beta=0 (no corner, marked with a black dot)

G : generate grid from the current boundary using gridgen T : toggle visability of the current grid

bry.**dump**(*bry_file*)
Write the current boundary informtion (bry.x, bry.y, bry.beta) to a cPickle file bry_file.

bry.**load**(*bry_file*)
Read in boundary informtion (x, y, beta) from the cPickle file bry_file.

bry.**remove_grid**()
Remove gridlines from axes.

bry.**x**
the X boundary points

bry.**y**
the Y boundary points

bry.**verts**
the verticies of the grid

bry.**grd**
the CGrid object

get_xdata()

get_ydata()

load_bry(*bry_file='bry.pickle'*)

remove_grid()
Remove a generated grid from the BoundaryClick figure

save_bry(*bry_file='bry.pickle'*)

save_grid(*grid_file='grid.pickle'*)

property verts

property x

property y

class gridtools.grids.roms_hgrid.**CGrid**(*x_vert*, *y_vert*, *x_rho=None*, *y_rho=None*, *x_u=None*, *y_u=None*, *x_v=None*, *y_v=None*, *x_psi=None*, *y_psi=None*, *dx=None*, *dy=None*, *dndx=None*, *dmde=None*, *angle_rho=None*)

Bases: object

Curvilinear Arakawa C-Grid

The basis for the CGrid class are two arrays defining the verticies of the grid in Cartesian (for geographic coordinates, see CGrid_geo). An optional mask may be defined on the cell centers. Other Arakawa C-grid properties, such as the locations of the cell centers (rho-points), cell edges (u and v velocity points), cell widths (dx and dy) and other metrics (angle, dmde, and dndx) are all calculated internally from the vertex points.

Input vertex arrays may be either type np.array or np.ma.MaskedArray. If masked arrays are used, the mask will be a combination of the specified mask (if given) and the masked locations.

Examples

```
>>> x, y = mgrid[0.0:7.0, 0.0:8.0]
>>> x = np.ma.masked_where( (x<3) & (y<3), x)
>>> y = np.ma.MaskedArray(y, x.mask)
>>> grd = pyroms.grid.CGrid(x, y)
>>> print(grd.x_rho)
[[-- -- -- 0.5 0.5 0.5 0.5]
 [-- -- -- 1.5 1.5 1.5 1.5]
```

(continues on next page)

(continued from previous page)

```
 [-- -- -- 2.5 2.5 2.5 2.5]
 [3.5 3.5 3.5 3.5 3.5 3.5 3.5]
 [4.5 4.5 4.5 4.5 4.5 4.5 4.5]
 [5.5 5.5 5.5 5.5 5.5 5.5 5.5]]
>>> print(grd.mask)
[[ 0.  0.  0.  1.  1.  1.  1.]
 [ 0.  0.  0.  1.  1.  1.  1.]
 [ 0.  0.  0.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.]]
```

calculate_orthogonality()
Calculate orthogonality error in radians

property mask
Return mask_rho

mask_polygon(*polyverts*, *mask_value=0.0*)
Mask Cartesian points contained within the polygon defined by polyverts

A cell is masked if the cell center (x_rho, y_rho) is within the polygon. Other sub-masks (mask_u, mask_v, and mask_psi) are updated automatically.

mask_value [=0.0] may be specified to alter the value of the mask set within the polygon. E.g., mask_value=1 for water points.

property mask_psi
Return mask_psi

property mask_u
Return mask_u

property mask_v
Return mask_v

property x
Return x_vert

property y
Return x_vert

class gridtools.grids.roms_hgrid.**CGrid_geo**(*lon_vert*, *lat_vert*, *proj*, *use_gcdist=True*, *ellipse='WGS84'*, *lon_rho=None*, *lat_rho=None*, *lon_u=None*, *lat_u=None*, *lon_v=None*, *lat_v=None*, *lon_psi=None*, *lat_psi=None*, *dx=None*, *dy=None*, *dndx=None*, *dmde=None*, *angle_rho=None*)

Bases: *gridtools.grids.roms_hgrid.CGrid*

Curvilinear Arakawa C-grid defined in geographic coordinates

For a geographic grid, a projection may be specified, or The default projection for will be defined by the matplotlib.toolkits.Basemap projection:

proj = Basemap(projection='merc', resolution=None, lat_ts=0.0)

For a geographic grid, the cell widths are determined by the great circle distances. Angles, however, are defined using the projected coordinates, so a projection that conserves angles must be used. This means typically either Mercator (projection='merc') or Lambert Conformal Conic (projection='lcc').

property lat
Shorthand for lat_vert

property lon
Shorthand for lon_vert

mask_polygon_geo(*mask_value=0.0*)

class gridtools.grids.roms_hgrid.**Focus**
Bases: object

Return a container for a sequence of Focus objects

foc = Focus()

The sequence is populated by using the 'add_focus_x' and 'add_focus_y' methods. These methods define a point ('xo' or 'yo'), around witch to focus, a focusing factor of 'focus', and x and y extent of focusing given by Rx or Ry. The region of focusing will be approximately Gausian, and the resolution will be increased by approximately the value of factor.

foc.**add_focus_x**(*xo*, *factor=2.0*, *Rx=0.1*)

foc.**add_focus_y**(*yo*, *factor=2.0*, *Ry=0.1*)

Calls to the object return transformed coordinates:
xf, yf = foc(x, y)

where x and y must be within [0, 1], and are typically a uniform,

normalized grid. The focused grid will be the result of applying each of

the focus elements in the sequence they are added to the series.

Examples

```
>>> foc = pyroms.grid.Focus()
>>> foc.add_focus_x(0.2, factor=3.0, Rx=0.2)
>>> foc.add_focus_y(0.6, factor=5.0, Ry=0.35)
```

```
>>> x, y = np.mgrid[0:1:3j,0:1:3j]
>>> xf, yf = foc(x, y)
```

```
>>> print(xf)
[[ 0.          0.          0.        ]
 [ 0.36594617  0.36594617  0.36594617]
 [ 1.          1.          1.        ]]
>>> print(yf)
[[ 0.          0.62479833  1.        ]
 [ 0.          0.62479833  1.        ]
 [ 0.          0.62479833  1.        ]]
```

add_focus_x(*xo*, *factor=2.0*, *Rx=0.1*)
docstring for add_point

add_focus_y(*yo*, *factor=2.0*, *Ry=0.1*)
docstring for add_point

class gridtools.grids.roms_hgrid.**Gridgen**(*xbry*, *ybry*, *beta*, *shape*, *ul_idx=0*, *focus=None*, *proj=None*, *nnodes=14*, *precision=1e-12*, *nppe=3*, *newton=True*, *thin=True*, *checksimplepoly=True*, *verbose=False*)

Bases: *gridtools.grids.roms_hgrid.CGrid*

docstring for Gridgen

generate_grid()

gridtools.grids.roms_hgrid.**dist_point_to_segment**(*p*, *s0*, *s1*)

Get the distance of a point to a segment.

**p*, *s0*, *s1* are *xy* sequences*

This algorithm is from http://geomalgorithms.com/a02-_lines.html

class gridtools.grids.roms_hgrid.**edit_mask_mesh**(*grd*, *proj=None*, ***kwargs*)

Bases: object

Interactive mask editor

edit_mask_mesh(grd, proj)

Edit grd mask. Mask/Unsmask cell by a simple click on the cell. Mask modification are store in mask_change.txt for further use.

Key commands: e : toggle between Editing/Viewing mode

class gridtools.grids.roms_hgrid.**edit_mask_mesh_ij**(*grd*, *coast=None*, ***kwargs*)

Bases: object

Interactive mask editor

edit_mask_mesh_ij(grd)

Edit grd mask. Mask/Unsmask cell by a simple click on the cell. Mask modification are store in mask_change.txt for further use.

Key commands: e : toggle between Editing/Viewing mode

class gridtools.grids.roms_hgrid.**get_position_from_map**(*grd*, *proj=None*, ***kwargs*)

Bases: object

Get cell index position Interactively

get_position_from_map(grd, proj)

Get index i, j as well as lon, lat coordinates for one cell simply by clicking on the cell.

Key commands: i : toggle between Interactive/Viewing mode

gridtools.grids.roms_hgrid.**rho_to_vert**(*xr*, *yr*, *pm*, *pn*, *ang*)

gridtools.grids.roms_hgrid.**rho_to_vert_geo**(*lonr*, *latr*, *lonp*, *latp*)

gridtools.grids.roms_hgrid.**uvp_masks**(*rmask*)

return u-, v-, and psi-masks based on input rho-mask

Parameters **rmask** (*ndarray*) – mask at CGrid rho-points

Returns **(umask, vmask, pmask)** – masks at u-, v-, and psi-points

Return type ndarrays

5.13.4 roms_io module

A thin wrapper for netCDF4.Dataset and netCDF4.MFDataset

This module provides two functions, Dataset and MFDataset, that are similar to the netCDF[3/4] functions of the same name. This package is a thin wrapper around these functions, and provides two services. First of all, it will use either netCDF3 or netCDF4 (prefering the later), so that the netCDF package does not need to be changed on different systems that only have one or the other. Second, it will pass through netCDF[3/4] objects unchanged, so that netCDF objects, filenames, lists of files, or strings with wildcards can be passed to the function indescriminately.

Examples of usage:

with an input of a string:

```
# returns netCDF4.Dataset object based on file
nc = pyroms.io.Dataset(file)

# returns MFnetCDF4.Dataset object based on file (with wildcard chars)
nc = pyroms.io.MFDataset(file)
```

with an input of a list of files:

```
# returns MFDataset object based on list of files
nc = pyroms.io.Dataset(files)

# returns MFDataset object based on list of files
nc = pyroms.io.MFDataset(files)
```

with an input of a netCDF4.Dataset or MFnetCDF4.Dataset object:

```
# passes through netCDF4.Dataset or MFnetCDF4.Dataset object
nc = pyroms.io.Dataset(nc)

# passes through MFDataset object based on file (with wildcard chars)
nc = pyroms.io.MFDataset(nc)
```

This module is a modified copy from [HHT+21].

gridtools.grids.roms_io.**Dataset**(*ncfile*)

Return an appropriate netcdf object: netCDF4 object given a file string MFnetCDF4 object given a list of files

A netCDF4 or MFnetCDF4 object returns itself.

gridtools.grids.roms_io.**MFDataset**(*ncfile*)

Return an MFnetCDF4 object given a string or list. A string is expanded with wildcards using glob. A netCDF4 or MFnetCDF4 object returns itself.

5.13.5 roms_vgrid module

Various vertical coordinates

Presently, only ocean s-coordinates are supported. Future plans will be to include all of the vertical coordinate systems defined by the CF conventions.

This module is a modified copy from [HHT+21].

class gridtools.grids.roms_vgrid.**s_coordinate**(*h*, *theta_b*, *theta_s*, *Tcline*, *N*, *hraw=None*, *zeta=None*)
Bases: object

Song and Haidvogel (1994) vertical coordinate transformation (Vtransform=1) and stretching functions (Vstretching=1).

return an object that can be indexed to return depths

s = s_coordinate(h, theta_b, theta_s, Tcline, N)

class gridtools.grids.roms_vgrid.**s_coordinate_2**(*h*, *theta_b*, *theta_s*, *Tcline*, *N*, *hraw=None*, *zeta=None*)
Bases: *gridtools.grids.roms_vgrid.s_coordinate*

A. Shchepetkin (2005) UCLA-ROMS vertical coordinate transformation (Vtransform=2) and stretching functions (Vstretching=2).

return an object that can be indexed to return depths

s = s_coordinate_2(h, theta_b, theta_s, Tcline, N)

class gridtools.grids.roms_vgrid.**s_coordinate_4**(*h*, *theta_b*, *theta_s*, *Tcline*, *N*, *hraw=None*, *zeta=None*)
Bases: *gridtools.grids.roms_vgrid.s_coordinate*

A. Shchepetkin (2005) UCLA-ROMS vertical coordinate transformation (Vtransform=2) and stretching functions (Vstretching=4).

return an object that can be indexed to return depths

s = s_coordinate_4(h, theta_b, theta_s, Tcline, N)

class gridtools.grids.roms_vgrid.**s_coordinate_5**(*h*, *theta_b*, *theta_s*, *Tcline*, *N*, *hraw=None*, *zeta=None*)
Bases: *gridtools.grids.roms_vgrid.s_coordinate*

A. Shchepetkin (2005) UCLA-ROMS vertical coordinate transformation (Vtransform=2) and stretching functions (Vstretching=5).

return an object that can be indexed to return depths

s = s_coordinate_5(h, theta_b, theta_s, Tcline, N)

Brian Powell's surface stretching.

class gridtools.grids.roms_vgrid.**z_coordinate**(*h*, *depth*, *N*)
Bases: object

return an object that can be indexed to return depths

z = z_coordinate(h, depth, N)

class gridtools.grids.roms_vgrid.**z_r**(*h*, *hc*, *N*, *s_rho*, *Cs_r*, *zeta*, *Vtrans*)
Bases: object

return an object that can be indexed to return depths of rho point

z_r = z_r(h, hc, N, s_rho, Cs_r, zeta, Vtrans)

class gridtools.grids.roms_vgrid.**z_w**(*h*, *hc*, *Np*, *s_w*, *Cs_w*, *zeta*, *Vtrans*)
Bases: object

return an object that can be indexed to return depths of w point

z_w = z_w(h, hc, Np, s_w, Cs_w, zeta, Vtrans)

5.13.6 greatcircle module

Module *pyroms/pyroms/extern/greatcircle.py* copied from [HHT+21].

class gridtools.grids.greatcircle.**GreatCircle**(*rmajor*, *rminor*, *lon1*, *lat1*, *lon2*, *lat2*)
Bases: object

formula for perfect sphere from Ed Williams' 'Aviation Formulary' (http://williams.best.vwh.net/avform.htm)

code for ellipsoid posted to GMT mailing list by Jim Leven in Dec 1999

Contact: Jeff Whitaker <jeffrey.s.whitaker@noaa.gov>

points(*npoints*)
compute arrays of npoints equally spaced intermediate points along the great circle.

input parameter npoints is the number of points to compute.

Returns lons, lats (lists with longitudes and latitudes of intermediate points in degrees).

For example npoints=10 will return arrays lons,lats of 10 equally spaced points along the great circle.

gridtools.grids.greatcircle.**vinc_dist**(*f*, *a*, *phi1*, *lembda1*, *phi2*, *lembda2*)
Returns the distance between two geographic points on the ellipsoid and the forward and reverse azimuths between these points. lats, longs and azimuths are in radians, distance in metres

Returns (s, alpha12, alpha21) as a tuple

gridtools.grids.greatcircle.**vinc_pt**(*f*, *a*, *phi1*, *lembda1*, *alpha12*, *s*)
Returns the lat and long of projected point and reverse azimuth given a reference point and a distance and azimuth to project. lats, longs and azimuths are passed in decimal degrees

Returns (phi2, lambda2, alpha21) as a tuple

CHAPTER

SIX

BIBLIOGRAPHY

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Adc03] A. Adcroft. Representation of topography by porous barriers and objective interpolation of topographic data. *Ocean Modelling*, 67:13–27, 2003. doi:10.1016/j.ocemod.2013.03.002.

[Adc19] A. Adcroft. Quick test of how portable intrinsics can be. Jul 2019. URL: https://github.com/adcroft/numpypi/commit/b9fcd8a122d463c9a43f67f8c15f5a3890833df8.

[Adc20] A. Adcroft. Conservatively re-grids runoff data to the nearest coastal wet-point of a mom6 ocean grid. Jun 2020. URL: https://github.com/adcroft/regrid_runoff.

[Dus20] R. Dussin. Regidding weights for bedmachine to gebco. Aug 2020. URL: https://github.com/raphaeldussin/regrid_weights_bedmachine_gebco/blob/16c85a9c544d54808abaaec223aa834a8c46d6fd/create_grids.py#L9.

[HHT+21] K. Hedstrom, M. Hadfield, B. Torgerson, R. Dussin, J. Pringle, O. Teruhisa, M. Dunphy, and C. Wang. Python tools for the Regional Ocean Modeling System (ROMS). Apr 2021. URL: https://github.com/ESMG/pyroms/commit/cd0fe39075825f97a7caf64e2c4c5a19f23302fd.

[IAH20] M. Ilicak, A. Adcroft, and M. Harrison. Roms to mom6 grid converter. Nov 2020. URL: https://github.com/ESMG/pyroms/blob/cd0fe39075825f97a7caf64e2c4c5a19f23302fd/examples/grid_MOM6/convert_ROMS_grid_to_MOM6.py.

[JCW+00] M. Jakobsson, N.Z. Cherkis, J. Woodward, R. Macnab, and B. Coakley. New grid of Arctic bathymetry aids scientists and mapmakers. 2000.

[JMC+12] M. Jakobsson, L. Mayer, B. Coakley, J.A. Dowdeswel, S. Forbes, B. Fridman, H. Hodnesdal, R. Noormets, R. Pederse, M. Rebesco, H. Werner Schenke, Y. Zarayskaya, D. Accettella, A. Armstrong, R.M. Anderson, P. Bienhoff, A. Camerlenghi, I. Church, M. Edwards, J.V. Gardner, J.K. Hall, B. Hell, O. Hestvik, Y. Kristoffersen, C. Marcussen, R. Mohammad, D. Mosher, S.V. Nghiem, M. Teresa Pedrosa, P.G. Travaglini, and P. Weatherall. The International Bathymetric Chart of the Arctic Ocean (IBCAO) Version 3.0. *Geophysical Research Letters*, 39:L12609:1–6, 2012. doi:10.1029/2012GL052219.

[JMB+20] M. Jakobsson, L.A. Mayer, C. Bringensparr, C.F. Castro, R. Mohammad, P. Johnson, T. Ketter, D. Accettella, D. Amblas, L. An, J.E. Arndt, M. Canals, J.L. Casamor, N. Chauché, B. Coakley, S. Danielson, M. Demarte, M. Dickson, B. Dorschel, J.A. Dowdeswell, S. Dreutter, A.C. Fremand, D. Gallant, J.K. Hall, L. Hehemann, H. Hodnesdal, J. Hong, R. Ivaldi, E. Kane, I. Klaucke, D.W. Krawczyk, Y. Kristoffersen, B.R. Kuipers, R. Millan, G. Masetti, M. Morlighem, R. Noormets M.M. Prescott, M. Rebesco, E. Rignot, I. Semiletov, A.J. Tate, P. Travaglini, I. Velicogna, P. Weatherall, W. Weinrebe, J.K. Willis, M. Wood, Y. Zarayskaya, T. Zhang, M. Zimmermann, and K.B. Zinglersen. The International Bathymetric Chart of the Arctic Ocean Version 4.0. *Scientific Data*, 7:176:1–14, 2020. doi:10.1038/s41597-020-0520-9.

[MJV03] R. Macnab, M. Jakobsson, and H. Varma. International Bathymetric Chart of the Arctic Ocean (IBCAO) — a new direction for ocean mapmaking. Apr 2003. URL: https://www.gebco.net/about_us/presentations_and_publications/.

[Sim21] J. Simkins. Conservative regridding using xesmf. Jun 2021. URL: https://github.com/jsimkins2/esm_lab/blob/da553a3608a2e3dfdb767013fbe904f29ed18709/bathyTools/examples/xesmf_regrid_ex.py.

[Zad20a] N. Zadeh. A collection of tools and documents for creating finite element tripolar grids for mom based ocean models. Feb 2020. URL: https://github.com/nikizadehgfdl/grid_generation/commit/136ffe549c8d9039d50a806833de839474b208e0.

[Zad20b] N. Zadeh. A set of tools to generate model topography given model grid and observation data. Aug 2020. URL: https://github.com/nikizadehgfdl/ocean_model_topog_generator/tree/38dd7043fe931e62dd563a47cf4b5edb7f8260be.

[GeophysicalFDLGFDLMSGroup21] Geophysical Fluid Dynamics Laboratory (GFDL) Modeling Systems Group. Tools for manipulating and creating netCDF inputs for FMS managed models. Jun 2021. URL: https://github.com/NOAA-GFDL/FRE-NCtools.

PYTHON MODULE INDEX

g

INDEX

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z