GeoModelGrids

Release 1.0.0rc3

Brad T. Aagaard

Apr 04, 2024

TABLE OF CONTENTS

1 Description **1**

2 License **3**

2.1 CC0 1.0 Universal Summary 3

2.2 Disclaimer 3

CHAPTER

ONE

DESCRIPTION

Georeferenced grid-based models composed of blocks with different grid resolutions.

CHAPTER

TWO

LICENSE

Unless otherwise noted, this project is in the public domain in the United States because it contains materials that originally came from the United States Geological Survey, an agency of the United States Department of Interior. For more information, see the official USGS copyright policy.

Additionally, we waive copyright and related rights in the work worldwide through the CC0 1.0 Universal public domain dedication.

2.1 CC0 1.0 Universal Summary

This is a human-readable summary of the Legal Code (read the full text).

2.1.1 No Copyright

The person who associated a work with this deed has dedicated the work to the public domain by waiving all of his or her rights to the work worldwide under copyright law, including all related and neighboring rights, to the extent allowed by law.

You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission.

2.1.2 Other Information

In no way are the patent or trademark rights of any person affected by CC0, nor are the rights that other persons may have in the work or in how the work is used, such as publicity or privacy rights.

Unless expressly stated otherwise, the person who associated a work with this deed makes no warranties about the work, and disclaims liability for all uses of the work, to the fullest extent permitted by applicable law. When using or citing the work, you should not imply endorsement by the author or the affirmer.

2.2 Disclaimer

This software is preliminary or provisional and is subject to revision. It is being provided to meet the need for timely best science. The software has not received final approval by the U.S. Geological Survey (USGS). No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. The software is provided on the condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from the authorized or unauthorized use of the software.

Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

2.2.1 Introduction

Overview

The GeoModelGrids project focuses on storing and querying georeferenced raster-based models with an emphasis on regional models. The models are composed of blocks, in which each block is a grid with a potentially different resolution that vary along the x, y, or z axis.

Motivation

- Support variable resolution models that include topography and are in any standard geographic projection.
- Support querying for model values on a wide range of platforms, from laptops and desktops to large clusters.

Features

- Store models using a self-describing, portable, widely-used storage scheme, HDF5.
- Models are composed of an arbitrary number of vertically stacked blocks of uniform or variable resolution.
- Models domains may be in wide variety of georeferenced coordinate systems and need not be aligned with the x and y axes of the geographic projection.
- If the model includes topography, the blocks will be warped to conform to the ground surface. Queries for applications requiring a flat top surface can use "squashing" relative to either the top surface of the of the model or topography/bathymetry (if provided).
- In serial queries (intended for use on laptops and desktops), only a portion of the model is loaded into memory, thereby allowing queries of models that are much larger than the available memory.
- In addition to discretization information, the self-describing format includes description of the names of the values and their units, and coordinate system information.

Use Cases

There are many possible use cases. Our first target use case is storing and querying 3D seismic velocity models for simulation and analysis of earthquake ground motions.

Storage Layout

Model Representation

We map the physical space of the model domain bounded on top by topography (or a flat surface) into logical space that has uniform or variable resolution grids. Fig. 2.1 illustrates a model with three uniform resolution grids.

The mapping from the physical space to topological space is:

$$z_{logical} = -dim_z * (z_{topography} - z_{physical})/(z_{topography} + dim_z) \tag{2.1}$$

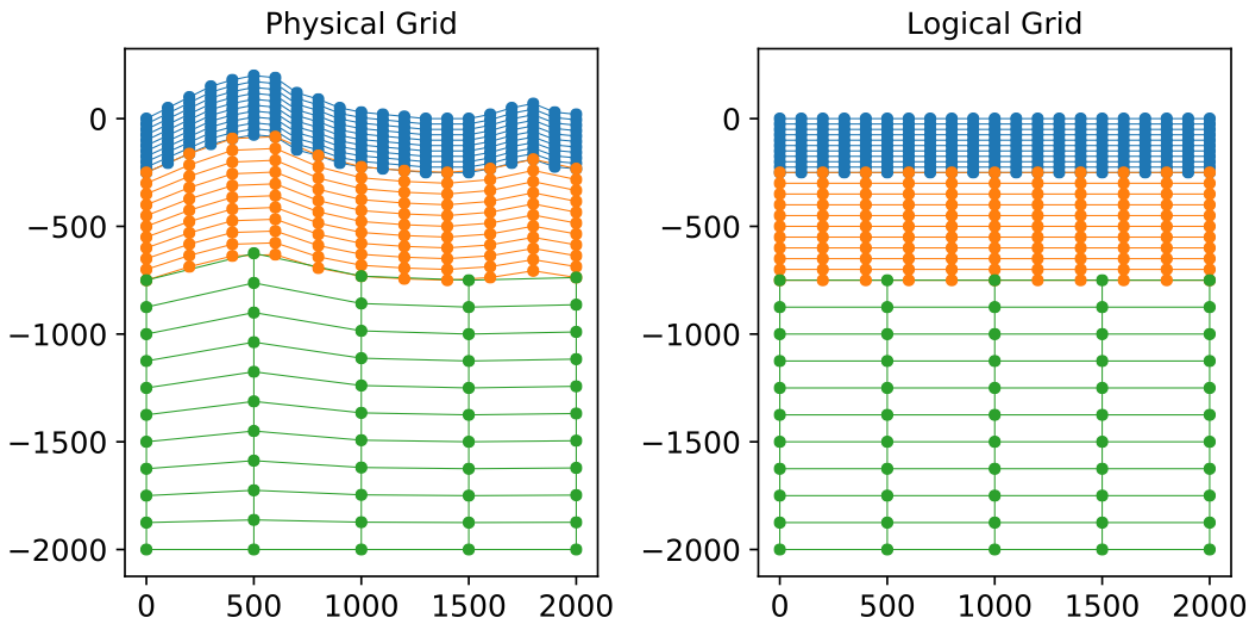

Fig. 2.1: Diagram showing a vertical slice through a model with three blocks, each of uniform resolution. The physical grid (left) represents the model in physical space and is warped to conform to topography. The logicl grid (right) represents the model in reference space and is used for fast, efficient queries of points.

Data layout

We use a Cartesian coordinate system (Fig. 2.2) for the model logical space with the origin at the top southwest corner (when the y axis is aligned with north) as shown in the figure. We measure the rotation of the coordinate system for the model logical space using the azimuth of the y direction relative to north (`y_azimuth` in the diagram).

For the logical space 4D arrays the x, y, and z indices are zero at the origin. The x index increases in the +x direction, the y index increases in the +y direction, and the z index increases in the -z direction.

HDF5 Storage Scheme

The model is stored in an HDF5 file. Fig. 2.3 shows a schematic of the data layout. If the top surface of the model is not a flat surface at sea level, then a `surfaces/top_surface` dataset provides the elevation of the top surface. Models may also include a `surfaces/topography_bathymetry` dataset to define topography and bathymetry, which corresponds to the top of the solid surface. The grids are stored in the `blocks` group. Attributes are included at the root level as well as for each dataset to yield a self-describing model. That is, no additional metadata is needed to define the model.

Model Metadata

Description

- **title** *(string)* Title of the model.
- **id** *(string)* Model identifier.
- **description** *(string)* Description of the model.
- **keywords** *(array of strings)* Keywords describing the model.

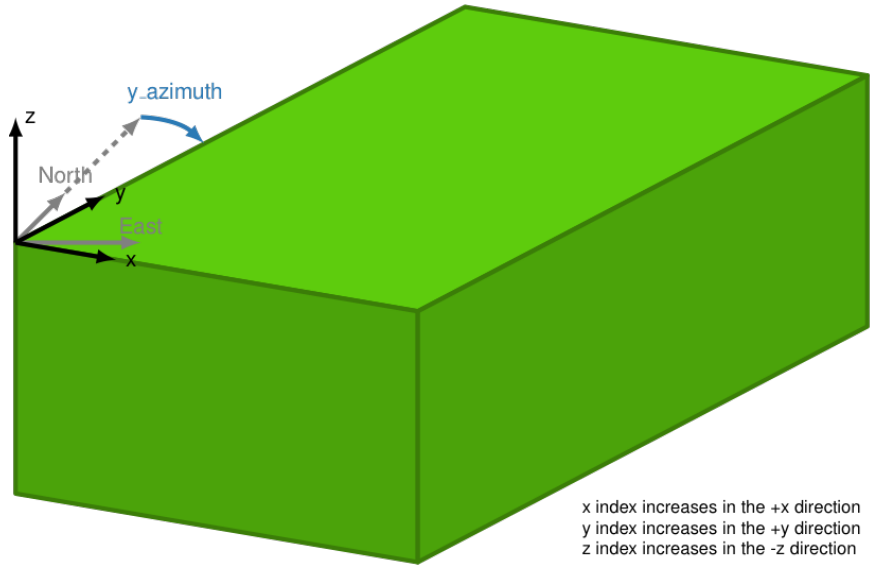

Fig. 2.2: Coordinate sytem for the logical grid with the origin in the top southwest corner. The logical grid may be rotated relative to the georeferenced coordinate system.

- **creator_name** *(string)* Name of person who created the HDF5 version of the model.
- **creator_institution** *(string)* Institution/organization of model creator.
- **creator_email** *(string)* Email of model creator.
- **acknowledgements** *(string)* acknowledgements for the model.
- **authors** *(array of strings)* Model authors.
- **references** *(array of strings)* Publications describing the model.
- **doi** *(string)* Digital Object Identifier for model.
- **version** *(string)* Version of model.
- **auxiliary** *(string, optional)* Auxiliary metadata in json format.

Model Data

- **data_values** *(array of strings)* Names of values in model grids.
- **data_units** *(array of strings)* Units of values in model grids.
- **data_layout** *(string)* `vertex` for vertex-based layout (values are specified at vertices) or `cell` for cell-based layout (values are specified at centers of grid cells). Currently only vertex-based values are supported.

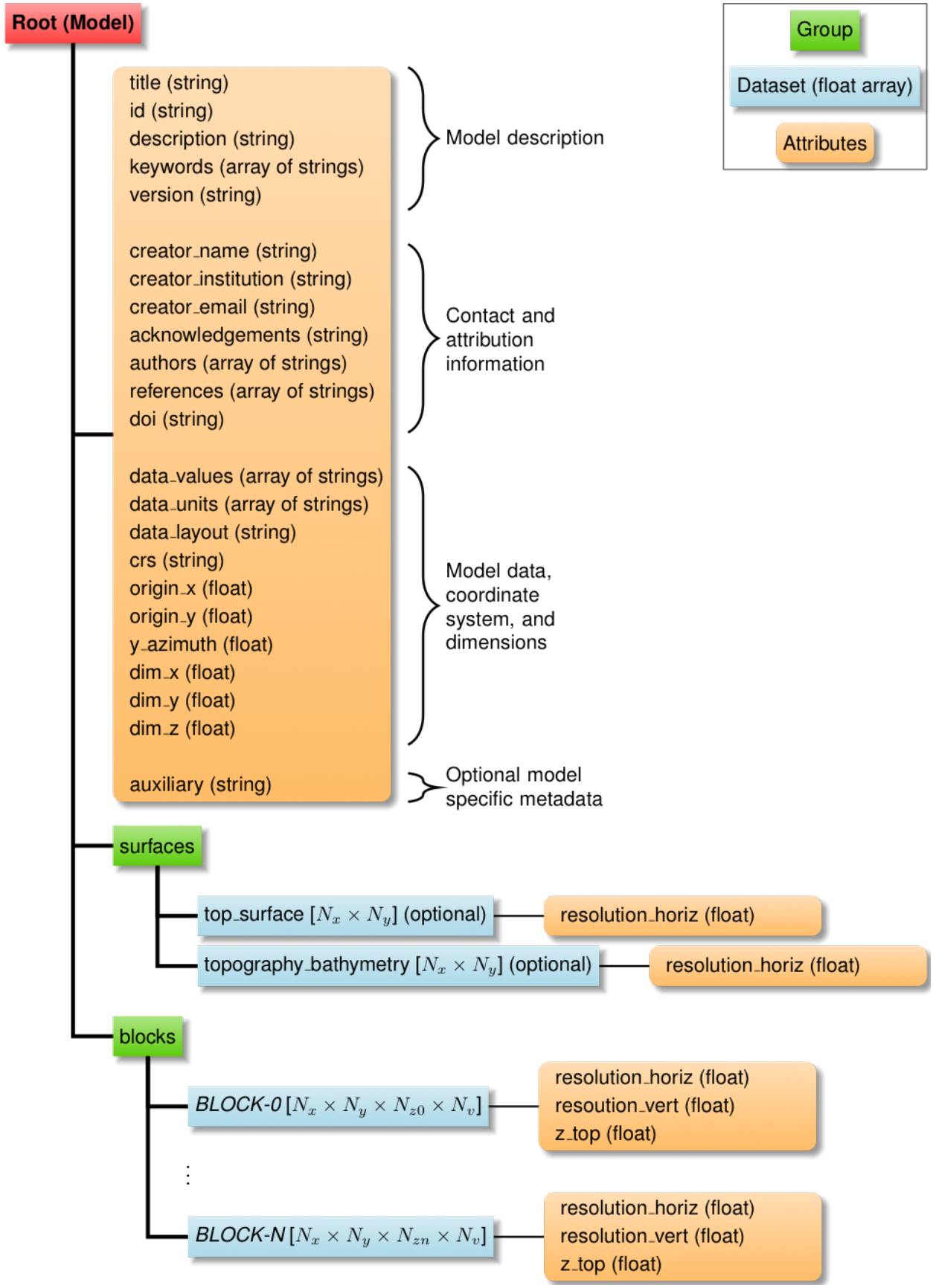

Fig. 2.3: Schematic of the HDF5 data layout for GeoModelGrids. Model values are stored as uniform grids within the `blocks` group. The elevation of surfaces are stored within the `surfaces` group. Attributes are attached at the root level and each dataset to provide all metadata. N_x is the number of points along the logical x axis. N_y is the number of points along the logical y axis. N_z is the number of points along the logical -z axis (downward). N_v is the number of values at each point.

Model Storage

- **crs** *(string)* Coordinate reference system (CRS) in Proj.4 format, WKT, or EPSG code.
- **origin_x** *(float)* X coordinate of origin of model in CRS coordinates.
- **origin_y** *(float)* Y coordinate of origin of model in CRS coordinates.
- **y_azimuth** *(float)* Azimuth (degrees clockwise from north) of y axis.
- **dim_x** *(float)* Dimension of model in local (rotated) x direction in units of CRS coordinates.
- **dim_y** *(float)* Dimension of model in local (rotated) y direction in units of CRS coordinates.
- **dim_z** *(float)* Dimension of model in local z direction in units of CRS coordinates.

Surface Metadata

- **resolution_horiz** *(float)* Horizontal resolution in units of CRS coordinates.

Block Metadata

- **resolution_horiz** *(float)* Horizontal resolution in units of CRS coordinates.
- **resolution_vert** *(float)* Vertical resolution in units of CRS coordinates.
- **z_top** *(float)* Z coordinate of top of block in topological space.

2.2.2 Getting Started

Release Notes

Version 1.0.0

Known issues

- The interpolation algorithm does not respect fault block and zone boundaries and the boundary between water and solid materials. This means fault block and zone ids will be interpolated across block and zone boundaries, and Vp and density will be interpolated Vp and density across the boundary between water and solid materials. Vs is not interpolated across the boundary between water and solid material, because it is not defined in water.
- GeoModelGrids does not support unit conversions. All values returned in queries are in the units of the underlying models. We do check that all models have consistent units for values contained in multiple models. Queries for the elevations of the top surface or topography/bathymetry will be returned in the units of the input Coordinate Reference System.
- The model storage has been optimized for faster successive queries in the vertical direction. That is, querying for points on a vertical slice will generally be faster than querying the same number of points on a horizontal slice.

Installation

There are two basic means of installing the GeoModelGrids software.

1. Use the Linux or macOS binary package.
2. Build from source.

The binary packages provide the command line programs. If you want to use the C/C++ API in your own code, then we recommend building from source to insure compatibility of the compiled code. For anyone wanting to contribute to GeoModelGrids software development, we offer a Docker development environment. Refer to *Developer Environment* for more information.

Installing the binary package

Note: The binary package contains the C/C++ query library along with its Python interface and command line programs for accessing GeoModelGrids files.

Binary packages are provided for Linux (GLIBC 2.17 and later; use `ldd --version` to check your version of GLIBC) and macOS for both the x86_64 architecture (10.14 and later) and arm64 architecture (11.0 and later). Windows users can use the Linux binary package within the Windows Subsystem for Linux. The binary packages are 200+ MB in size; the large size is a result of the inclusion of geographic coordinate datum information used by the Proj geographic projection library and Python.

Open a terminal window and change to the directory where you want to place the binary distribution.

```
cd  $HOME
mkdir geomodelgrids
cd geomodelgrids
```

Download the Linux or macOS tarball from GeoModelGrids GitHub releases and save it to the desired location, e.g., `$HOME/geomodelgrids`.

Unpack the tarball.

```
# Linux 64-bit
$ tar -xf geomodelgrids-1.0.0-Linux_x86_64.tar.gz

# macOS x86_64
$ tar -xf geomodelgrids-1.0.0-macOS_x86_64.tar.gz
```

Set environment variables. The provided `setup.sh` script in the top-level directory script only works if you are using the bash shell. If you are using a different shell, you may need to alter how the environment variables are set in `setup.sh` depending on your shell.

```
# Linux 64-bit
cd geomodelgrids-1.0.0-Linux_x86_64

# macOS
cd geomodelgrids-1.0.0-macOS_x86_64

source setup.sh
```

Important: You will need to either source the `setup.sh` script each time you open a new bash shell (terminal) window and want to use GeoModelGrids or add the environment variables to your shell setup script (for example, `.bashrc`).

Tip: To bypass macOS quarantine restrictions, simply use command line program `curl` to download the tarball from within a terminal rather than using a web browser.

```
curl -L -O https://github.com/baagaard-usgs/geomodelgrids/releases/download/v1.0.0/
↪geomodelgrids-1.0.0-macOS-10.15-x86_64.tar.gz
```

Alternatively, if you do download the tarball using a web browser, after you unpack the tarball you can remove the macOS quarantine flags using the following commands (requires Administrator access):

```
# Show extended attributes
xattr ./geomodelgrids-1.0.0-macOS-10.15-x86_64

# Output should be
com.apple.quarantine

# Remove quarantine attributes
sudo xattr -r -d com.apple.quarantine ./geomodelgrids-1.0.0-macOS-10.15-x86_64
```

Warning: The binary distribution contains the GeoModelGrids software and all of its dependencies (Proj, HDF5, OpenSSL, sqlite, curl, and tiff). If you have any of this software already installed on your system, you need to be careful in setting up your environment so that preexisting software does not conflict with the GeoModelGrids binary. By default the `setup.sh` script will prepend to the `PATH` (for Linux and macOS) and `LD_LIBRARY_PATH` (for Linux) environment variables. This will prevent most conflicts.

Installing from source

Prerequisites

Most Linux distributions can provide all of the prerequisites via a package manager. On macOS systems the operating system supplies with XCode command line tools installed supply the compiler, Sqlite, libtiff, openssl, and libcurl. You can use a package manager to install Proj and HDF5 or build them from source. You can also use the `build_binary.py` Python script in the `docker` directory of the GeoModelGrids source code to install the software and any prerequisites that you do not have.

- C/C++ compiler supporting C++11
- HDF5 (version 1.10.0 or later)
- Sqlite (version 3 or later; required by Proj)
- Proj (version 6.3.0 or later). Proj 7.0.0 and later also require:
  - libtiff
  - openssl
  - libcurl
- Python 3 (version 3.7 or later) with the following packages:

  - h5py
  - numpy
  - pybind11
  - netCDF4 (optional; for creating models from NetCDF files)
  - coverage (optional; for reporting test coverage)
- Catch2 (version 3.3.0 or later; if running test suite)

Downloading the source code

We highly recommend building code in a separate directory from the source tree. In the following configuration, we will unpack the source code in `$HOME/src`, build the code in `$HOME/build/geomodelgrids`, and install the code to `$HOME/geomodelgrids`.

Download the source code for the latest release from GeoModelGrids GitHub release and unpack the tarball.

```
mkdir $HOME/src
mv geomodelgrids-1.0.0.tar.gz $HOME/src
cd $HOME/src
tar -xf geomodelgrids-1.0.0.tar.gz
```

Running configure

Useful configure options (run `configure --help` to see all options):

- `--prefix=DIR` Install GeoModelGrids in directory `DIR`.
- `--enable-python` Enable building Python modules [default=no]
- `--enable-gdal` Enable GDAL support for writing GeoTiff files [default=no]
- `--enable-testing` Enable Python and C++ (requires Catch2) unit testing [default=no]
- `--with-catch2-incdir` Specify location of Catch2 header files [default=no]
- `--with-catch2-libdir` Specify location of Catch2 library [default=no]
- `--with-proj-incdir` Specify location of proj header files [default=no]
- `--with-proj-libdir` Specify location of proj library [default=no]
- `--with-hdf5-incdir` Specify location of hdf5 header files [default=no]
- `--with-hdf5-libdir` Specify location of hdf5 library [default=no]
- `--with-gdal-incdir` Specify location of gdal header files [default=no]
- `--with-gdal-libdir` Specify location of gdal library [default=no]

The Python interface for accessing or creating GeoModelGrids files requires configuring with `--enable-python`; we strongly recommend creating a separate Python virtual environment for geomodelgrids and installing all related dependencies and GeoModelGrids software into this virtual environment. Generating horizontal isosurfaces using `geomodelgrids_isosurface` requires the GDAL library and configuring with `--enable-gdal`.

```
# Create a directory where we will build geomodelgrids
mkdir -p $HOME/build/geomodelgrids
cd $HOME/build/geomodelgrids

# Configuration on a Linux system for C/C++ API with all dependencies installed
# by a package manager. We usually have to specify the location of HDF5 because
# multiple variations can be present.
$HOME/src/geomodelgrids-1.0.0/configure --prefix=$HOME/geomodelgrids \
--with-hdf5-incdir=/usr/include/hdf5/serial --with-hdf5-libdir=/usr/lib/x86_64-linux-gnu/
↪hdf5/serial

# Configuration on a macOS system for C/C++ API with HDF5 installed in $HOME/hdf5
# and Proj installed in $HOME/proj.
$HOME/src/geomodelgrids-1.0.0/configure --prefix=$HOME/geomodelgrids \
--with-hdf5-incdir=$HOME/hdf5/include --with-hdf5-libdir=$HOME/hdf5/lib \
--with-proj-incdir=$HOME/proj/include --with-proj-libdir=$HOME/proj/lib
```

```
# We start by creating a Python virtual environment and some additional Python packages.
python3 -m venv $HOME/geomodelgrids
source $HOME/geomodelgrids/bin/activate
python3 -m pip install --upgrade pip setuptools
pip install numpy pybind11

# If generating GeoModelGrid files or accessing HDF5 files directly using h5py
pip install h5py

mkdir -p $HOME/build/geomodelgrids
cd $HOME/build/geomodelgrids

# Configuration on a Linux system for C/C++ API with all dependencies installed
# by a package manager. We usually have to specify the location of HDF5 because
# multiple variations can be present.
$HOME/src/geomodelgrids-1.0.0/configure --prefix=$HOME/geomodelgrids --enable-python \
--with-hdf5-incdir=/usr/include/hdf5/serial --with-hdf5-libdir=/usr/lib/x86_64-linux-gnu/
↪hdf5/serial

# Configuration on a macOS system for C/C++ API with HDF5 installed in $HOME/hdf5
# and Proj installed in $HOME/proj.
$HOME/src/geomodelgrids-1.0.0/configure --prefix=$HOME/geomodelgrids --enable-python \
--with-hdf5-incdir=$HOME/hdf5/include --with-hdf5-libdir=$HOME/hdf5/lib \
--with-proj-incdir=$HOME/proj/include --with-proj-libdir=$HOME/proj/lib
```

Building and installing

```
make && make install
```

Running tests (optional)

If GeoModelGrids is configured with `--enable-testing` (requires Catch2), then the test suite can be run via `make check`. If GeoModelGrids is configured with Python enabled, then `make check` will also run the unit tests for the Python code.

Setting environment variables

Set environment variables for use of the bash shell:

```
PATH=$HOME/geomodelgrids/bin:$PATH
export LD_LIBRARY_PATH=$HOME/geomodelgrids/lib:$LD_LIBRARY_PATH
```

Running the code

There are two main ways to run the code:

- Use the `geomodelgrids` suite of command line programs, or
- Call the API from your own code.

The `geomodelgrids` suite of command line programs targets end users who simply want to query for values in a model at a set of locations. For additional details on how this program works, see the "Command line programs" section.

We also provide an API for calling the query routines from external code. See the *Python API*, *C++ API*, and *C API* sections for more information.

2.2.3 User Guide

Features

Topography

In cases where you have your own topography model, which may or may not match the one provided in a GeoModelGrid model, we recommend transforming your vertical coordinates into the GeoModelGrids model space. For example, to keep the same relative position with repsect to the model top and bottom, the z coordinate passed to the GeoModelGrids query function, z_query, would be

$$z_{query} = z_{bot} + \frac{z_{user} - z_{bot}}{z_{top\_user} - z_{bot}}(z_{top} - z_{bot}),$$

where z_{top} is the elevation of the top of the GeoModelGrids model (computed using `queryTopElevation()`), z_{bot} is the elevation of the bottom of the GeoModelGrids model, z_{top_user} is the elevation from your topography model, and z_{user} is the elevation you want in your query.

Squashing

Some applications ignore topography and assume a flat Earth model. In such cases, you will likely want to query the model at elevations relative to the topographic surface. You can query for elevation and adjust your input values accordingly, or you can use squashing to have the top surface automatically squashed and stretched so that it is at z=0. The model is squashed and stretched over the elevation range from the minimum squashing elevation up to the top surface. Either the top surface or the topography/bathymetry surface can be used as the reference surface for squashing.

Important: Squashing distorts the geometry of the Earth structure above the minimum squashing elevation. The amount of distortion depends on the elevation of the top surface and the minimum squashing elevation. The distortion in the vertical direction is uniform over the entire depth range of a single vertical profile. The distortion in the horizontal direction depends on the gradient in the elevation of the reference surface with a larger gradient leading to larger distortion in the horizontal direction.

Tip: To minimize distortion in the horizontal and vertical directions, you will want to use a minimum squashing elevation that is equal to the bottom elevation of the model.

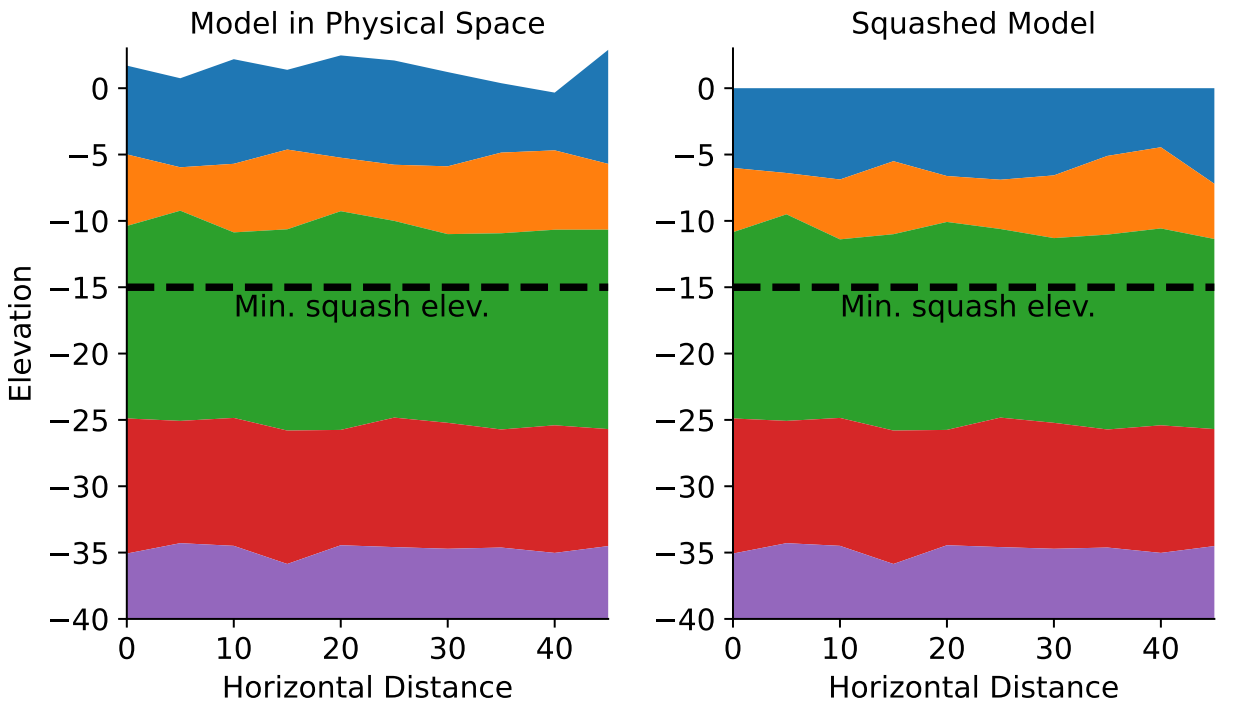

Fig. 2.4: Diagram showing the model in physical space (left) and squashed (right). Above the minimum squashing elevation of -15, we squash and stretch the model so that the top surface is at z=0. Below the minimum squashing elevation, the model is kept fixed.

Mapping between physical space and squashed model

The mappings between the physical space and squashed model are given by:

$$z_{physical} = z_{surf} + z_{squashed} \frac{z_{minsquash} - z_{surf}}{z_{minsquash}}, \text{ and} \tag{2.2}$$

$$z_{squashed} = \frac{z_{minsquash}}{z_{minsquash} - z_{surf}} (z_{physical} - z_{surf}) \tag{2.3}$$

Example

For an elevation of the top surface of 100 m and a minimum squashing elevation of -10 km, the following two queries are equivalent.

Listing 2.1: Query of model in physical space

```
geomodelgrids_query \
--models=my_model_with_topography.h5 \
--points=points_topo.in \
--output=points_topo.out

# Input: points_topo.in
37.455  -121.941   100.0
37.455  -121.941  -910.0
37.455  -121.941   -10.0e+3
37.455  -121.941   -20.0e+3
```

Listing 2.2: Query of model using squashing using the default surface (top surface) as the reference.

```
geomodelgrids_query \
--models=my_model_with_topography.h5 \
--points=points_flat.in \
--output=points_flat.out \
--min-squash-elev=-10.0e+3

# Input: points_flat.in
37.455  -121.941      0.0
37.455  -121.941     -1.0e+3
37.455  -121.941    -10.0e+3
37.455  -121.941    -20.0e+3
```

Algorithm for using squashing with topography/bathymetry

When using squashing with the topography/bathymetry surface, queries just below the surface may return `NODATA_VALUES` for Vs values. This happens when the query point lies in a grid cell that has some values above the topography/bathymetry surface and some below; the interpolation algorithm will return `NODATA_VALUES` if any values at the vertices of a grid cell are `NODATA_VALUES`. You can use the following algorithm to efficiently find the highest point that will return a valid value.

Nomenclature

- `dimZ`: total height of the model
- `zPhysical`: elevation (m) in physical space
- `zSquashed`: elevation (m) in squashed space
- `zMinSquashed`: minimum elevation (m) of squashing (recommended value is `zMinSquashed=-dimZ`)
- `zLogical`: elevation (m) in logical space (grid space)
- `zTop`: elevation (m) of the top of the model (topography and top of ocean)
- `zSurf`: elevation (m) of surface used in squashing (topography / bathymetry in this case)
- `dZ`: discretization size (m) in the vertical direction in the region being queried

Relationship between `zSquashed` and `zLogical`

Using the relationship between zSquashed and zPhysical,

```
zSquashed = zMinSquashed / (zMinSquashed - zSurf) * (zPhysical - zSurf)
```

and the relationship between zPhysical and zLogical,

```
zLogical = -dimZ * (zTop - zPhysical) / (zTop + dimZ)
```

we can derive relationships between zSquashed and zLogical:

```
zSquashed = zMinSquashed / (zMinSquashed - zSurf) * (zTop - zSurf + zLogical / dimZ *␣
↪(zTop + dimZ))
zLogical = (zSquashed * (zMinSquashed - zSurf) / zMinSquashed + zSurf - zTop) * (dimZ /␣
↪(zTop + dimZ))
```

Note: If `zSurf == zTop` (top of the model) and `zMinSquashed == -dimZ`, then `zLogical == zSquashed`. That is, the logical grid lines up with a squashed model. However, if `zSurf` is the elevation of the topography/bathymetry surface, then `zSurf != zTop`, so `zLogical != zSquashed` even if `zMinSquashed == -dimZ`.

Pseudocode

1. Query for values using squashing and the desired elevation, for example, to get a value at the "surface", you could use an elevation of -1.0 m (1 m below the topography/bathymetry surface). This is zSquashed.
2. If the Vs value returned is `NODATA_VALUE`, then adjust the elevation and query the model again.
    1. Compute `zLogical` from `zSquashed` using `zLogical = (zSquashed * (zMinSquashed - zSurf) / zMinSquashed + zSurf - zTop) * (dimZ / (zTop + dimZ))`
    2. Update `zLogical` to be 1 grid point lower using `zLogical = floor(zLogical/dZ)*dZ`
    3. Compute an updated `zSquashed` from `zLogical` using `zSquashed = zMinSquashed / (zMinSquashed - zSurf) * (zTop - zSurf + zLogical / dimZ * (zTop + dimZ))`
    4. Query the model using this updated `zSquashed`.
    5. While Vs is `NODATA_VALUE`, do

1. Reduce `zLogical` by `dZ`, `zLogical -= dZ`
2. Update `zSquashed` using `zSquashed = zMinSquashed / (zMinSquashed - zSurf) * (zTop - zSurf + zLogical / dimZ * (zTop + dimZ))`
3. Query the mode using this update `zSquashed`

Command Line Programs

geomodelgrids_info

The `geomodelgrids_info` command line program is used to print information about a model to stdout and verify a model conforms to the GeoModelGrids specification.

Synopsis

Optional command line arguments are in square brackets.

```
geomodelgrids_info [--help]
  --models=FILE_0,...,FILE_M
  [--verify]
  [--description]
  [--coordsys]
  [--values]
  [--blocks]
  [--all]
```

Required arguments

- **–models=FILE_0,…,FILE_M** Names of `M` model files to examine. The information for each model is printed to stdout.

Optional arguments

- **–help** Print help information to stdout and exit.
- **–verify** Verify a model conforms to the GeoModelGrids specification.
- **–description** Display model description.
- **–blocks** Display block grid information.
- **–coordsys** Display coordinate system information.
- **–values** Display names and units of values stored in the model.
- **–all** Display description, coordinate system, values, and blocks.

Verification

Verification includes:

- checking for required metadata (HDF5 attributes);
- verifying that the topography and blocks span the horizontal dimensions; and
- verifying that the blocks span the vertical dimension of the domain.

Example

Show all information for a model with three blocks and topography. The file is `three-blocks-topo.h5` in the `tests/data` directory.

```
geomodelgrids_info --all --models=tests/data/three-blocks-topo.h5

# Output
Model: tests/data/three-blocks-topo.h5
    Verification
        Verifying metadata...OK
        Verifying model coordinate system ...OK
        Verifying surface 'top surface'...OK
        Verifying surface 'topography/bathymetry'...OK
        Verifying resolution of top surface matches resolution of topography/bathymetry..
↪.OK
        Verifying block 'top'...OK
        Verifying block 'middle'...OK
        Verifying block 'bottom'...OK
        Verifying blocks span vertical dimension of domain...OK
    Title: Three Blocks Topo
    Id: three-blocks-topo
    Description: Model with three blocks and topography.
    Keywords: key one, key two, key three
    History: First version
    Comment: One comment
    Creator: John Doe, Agency, johndoe@agency.org
    Authors: Smith, Jim; Doe, John; Doyle, Sarah
    References:
        Reference 1
        Reference 2
    Acknowledgement: Thank you!
    Repository: Some repository http://somewhere.org
    DOI: this.is.a.doi
    Version: 1.0.0
    License: CC0
    Dimensions of model: x=60000, y=120000, z=45000
    Bounding box (WGS84): (34.3954, -117.8241) (34.6535, -117.2496) (35.603, -117.8789)␣
↪(35.342, -118.4583)
    Coordinate system:
        CRS (PROJ, EPSG, WKT): EPSG:3311
        Coordinate system units: x=meter, y=meter, z=meter (assumed)
        Origin: x=200000, y=-400000
        Azimuth (degrees) of y axis from north: 330
```

(continues on next page)

(continued from previous page)

```
    Values stored in model:
        0: one (m)
        1: two (m/s)
        Vertex-based data
    Surfaces
        Top surface:
            Number of points: x=13, y=25
            Resolution: x=5000, y=5000
        Topography/bathymetry:
            Number of points: x=13, y=25
            Resolution: x=5000, y=5000
    Blocks (3)
        Block 'top'
            Resolution: x=10000, y=10000, z=5000
            Elevation of top of block in logical space: 0
            Number of points: x=7, y=13, z=2
            Dimensions: x=60000, y=120000, z=5000
        Block 'middle'
            Resolution: x=20000, y=20000, z=10000
            Elevation of top of block in logical space: -5000
            Number of points: x=4, y=7, z=3
            Dimensions: x=60000, y=120000, z=20000
        Block 'bottom'
            Resolution: x=30000, y=30000, z=10000
            Elevation of top of block in logical space: -25000
            Number of points: x=3, y=5, z=3
            Dimensions: x=60000, y=120000, z=20000```
```

geomodelgrids_query

The `geomodelgrids_query` command line program is used to query for values of the model at a set of points.

The query values will be interpolated from the model using trilinear interpolation (interpolation along each model axis). If any value in the interpolation is equal to NODATA_VALUE, then the value returned will be NODATA_VALUE.

Synopsis

Optional command line arguments are in square brackets.

```
geomodelgrids_query [--help] [--log=FILE_LOG]
  --values=VALUE_0,...,VALUE_N
  --models=FILE_0,...,FILE_M
  --points=FILE_POINTS
  --output=FILE_OUTPUT
  [--squash-min-elev=ELEV]
  [--squash-surface=SURFACE]
  [--points-coordsys=PROJ|EPSG|WKT]
```

Required arguments

- **–values=VALUE_0,…,VALUE_N** Names of `N` values to be returned in query. Values will be returned in the order specified.
- **–models=FILE_0,…,FILE_M** Names of `M` model files to query. For each point the models are queried in the order given until a model is found that contains value(s) the point.
- **–points=FILE_POINTS** Name of file with a list of input points. The format is whitespace separated columns of x, y, z in the user specified coordinate reference system.
- **–output=FILE_OUTPUT** Name of file for output values. The format is whitespace separated columns of the input coordinates and `VALUE_0`, ..., `VALUE_N`.

Optional arguments

- **–help** Print help information to stdout and exit.
- **–log=FILE_LOG** Name of file for logging.
- **–squash-min-elev=ELEV** Top of the model is squashed/stretched to z=0 with the model below z=`ELEV` held fixed (default=-10.0e+3). See *Squashing* for more information.
- **–squash-surface=SURFACE** Surface to use as a vertical reference for computing depth. Valid values for `SURFACE` include `top_surface` (default), `topography_bathymetry`, and `none` (disables squashing).
- **–points-coordsys=PROJ|EPSG|WKT** Coordinate reference system of input points as Proj parameters, EPSG code, or Well-Known Text. Default is EPSG:4326 (latitude, WGS84 degrees; longitude, WGS84 degrees; elevation, m above ellipsoid.

New in v1.0.0

The default value for the minimum squashing elevation has been changed from 0 to -10.0e+3 (-10 km). This value works well for many applications with topography of about 1-2 km. You can enable squashing with this default value using the command line argument `--squash-surface`.

Output file

The output file contains a one line header with the command used to generate the file. The header is followed by lines with columns of the input coordinates and the values (in the order they were specified on the command line).

Examples

The input files for these examples are located in `tests/data`. See *Squashing* for examples illustrating how to use squashing.

Query with input points in WGS84

Query the model `one-block-flat.h5` for a single value `two` at three points given in file `one-block-flat_latlon.in` with the output written to file `one-block-flat_latlon.out`. We use the default coordinate system for the input points, which is latitude, longitude, elevation with the WGS84 horizontal datum and meters above the ellipsoid for the vertical datum.

```
geomodelgrids_query \
--models=tests/data/one-block-flat.h5 \
--points=tests/data/one-block-flat_latlon.in \
--output=tests/data/one-block-flat_latlon.out \
--values=two

# Input: one-block-flat_latlon.in
37.455  -121.941   0.0
37.479  -121.734  -5.0e+3
37.381  -121.581  -3.0e+3

# Output: one-block-flat_latlon.out, latitude (deg), longitude (deg), elevation (m), two␣
↪(m/s)
# geomodelgrids_query --models=one-block-flat.h5 --points=one-block-flat_latlon.in --
↪output=one-block-flat_latlon.out --values=two
#           x0            x1            x2           two
  3.745500e+01 -1.219410e+02  0.000000e+00 -1.520064e+03
  3.747900e+01 -1.217340e+02 -5.000000e+03  1.853648e+04
  3.738100e+01 -1.215810e+02 -3.000000e+03  7.266073e+03
```

Query with input points in UTM

Query the model `one-block-flat.h5` for values `one` and `two` at the same three points as the previous example given in file `one-block-flat_utm.in` with the output written to file `one-block-flat_utm.out`. In this example, we provide the points in UTM zone 10 coordinates, which corresponds to EPSG:26910.

```
geomodelgrids_query \
--models=tests/data/one-block-flat.h5 \
--points=tests/data/one-block-flat_utm.in \
--output=tests/data/one-block-flat_utm.out \
--values=one,two \
--points-coordsys=EPSG:26910

# Input: one-block-flat_utm.in
593662.64        4145875.37     0.00
611935.55        4148764.09 -5000.00
625627.70        4138083.87 -3000.00

# Output: one-block-flat_utm.out, easting (m), northing (m), elevation (m), one (m), two␣
↪(m/s)
# geomodelgrids_query --models=one-block-flat.h5 --points=one-block-flat_utm.in --
↪output=one-block-flat_utm.out --values=one,two --points-coordsys=EPSG:26910
#           x0            x1            x2           one           two
  5.936626e+05  4.145875e+06  0.000000e+00  4.702445e+03 -1.520065e+03
```

(continues on next page)

(continued from previous page)

```
  6.119356e+05  4.148764e+06 -5.000000e+03  3.114499e+04  1.853648e+04
  6.256277e+05  4.138084e+06 -3.000000e+03  3.182592e+04  7.266073e+03
```

geomodelgrids_queryelev

The `geomodelgrids_queryelev` command line program is used to query for the elevation of the top surface of the model or the topography/bathymetry surface at a set of points in a model.

The elevation will be interpolated from the raster surface grid using bilinear interpolation (interpolation along each model axis).

Synopsis

Optional command line arguments are in square brackets.

```
geomodelgrids_queryelev [--help] [--log=FILE_LOG]
  --models=FILE_0,...,FILE_M
  --points=FILE_POINTS
  --output=FILE_OUTPUT
  [--surface=SURFACE]
  [--points-coordsys=PROJ|EPSG|WKT]
```

Required arguments

- **–models=FILE_0,…,FILE_M** Names of `M` model files to query. For each point the models are queried in the order given until a model is found that contains value(s) the point.
- **–points=FILE_POINTS** Name of file with a list of input points. The format is whitespace separated columns of x, y in the user specified coordinate reference system.
- **–output=FILE_OUTPUT** Name of file for output values. The format is whitespace separated columns of the input coordinates and elevation.

Optional arguments

- **–help** Print help information to stdout and exit.
- **–log=FILE_LOG** Name of file for logging.
- **–surface=SURFACE** Name of surface to query; `top_surface` (default) or `topography_bathymetry`.
- **–points-coordsys=PROJ|EPSG|WKT** Coordinate reference system of input points as Proj parameters, EPSG code, or Well-Known Text. Default is EPSG:4326 (latitude, WGS84 degrees; longitude, WGS84 degrees; elevation, m above ellipsoid.

Output file

The output file contains a one line header with the command used to generate the file. The header is followed by lines with columns of the input coordinates and the elevation of the ground surface in the model coordinate system.

Example

The input files for these examples are located in `tests/data`.

Query the model `one-block-topo.h5` for the elevation of the ground surface at three points given in file `one-block-flat_elev.in` with the output written to file `one-block-flat_elev.out`. We use the default coordinate system for the input points, which is latitude and longitude in the WGS84 horizontal datum. The elevation of the ground surface is given in meters above the WGS84 ellipsoid.

```
geomodelgrids_queryelev \
--models=tests/data/one-block-topo.h5 \
--points=tests/data/one-block-topo_elev.in \
--output=tests/data/one-block-flat_elev.out

# Input: one-block-topo_elev.in
37.455  -121.941
37.479  -121.734
37.381  -121.581

# Output: one-block-topo_latlon.out, latitude (deg), longitude (deg), elevation (m)
# geomodelgrids_queryelev --models=tests/data/one-block-topo.h5 --points=tests/data/one-
↪block-topo_elev.in --output=tests/data/one-block-flat_elev.out
#           x0            x1     elevation
  3.745500e+01 -1.219410e+02  1.500461e+02
  3.747900e+01 -1.217340e+02  1.497750e+02
  3.738100e+01 -1.215810e+02  1.500231e+02
```

geomodelgrids_borehole

The `geomodelgrids_borehole` command line program is used to query for values of the model at a virtual borehole. This program provides a higher-level interface for this common, specific query compared to using `geomodelgrids_query`.

The points in the borehole will start at the surface and go down to the specified maximum depth. The query values will be interpolated from the model using trilinear interpolation (interpolation along each model axis).

Synopsis

Optional command line arguments are in square brackets.

```
geomodelgrids_borehole [--help] [--log=FILE_LOG]
  --location=X,Y
  --values=VALUE_0,...,VALUE_N
  --models=FILE_0,...,FILE_M
  --output=FILE_OUTPUT
  [--max-depth=DEPTH]
```

(continues on next page)

(continued from previous page)

```
  [--dz=RESOLUTION]
  [--points-coordsys=PROJ|EPSG|WKT]
```

Required arguments

- **–location=X,Y** Location of virtual borehole in points coordinate system.
- **–values=VALUE_0,...,VALUE_N** Names of `N` values to be returned in query. Values will be returned in the order specified.
- **–models=FILE_0,...,FILE_M** Names of `M` model files to query. For each point the models are queried in the order given until a model is found that contains value(s) the point.
- **–output=FILE_OUTPUT** Name of file for output values. The format is whitespace separated columns of the input coordinates and `VALUE_0`, ..., `VALUE_N`.

Optional arguments

- **–help** Print help information to stdout and exit.
- **–log=FILE_LOG** Name of file for logging.
- **–max-depth=DEPTH** Depth extent of virtual borehole in point coordinate system vertical units (default=5000m).
- **–dz=RESOLUTION** Vertical resolution of query points in virtual borehole in point coordinate system vertical units (default=10m).
- **–points-coordsys=PROJ|EPSG|WKT** Coordinate reference system of input points as Proj parameters, EPSG code, or Well-Known Text. Default is EPSG:4326 (latitude, WGS84 degrees; longitude, WGS84 degrees; elevation, m above ellipsoid.

Output file

The output file contains a two line header with the command used to generate the file and column headings. The header is followed by lines with columns of the elevation and depth in the points coordinate system and the values (in the order they were specified on the command line).

Example

Query the model `three-blocks-topo.h5` for value `two` in a virtual borehole at a point given in UTM coordinates with the output written to file `three-blocks-topo_borehole.out`. In this example, we provide the location of the virtual borehole in UTM zone 11 coordinates, which corresponds to EPSG:26911. We probe to a maximum depth of 20 km with a sampling interval of 2 km.

The model file for this example is located in `tests/data`.

```
geomodelgrids_borehole \
--models=tests/data/three-blocks-topo.h5 \
--location=436201.11,3884356.88 \
--max-depth=20.0e+3 \
--dz=2000.0 \
```

(continues on next page)

(continued from previous page)

```
--output=tests/data/three-blocks-topo_borehole.out \
--values=two \
--points-coordsys=EPSG:26911

# Contents of tests/data/three-blocks-topo_borehole.out:
# geomodelgrids_borehole --models=tests/data/three-blocks-topo.h5 --location=436201.11,
↪3884356.88 --max-depth=20.0e+3 --dz=2000.0 --output=tests/data/three-blocks-topo_
↪borehole.out --values=two --points-coordsys=EPSG:26911
#    Elevation         Depth           two
  1.516904e+02  0.000000e+00 -9.093030e+03
 -1.848310e+03  2.000000e+03  4.747184e+02
 -3.848310e+03  4.000000e+03  1.004247e+04
 -5.848310e+03  6.000000e+03  1.961021e+04
 -7.848310e+03  8.000000e+03  2.917796e+04
 -9.848310e+03  1.000000e+04  3.874571e+04
 -1.184831e+04  1.200000e+04  4.831346e+04
 -1.384831e+04  1.400000e+04  5.788121e+04
 -1.584831e+04  1.600000e+04  6.744896e+04
 -1.784831e+04  1.800000e+04  7.701670e+04
 -1.984831e+04  2.000000e+04  8.658445e+04
```

geomodelgrids_isosurface

Note: This application requires the GDAL library and is not included in the binary package.

The `geomodelgrids_isosurface` command line program is used to generate horizontal isosurfaces for model values. The isosurface values are the depth from the reference surface; the default surface is `topography_bathymetry`. Currently, only output as GeoTiff raster image files is supported. The isosurface values are computed at the center of each pixel in the raster image.

The isosurfaces are found using a multigrid line search with the number of points between the surface and the specified maximum depth (`--max-depth=DEPTH` in the units of the model coordinate system) given by `--num-search-points`. The default direction of the line search is shallow to deep; this can be reversed using the `--prefer-deep` command line argument. The same number of points are used at each level of refinement with the resolution of the maximum level of refinement given by `--vresolution=RESOLUTION` (default=10.0) in the model vertical coordinate system. After the line search at the finest resolution, the depth of the isosurface is found using linear interpolation.

Synopsis

Optional command line arguments are in square brackets.

```
geomodelgrids_isosurface [--help] [--log=FILE_LOG]
  --bbox=XMIN,XMAX,YMIN,YMAX
  --hresolution=RESOLUTION
  --isosurface=NAME,VALUE
  --max-depth=DEPTH
  --models=FILE_0,...,FILE_M
  --output=FILE_OUTPUT
  [--depth-reference=SURFACE]
```

(continues on next page)

(continued from previous page)

```
  [--num-search-points=NUM]
  [--vresolution=RESOLUTION]
  [--prefer-deep]
  [--bbox-coordsys=PROJ|EPSG|WKT]
```

Required arguments

- **–bbox=XMIN,XMAX,YMIN,YMAX** Bounding box for isosurface.
- **–hresolution=RESOLUTION** Horizontal resolution of isosurface.
- **–isosurface=NAME,VALUE** Name and values for isosurfaces (default=Vs,1.0 and Vs,2.5; repeat for multiple values).
- **–max-depth=DEPTH** Maximum depth allowed for isosurface.
- **–models=FILE_0,...,FILE_M** Models to query (in order).
- **–output=FILE_OUTPUT** Name of file for isosurface values.

Optional arguments

- **–help** Print help information to stdout and exit.
- **–log=FILE_LOG** Name of file for logging.
- **–depth-reference=SURFACE** Surface to use for calculating depth (default=`topography_bathymetry`)
- **–num-search-points=NUM** Number of search points in each iteration (default=10).
- **–vresolution=RESOLUTION** Vertical resolution for depth of isosurface (default=10.0).
- **–prefer-deep** Prefer deepest elevation for isosurface rather than shallowest (default=shallowest).
- **–bbox-coordsys=PROJ|EPSG|WKT** Coordinate system for isosurface points as Proj parameters, EPSG code, or Well-Known Text. Default is EPSG:4326 (latitude, WGS84 degrees; longitude, WGS84 degrees; elevation, m above ellipsoid.

Output file

The output is a raster grid with two bands stored as a GeoTiff file. The GeoTiff files includes the geographic coordinate system information as Well-Known-Text (WKT) along with the labels for the bands in the form `NAME=VALUE`, where `NAME` is the name of the model value and `VALUE` is the isosurface value. The GeoTiff file can be read using a variety of open-source and commercial GIS software. Note that GeoTiff images can be loaded into many image viewers, but because the bands contain floating point numbers, they will not be rendered as conventional RGB images.

Example

Extract isosurfaces for values `one=12.0e+4` and `two=40.0e+3` from model `three-blocks-topo.h5` using a bounding box given in latitude and longitude in the WGS84 datum. We specify a horizontal resolution of 0.1 degrees and a vertical resolution of 500 meters for the finest scale line search. We specify the maximum search depth to be 45 km, which corresponds to the base of the model. The model for this example is located in `tests/data`.

```
geomodelgrids_isosurface \
--models=tests/data/three-blocks-topo.h5 \
--bbox=34.6,34.8,-117.7,-117.3 \
--hresolution=0.1 \
--vresolution=500.0 \
--isosurface=one,12.0e+4 \
--isosurface=two,40.0e+3 \
--max-depth=45.0e+3 \
--depth-reference=topography_bathymetry \
--output=tests/data/three-blocks-topo-isosurface.tiff \
--bbox-coordsys=EPSG:4326
```

We can see the metadata by running gdalinfo on the resulting GeoTiff file.

```
gdalinfo tests/data/three-blocks-topo-isosurface.tiff

# Output
Driver: GTiff/GeoTIFF
Files: tests/data/three-blocks-topo-isosurface.tiff
Size is 4, 2
Coordinate System is:
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    USAGE[
        SCOPE["unknown"],
        AREA["World"],
        BBOX[-90,-180,90,180]],
    ID["EPSG",4326]]
Data axis to CRS axis mapping: 2,1
Origin = (-117.700000000000003,34.799999999999997)
Pixel Size = (0.100000000000001,-0.099999999999998)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  COMPRESSION=DEFLATE
```

(continues on next page)

(continued from previous page)

```
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  (-117.7000000,  34.8000000) (117d42\' 0.00\"W, 34d48\' 0.00\"N)
Lower Left  (-117.7000000,  34.6000000) (117d42\' 0.00\"W, 34d36\' 0.00\"N)
Upper Right (-117.3000000,  34.8000000) (117d18\' 0.00\"W, 34d48\' 0.00\"N)
Lower Right (-117.3000000,  34.6000000) (117d18\' 0.00\"W, 34d36\' 0.00\"N)
Center      (-117.5000000,  34.7000000) (117d30\' 0.00\"W, 34d42\' 0.00\"N)
Band 1 Block=4x2 Type=Float32, ColorInterp=Gray
  Description = one=120000
  NoData Value=-1.0000000200408773e+20
Band 2 Block=4x2 Type=Float32, ColorInterp=Undefined
  Description = two=40000
  NoData Value=-1.0000000200408773e+20
```

geomodelgrids_create_model

Note: This application requires Python 3.7 or later and is not included in the binary package.

The `geomodelgrids_create_model` command line program is used to generate a model from a data source. The current data sources include CSV files, IRIS EMC files (vertex-based grids), and EarthVision.

Synopsis

Optional command line arguments are in square brackets.

```
geomodelgrids_create_model
  --config=CONFIG
  [--help]
  [--show-parameters]
  [--import-domain]
  [--import-surfaces]
  [--import-blocks]
  [--update-metadata]
  [--all]
  [--quiet]
  [--log=LOG_FILENAME]
  [--debug]
```

Required arguments

- **`--config=FILE_0,...,FILE_N`** Names of `N` model configuration files.

Optional arguments

- **--help** Print help information to stdout and exit.
- **--show-parameters** Print parameters to stdout.
- **--import-domain** Create domain.
- **--import-surfaces** Create surfaces.
- **--import-blocks** Create blocks.
- **--all** Equivalent to `--import-domain --import-surfaces --import-block`.
- **--update-metadata** Update all metadata in file using current model configuration.
- **--quiet** Turn off printing progress information to stdout.
- **--log=LOG_FILENAME** Name of file for logging output.
- **--debug** Log debugging information.

Hint: The `--update-metadata` option makes it possible to update the model, surface, and block metadata without altering the values. It is very useful for updating metadata associated with publication or archiving of the model, such as repository information, references, acknowledgement, history, and comment.

Model configuration files

The model configuration files use the Python Config syntax. In general, we denote lists using comma separated entries surrounded by square brackets. The files contain the following sections:

- **geomodelgrids** Model metadata.
- **coordsys** Model coordinate system.
- **data** Values stored in the model.
- **domain** Domain information.

The files also contain sections for the data source and sections corresponding to the surface and block names.

`geomodelgrids` parameters

- **title** *(string)* Title of model.
- **id** *(string)* Model identifier.
- **description** *(string)* Description of model.
- **keywords** *(array of string)* Comma separated list of keywords.
- **history** *(string)* History of model.
- **comment** *(string)* General comments for model.
- **creator_name** *(string)* Name of person who generated the GeoModelGrids file.
- **creator_email** *(string)* Email address of person who create the GeoModelGrids file.
- **creator_institution** *(string)* Institution of person who create the GeoModelGrids file.
- **acknowledgement** *(string)* Acknowledgement for the model.

- **authors** *(array of string)* | separated list of model authors.
- **references** *(array of string)* | separated list of publications associated with creation of the model that should be cited when using the model.
- **repository_name** *(string)* Name of repository holding model.
- **repository_url** *(string)* URL of repository holding model.
- **repository_doi** *(string)* Digital Object Identifier (DOI) of the model.
- **version** *(string)* Version number of model.
- **license** *(string)* Name of license for the model.
- **filename** *(string)* Filename for the generated model.
- **data_source** *(string)* Path to the Python DataSrc object used to generate the model. The path must be in the `PYTHONPATH`.

`coordsys` parameters

- **crs** *(string)* Proj compatible coordinate reference system (CRS). Valid values include Proj parameters, EPSG codes, and Well-Known Text (WKT).
- **origin_x** *(float)* X coordinate of the southwest corner in the CRS.
- **origin_y** *(float)* Y coordinate of the southwest corner in the CRS.
- **y_azimuth** *(float)* Azimuth of the y coordinate axes (clockwise from North).

`data` parameters

- **values** *(array of string)* Comma separated list of values stored in the model.
- **units** *(array of string)* Comma separated list of units of the values stored in the model. Units for dimensionless values are `none`.
- **layout** *(string)* `vertex` for vertex-based values (values are specified at coordinates of vertices) or `cell` for cell-based values (values are specified at center of grid cells). Currently only vertex-based values are supported.

`domain` parameters

- **dim_x** *(float)* Dimension of domain in x direction in units of CRS.
- **dim_y** *(float)* Dimension of domain in y direction in units of CRS.
- **dim_z** *(float)* Dimension of domain in z direction in units of CRS.
- **blocks** *(float)* Comma separated list of block names.
- **batch_size** *(integer)* Target number of points to use in a single batch when generating a model in pieces (avoids loading an entire model into memory).

`surface` parameters

Metadata for the `top_surface` and `topography_bathymetry` surfaces.

- **use_surface** *(boolean)* `True` to use surface, `False` to ignore.
- **chunk_size** *(array)* Tuple of 3 integer values for HDF5 chunk size. The chunk size cannot exceed the dataset size and should be in the range of 10 kilobytes to 1 megabyte.

Any axis can have either a uniform resolution grid or a variable resolution grid. The grid resolution need not be the same type along the axes.

Uniform resolution parameters

Parameters if a grid has uniform along corresponding coordinate axis.

- **resolution_x** *(float)* Horizontal resolution, in CRS units, along x axis.
- **resolution_y** *(float)* Horizontal resolution, in CRS units, along y axis.

Variable resolution parameters

Parameters if a grid has variable resolution along corresponding coordinate axis.

- **coordinates_x** *(float)* Coordinates, in CRS units, along x axis.
- **coordinates_y** *(float)* Coordinates, in CRS units, along y axis.

Important: Some data sources provide some of the parameters. In those cases, the parameter file need not include those parameters provided by the data source.

`block` parameters

Metadata for blocks in the model.

- **z_top** *(float)* Z coordinate, in CRS units, of the top of the block.
- **z_bot** *(float)* Z coordinate, in CRS units, of the bottom of the block.
- **z_offset** *(float)* Offset in z coordinate, in CRS units, for the top of the block applied to queries of the data source.
- **chunk_size** *(array)* Tuple of 4 integer values for HDF5 chunk size. The chunk size cannot exceed the dataset size and should be in the range of 10 kilobytes to 1 megabyte.

Uniform resolution parameters

Parameters if a grid has uniform along corresponding coordinate axis.

- **resolution_x** *(float)* Horizontal resolution, in CRS units, along x axis.
- **resolution_y** *(float)* Horizontal resolution, in CRS units, along y axis.
- **resolution_z** *(float)* Horizontal resolution, in CRS units, along z axis.

Variable resolution parameters

Parameters if a grid has variable resolution along corresponding coordinate axis.

- **coordinates_x** *(float)* Coordinates, in CRS units, along x axis.
- **coordinates_y** *(float)* Coordinates, in CRS units, along y axis.
- **coordinates_z** *(float)* Coordinates, in CRS units, along z axis.

Important: Some data sources provide some of the parameters. In those cases, the parameter file need not include those parameters provided by the data source.

Data sources

CSV Data Source

Data source for generating a GeoModelGrids model from a model provided as a CSV file.

`csv` parameters

- **filename** *(string)* Relative or absolute path of CSV file.
- **crs** *(str)* CRS of coordinates in CSV file.

`csv.columns` parameters

Mapping of columns in the CSV file to data values. The first column is 0.

- **x** *(integer)* Index of column with x coordinate values.
- **y** *(integer)* Index of column with y coordinate values.
- **z** *(integer)* Index of column with z coordinate values.
- **VALUE** *(integer)* Index of column with values for `VALUE`.

Warning: The CSV data source currently loads the entire CSV file into memory.

Example

This example illustrates use of the CSV data source to convert the Eberhart-Phillips and others 3D seismic velocity model from a CSV file to a GeoModelGrids model. The model grid has a variable resolution in each of the coordinate directions.

```
[geomodelgrids]
title = New Zealand Wide model 2.3 Qs and Qp models for New Zealand, updated for Kaikoura
id = nz-cvm
description = The Qs and Qp New Zealand Wide models 2.3 incorporate the results from the␣
↪Kaikoura region using aftershocks (Eberhart-Phillips, et al. 2021). The results from␣
```

(continues on next page)

(continued from previous page)

```
↪that study have been interpolated and merged into the previous New Zealand Wide model␣
↪2.2 (https://doi.org/10.5281/zenodo.3779523). There is no update of the Vp and Vp/Vs␣
↪models.
keywords = [seismic velocity model, New Zealand, tomography]
history =
comment =
version = 2.3
creator_name = Donna Eberhart-Phillips
creator_institution = GNS Science, UC Davis
creator_email = eberhartphillips@ucdavis.edu
acknowledgement =
authors = [ Eberhart-Phillips, Donna |  Bannister, Stephen |  Reyners, Martin |  Ellis,␣
↪Susan |  Lanza, Federica]
references = [Eberhart-Phillips, D., S. Ellis, F. Lanza, and S. Bannister (2021),␣
↪Heterogeneous material properties - as inferred from seismic attenuation - influenced␣
↪multiple fault rupture and ductile creep of the Kaikoura Mw 7.8 earthquake, New␣
↪Zealand, Geophys. J. Int., doi:10.1093/gji/ggab272.]
repository_name =
repository_url =
repository_doi =
license = Creative Commons Attribution 4.0 International Public License

filename = seismic_nz_cvm-2-3.h5
data_source = geomodelgrids.create.data_srcs.csv.datasrc.CSVFile

[coordsys]
crs = EPSG:2193
origin_x = 1739903.44
origin_y = 3685845.31
y_azimuth = 310.0

[csv]
filename = nz_cvm-2-3.txt
crs = EPSG:4979

[csv.columns]
x = 1
y = 0
z = 2
density = 3
Vp = 4
Vs = 5
Qp = 6
Qs = 7


[data]
values = [density, Vp, Vs, Qp, Qs]
units = [kg/m**3, m/s, m/s, None, None]

[domain]
dim_x = 2400.0e+3
```

(continues on next page)

(continued from previous page)

```
dim_y = 2400.0e+3
dim_z = 765.0e+3

blocks = [block]

[block]
x_coordinates = [0.0e+3, 464.0e+3, 514.0e+3, 561.0e+3, 600.0e+3, 633.0e+3, 663.0e+3, 693.
↪0e+3, 723.0e+3, 752.0e+3, 782.0e+3, 812.0e+3, 842.0e+3, 887.0e+3, 931.0e+3, 976.0e+3,␣
↪1003.0e+3, 1013.0e+3, 1021.0e+3, 1029.0e+3, 1037.0e+3, 1045.0e+3, 1055.0e+3, 1070.0e+3,
↪ 1085.0e+3, 1100.0e+3, 1115.0e+3, 1130.0e+3, 1145.0e+3, 1160.0e+3, 1170.0e+3, 1180.
↪0e+3, 1190.0e+3, 1200.0e+3, 1210.0e+3, 1220.0e+3, 1230.0e+3, 1240.0e+3, 1250.0e+3,␣
↪1260.0e+3, 1270.0e+3, 1280.0e+3, 1290.0e+3, 1305.0e+3, 1323.0e+3, 1348.0e+3, 1373.0e+3,
↪ 1398.0e+3, 1428.0e+3, 1453.0e+3, 1472.0e+3, 1491.0e+3, 1510.0e+3, 1529.0e+3, 1549.
↪0e+3, 1579.0e+3, 1609.0e+3, 1639.0e+3, 1668.0e+3, 1694.0e+3, 1715.0e+3, 1736.0e+3,␣
↪1754.0e+3, 1773.0e+3, 1790.0e+3, 1810.0e+3, 1832.0e+3, 1853.0e+3, 1903.0e+3, 2400.0e+3]
y_coordinates = [0.0e+3, 850.0e+3, 900.0e+3, 940.0e+3, 970.0e+3, 995.0e+3, 1022.0e+3,␣
↪1039.0e+3, 1048.0e+3, 1055.0e+3, 1061.0e+3, 1067.0e+3, 1073.0e+3, 1079.0e+3, 1082.0e+3,
↪ 1086.0e+3, 1089.0e+3, 1093.0e+3, 1096.0e+3, 1100.0e+3, 1103.0e+3, 1107.0e+3, 1110.
↪0e+3, 1114.0e+3, 1117.0e+3, 1121.0e+3, 1124.0e+3, 1128.0e+3, 1131.0e+3, 1135.0e+3,␣
↪1138.0e+3, 1142.0e+3, 1145.0e+3, 1149.0e+3, 1152.0e+3, 1156.0e+3, 1159.0e+3, 1163.0e+3,
↪ 1171.0e+3, 1178.0e+3, 1186.0e+3, 1193.0e+3, 1201.0e+3, 1208.0e+3, 1216.0e+3, 1223.
↪0e+3, 1231.0e+3, 1238.0e+3, 1246.0e+3, 1253.0e+3, 1261.0e+3, 1268.0e+3, 1276.0e+3,␣
↪1286.0e+3, 1296.0e+3, 1306.0e+3, 1321.0e+3, 1336.0e+3, 1351.0e+3, 1400.0e+3, 1500.0e+3,
↪ 1600.0e+3, 1740.0e+3, 2400.0e+3]
z_coordinates = [15000.0, 1000.0, -1000.0, -3000.0, -5000.0, -8000.0, -15000.0, -23000.0,
↪ -30000.0, -34000.0, -38000.0, -42000.0, -48000.0, -55000.0, -65000.0, -85000.0, -
↪105000.0, -130000.0, -155000.0, -185000.0, -225000.0, -275000.0, -370000.0, -620000.0,␣
↪-750000.0]
z_top_offset = 0.0
chunk_size = (18, 16, 25, 5)
```

IRIS EMC Data Source

Data source for generating a GeoModelGrids model from an IRIS EMC model. Currently, only conversion of vertex-based grids are supported.

`iris_emc` parameters

- **filename** *(string)* Relative or absolute path of IRIS EMC NetCDF file.

Example

This example illustrates use of the IRIS EMC data source to convert the Moulik and others S362ANI+M 3D seismic velocity model from the IRIS EMC format to a GeoModelGrids model. Most of the model metadata is extracted from the NetCDF file.

```
[geomodelgrids]
filename = seismic_moulik_etal_S362ANI+M.h5
data_source = geomodelgrids.create.data_srcs.iris_emc.datasrc.EMCNetCDF

repository_url = N/A
repository_name = None
repository_institution = None
repository_doi = None

[domain]
batch_size = 25000000

[block]
chunk_size = (15, 15, 25, 3)

[iris_emc]
filename = S362ANI+M_kmps.nc
```

EarthVision Data Source

Data source for generating a GeoModelGrids model from an EarthVision model.

`earthvision` parameters

- **model_dir** *(string)* Full absolute path to the directory containing the EarthVision model.
- **top_surface_2grd** *(str, optional)* Filename of `2grd` file for the top surface of the model.
- **topography_bathymetry_2grd** *(str, optional)* Filename of `2grd` file for the topography/bathymetry surface of the model.
- **geologic_model** *(string)* Filename of the sequence file corresponding to the EarthVision geologic model.
- **elev_units** *(string)* Units of elevation in the EarthVision geologic model.
- **xy_units** *(string)* Units of x and y coordinates in the EarthVision geologic model.
- **rules_module** *(string)* Path to the Python object used to assign material properties to the geologic units.
- **rules_pythonpath** `PYTHONPATH` for `rules_module`.

`earthvision.environment` parameters

Bash environment variables that include locations of the EarthVision files.

- **PATH** *(string)* Value of `PATH` environment variable for executables.
- **LD_LIBRARY_PATH** *(string)* Value of `LD_LIBRARY_PATH` environment variable for dynamic libraries.
- **LM_LICENSE_FILE** *(string)* Absolute path of the EarthVision license file.

Example

This example illustrates use of the EarthVision data source to generate a 3D seismic velocity model from a 3D geologic model in EarthVision. Rules are applied on each grid point to assign elastic properties based on the geologic unit and depth. The model includes two surfaces: one for the top of the model and one for the top of the solid material. The model includes two blocks, each with a uniform grid resolution.

```
[geomodelgrids]
title = USGS 3D seismic velocity model for the San Francisco Bay region (regional domain)
id = usgs-sfcvm-regional
description = USGS 3D seismic velocity model for the San Francisco Bay region (regional␣
↪domain)
keywords = [seismic velocity model, San Francisco Bay region]
history = Version 08.3.0 re-released as a GeoModelGrids model.
creator_name = Brad Aagaard
creator_institution = U.S. Geological Survey
creator_email = baagaard@usgs.gov
acknowledgement = None
authors = [Brocher, Thomas | Jachens, Robert | Simpson, Robert | Aagaard, Brad]
references = [None]
repository_name = ScienceBase
repository_url = https://sciencebase.gov
repository_doi = TBD
version = 21.0.0
license = CC0

filename = seismic_sfbay_regional-21.0.0.h5
data_source = geomodelgrids.create.earthvision.datasrc.RulesDataSrc

[coordsys]
crs = PROJCS["unnamed",GEOGCS["NAD83",DATUM["North_American_Datum_1983",SPHEROID["GRS␣
↪1980",6378137,298.257222101,AUTHORITY["EPSG","7019"]],TOWGS84[0,0,0,0,0,0,0],AUTHORITY[
↪"EPSG","6269"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.
↪0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4269"]],PROJECTION[
↪"Transverse_Mercator"],PARAMETER["latitude_of_origin",35],PARAMETER["central_meridian",
↪-123],PARAMETER["scale_factor",0.9996],PARAMETER["false_easting",0],PARAMETER["false_
↪northing",0],UNIT["Meter",1]]
origin_x = 97513.8
origin_y = 4562.2
y_azimuth = 323.638

[data]
values = [density, Vp, Vs, Qp, Qs, fault_block_id, zone_id]
units = [kg/m**3, m/s, m/s, None, None, None, None]
```

(continues on next page)

(continued from previous page)

```
[domain]
dim_x = 325.0e+3
dim_y = 645.0e+3
dim_z = 45.0e+3

blocks = [vres125m, vres250m]

batch_size = 2500000

[earthvision]
model_dir = /data/sfbay-geology/regional_domain
top_surface_2grd = [bigbay_reg_topo_km_prj.2grd, zero.2grd]
topography_bathymetry_2grd = [bigbay_reg_topo_km_prj.2grd]
geologic_model = CenCal_Reg_13b.seq
elev_units = km
xy_units = km
rules_fn = seismic_regional_08_03_00.get_rules
rules_pythonpath = .

[earthvision.environment]
PATH = /bin:/usr/bin:/sbin:/usr/sbin:/opt/earthvision/ev11/bin64
LD_LIBRARY_PATH = /opt/earthvision/ev11/lib64
LM_LICENSE_FILE = /opt/earthvision/license/license.dat

[top_surface]
use_surface = True
resolution_x = 500.0
resolution_y = 500.0
chunk_size = (150, 150, 1)

[topography_bathymetry]
use_surface = True
resolution_x = 500.0
resolution_y = 500.0
chunk_size = (150, 150, 1)

[vres125m]
resolution_x = 500.0
resolution_y = 500.0
resolution_z = 125.0
z_top = 0.0
z_bot = -10.0e+3
z_top_offset = -0.1
chunk_size = (12, 12, 81, 7)

[vres250m]
resolution_horiz = 1000.0
resolution_vert = 250.0
z_top = -10.0e+3
z_bot = -45.0e+3
z_top_offset = 0.0
```

(continues on next page)

(continued from previous page)

```
chunk_size = (8, 8, 141, 7)
```

Python API

Query

Classes

Query

Full name: geomodelgrids.Query

Enums

SquashingEnum

- **SQUASH_NONE** No squashing.
- **SQUASH_TOP_SURFACE** Squash model so top surface is flat.
- **SQUASH_TOPGRAPHY_BATHYMETRY** Squash model so topography/bathymetry surface is flat.

Constants

- **NODATA_VALUE** Value returned when query could not return a valid value.

Methods

Query()

Constructor.

initialize(models: list(Model), values: list(str), input_crs: str)

Perform initialization required to query the models.

- **models** List of model filenames (in query order).
- **values** List of names of values to return in queries.
- **input_crs** CRS as string (PROJ, EPSG, WKT) for input points.

set_squash_min_elev(min_elev: float)

Turn on squashing using the top surface and set the minimum elevation for squashing.

- **min_elev** Elevation in meters.

set_squashing(squash_type: SquashingEnum)

Set type of squashing.

- **squash_type** Squashing setting (SQUASH_NONE, SQUASH_TOP_SURFACE, SQUASH_TOPOGRAPHY_BATHYMETRY)

query_top_elevation(points: numpy.ndarray)

Query model for elevation of the top surface at a point using bilinear interpolation.

- **points** NumPy array [numPoints, 2[of point coordinates in input CRS.
- **returns** NumPy array of elevation (meters) of surface at each point.

query_topobathy_elevation(points: numpy.ndarray)

Query model for elevation of the topography/bathymetry surface at a point using bilinear interpolation.

- **points** NumPy array [numPoints, 2] of point coordinates in input CRS.
- **returns** NumPy array of elevation (meters) of surface at each point.

query(points: numpy.ndarray)

Query model for values at a point using bilinear interpolation

- **points** NumPy array [numPoints, 3] of point coordinates in input CRS.
- **returns** Tuple(values, status) where values is a NumPy array of model values at each point and status is a NumPy array with ErrorHandler.OK for a point if returning a valid value and ErrorHandler.WARNING for a point if unable to return a valid value.

Model

Full name: geomodelgrids.Model

Enums

ModelMode

- **READ** Read only mode.
- **READ_WRITE** Read/write mode.
- **READ_WRITE_TRUNCATE** Read/write mode, truncate upon opening.

DataLayout

- **VERTEX** Vertex-based data (values are specified at coordinates of vertices).
- **CELL** Cell-based data (values are specified at centers of grid cells).

Read-only Attributes

- **value_names** List of names of values in the model `list(str)`
- **value_units** List of units of values in the model `list(str)`
- **data_layout** Data layout for values (`VERTEX` or `CELL`)
- **dims** Model dimensions in meters [x, y, z] `list(float)`
- **origin** Coordinates of model origin [x, y] `list(float)` in model coordinate system
- **y_azimuth** Azimuth (degrees) of y coordinate axis `float`
- **crs** Coordinate reference system for model as Proj, EPSG, or WKT `str`

Methods

Model()

Constructor.

set_input_crs(value: str)

Set the coordinate reference system (CRS) of query input points. The default value is `EPSG:4326`, which corresponds to latitude and longitude in degrees in the WGS84 datum.

- **value** CRS of input points as string (PROJ, EPSG, WKT).

open(filename: str, mode: ModelMode)

Open the model for querying.

- **filename** Name of model file.
- **mode** File mode.

close()

Close the model after querying.

load_metadata()

Load model metadata.

initialize()

Initialize the model.

get_info()

Get model description information.

- **returns** Model description.

contains(points: numpy.ndarray)

Does model contain given point?

- **points** NumPy array [numPoints, 3] of point coordinates in input CRS.
- **returns** numpy.ndarray with True if model contains given point, False otherwise.

query_top_elevation(points: numpy.ndarray)

Query model for elevation of the top surface at a point using bilinear interpolation.

- **points** NumPy array [numPoints, 2[of point coordinates in input CRS.
- **returns** NumPy array of elevation (meters) of surface at each point.

query_topobathy_elevation(points: numpy.ndarray)

Query model for elevation of the topography/bathymetry surface at a point using bilinear interpolation.

- **points** NumPy array [numPoints, 2] of point coordinates in input CRS.
- **returns** NumPy array of elevation (meters) of surface at each point.

query(points: numpy.ndarray)

Query model for values at a point using bilinear interpolation

- **points** NumPy array [numPoints, 3] of point coordinates in input CRS.
- **returns** NumPy array of model values at each point.

ModelInfo

Full name: geomodelgrids.ModelInfo

Methods

ModelInfo()

Constructor.

Read-only Attributes

- **title** Title of model `str`
- **id** Model identifier `str`
- **description** Model description `str`
- **keywords** Keywords describing model `list(str)`
- **history** Model history `str`
- **comment** Comment about model `str`
- **creator_name** Name of creator `str`
- **creator_institution** Institution of creator `str`
- **creator_email** Email of creator `str`
- **acknowledgement** Acknowledgements for model `str`
- **authors** Author names `list(str)`
- **references** List of references `list(str)`
- **repository_name** Name of respository `str`
- **repository_url** URL of repository holding model `str`
- **repository_doi** DOI for model in repository `str`
- **version** Model version `str`

- **license** Name of license for model `str`
- **auxiliary** Auxiliary information stored as json `str`

ErrorHandler

Full name: geomodelgrids.ErrorHandler

Enums

StatusEnum

- **OK** No errors or warnings.
- **WARNING** Non-fatal error.
- **ERROR** Fatal error.

Methods

ErrorHandler()

Constructor.

reset_sttus()

Reset status to OK.

get_status()

Get status.

- **returns** Current status (OK, WARNING, ERROR).

Examples

Query Example

We provide a complete example Python script for querying the model with comments describing each step. The code can be run in the `tests` directory of the GeoModelGrids source tree.

```
# We use the `numpy` and `geomodelgrids` Python modules, so we must import them before␣
↪use.
import numpy
import geomodelgrids

# List of models to query (in order of precedence).
# Multiple models can be queried and they will be accessed in the order given.
```

(continues on next page)

(continued from previous page)

```
# Once values are found in one of the models, the rest of the models are skipped.
MODELS = (
        "data/one-block-topo.h5",
        "data/three-blocks-flat.h5",
)

# Values we want returned in queries (in order).
VALUES = ("two", "one")

# Coordinate system that is used for the points passed to the query function.
#
# The coordinate system is specified using a string corresponding to any
# coordinate reference system (CRS) recognized by the [Proj library](https://proj.org).
# This includes EPGS codes, Proj parameters, and Well-Known Text (WKT).
# In this case, we specify the coordinates as longitude, latitude, elevation
# in the WGS horizontal datum, which corresponds to `EPSG:4326`.
# Elevation is meters above the WGS84 ellipsoid.
CRS = "EPSG:4326"

# We create the query object and initialize it with the query parameters.
query = geomodelgrids.Query()
query.initialize(MODELS, VALUES, CRS)

# Coordinates of points for query (latitude, longitude, elevation)
# as a NumPy array with shape [numPoints, 3].
# The coordinate system is the one specified by the `CRS` variable.
points = numpy.array([
        [37.455, -121.941, 0.0],
        [37.479, -121.734, -5.0e+3],
        [37.381, -121.581, -3.0e+3],
        [37.283, -121.959, -1.5e+3],
        [37.262, -121.684, -4.0e+3],
])

(values, status) = query.query(points)

# Check that we have valid status (=0) for all points.
assert numpy.sum(status) == 0
```

MetaData Example

We provide a complete example Python script for accessing model metadata with comments describing each step. The code can be run in the `tests` directory of the GeoModelGrids source tree.

Tip: The *geomodelgrids_info* command line program generates a more complete description of the model metadata, including the details about the surfaces and blocks.

```
# We use the `geomodelgrids` Python modules, so we must import them before use.
import geomodelgrids
```

(continues on next page)

(continued from previous page)

```
# Model to examine.
FILENAME = "data/one-block-topo.h5"

# Create a model object and open the file as read only.
model = geomodelgrids.Model()
model.open(FILENAME, model.READ)

# Load the metadata but not the model contents.
model.load_metadata()

# Get and print model values, units, and data layout
print(f"Model values: {model.value_names}")
print(f"      units: {model.value_units}")
print(f"Data layout: {model.data_layout}")

# Get and print model dimensions, origin, y_azimuth, and coordinate system.
print(f"Model dimensions (m): {model.dims}")
print("Model coordinate system:")
print(f"    Origin: {model.origin}")
print(f"    Y azimuth: {model.y_azimuth}")
print(f"    CRS: {model.crs}")

# Get the model metadata
info = model.get_info()

# Get and print the title, id, description, version, license, keywords, history, and␣
↪comments.
print(f"Title: {info.title}")
print(f"Id: {info.id}")
print(f"Description: {info.description}")
print(f"Version: {info.version}")
print(f"License: {info.license}")
print(f"Keywords: {", ".join(info.keywords)}")
print(f"History: {info.history}")
print(f"Comments: {info.comment}")

# Get and print creator information
name = info.creator_name
institution = info.creator_institution
email = info.creator_email
print(f"Creator: {name}, {institution} ({email})")

# Get and print attribution information
print(f"Authors: {"; ".join(info.authors)}")
print(f"References:\n    {"\n    ".join(info.references)}")
print(f"Acknowledgements: {info.acknowledgement}")

# Get and print repository information
print(f"Repository: {info.repository_name}")
print(f"    URL: {info.repository_url}")
print(f"    DOI: {info.repository_doi}")
```

(continues on next page)

(continued from previous page)

```
# Get, parse, and print auxiliary information (dict).
if info.auxiliary:
    import json
    auxiliary = json.loads(info.auxiliary)
    print(f"Auxiliary: {auxiliary}")
```

Vertical Profiles Example

This example discusses a Python application that queries a 3D seismic velocity model for values on a vertical profile at seismic stations specified in a CSV file. The output is written to a directory with the vertical profile for each station in a different file. The command line interface for the Python script allows the user to specify which seismic velocity models in GeoModelGrids format to query, the name of the CSV file containing the information for the seismic stations, the maximum depth of the vertical profile, the resolution of the vertical profile, the name of the directory for the output, and whether to output the results as CSV files or text files (whitespace separated columns).

The source code is in `examples/python-api/station_profiles.py`. The Python script includes application class `App` with public methods `initialize()` and `run_query()` to set up the query and run the query, respectively. The Python script includes a function `cli()` that implements the command line interface.

```
# Show usage
./station_profiles.py --help

# Output
usage: station_profiles.py [-h] --models MODELS --filename FILENAME_LOCATIONS --output-
↪dir OUTPUT_DIR [--output-format {csv,txt}] --max-depth MAX_DEPTH
                           --resolution RESOLUTION [--log LOG_FILENAME] [--debug]

Application for running linear mixed effects regression on ground-motion records.

options:
  -h, --help            show this help message and exit
  --models MODELS       Comma separated list of GeoModelGrids files to query. (default:␣
↪None)
  --filename FILENAME_LOCATIONS
                        CSV file with Network, StationCode, StationLongitude, and␣
↪StationLatitude. (default: None)
  --output-dir OUTPUT_DIR
                        Directory for output. (default: None)
  --output-format {csv,txt}
                        Vertical resoluion in queries. (default: csv)
  --max-depth MAX_DEPTH
                        Maximum depth in kilometers. (default: None)
  --resolution RESOLUTION
                        Vertical resoluion in queries. (default: None)
  --log LOG_FILENAME
  --debug
```

Demonstration with the USGS SF-CVM

We demonstrate use of this Python application using the USGS San Francisco Bay region 3D seismic velocity model. We use version 21.1 of the model, `USGS_SFCVM_v21-1_detailed.h5`, which can be downloaded from USGS ScienceBase.

We create an input file `stations.csv` with information for two seismic stations, `BK.BKS` and `NC.JGR`. Station `BK.BKS` is located at 37.87622 degrees north and 122.23558 degrees west, and station `NC.JGR` is located at 37.51604 degrees north and 122.45815 degrees west in the WGS84 datum.

Listing 2.3: Contents of `stations.csv`.

```
Network,StationCode,StationLatitude,StationLongitude
BK,BKS,37.87622,-122.23558
NC,JGR,37.51604,-122.45815
```

We run the application, specifying the seismic velocity model, station information in `stations.csv`, output in CSV files in a `profiles` directory, and vertical profiles 1.0 km deep with a resolution of 25 m.

```
./station_profiles.py --models=USGS_SFCVM_v21-1_detailed.h5 --filename=stations.csv --
↪output-dir=profiles --max-depth=1.0 --resolution=25.0
```

The `profiles` directory contains files `BK.BKS.csv` and `NC.JGR.csv`. The values are all in SI units (`m` for elevation and depth, `kg/m**3` for density, and `m/s` for Vp and Vs).

Listing 2.4: Contents of `BK.BKS.csv` in the `profiles` output directory.

```
Elevation,Depth,Density,Vp,Vs,Qp,Qs
298.11,0.00,2240.71,2750.09,1194.08,172.73,86.37
273.11,25.00,2243.10,2761.81,1204.19,174.33,87.16
248.11,50.00,2245.49,2773.62,1214.38,175.94,87.97
223.11,75.00,2247.88,2785.44,1224.60,177.56,88.78
198.11,100.00,2250.26,2797.25,1234.83,179.19,89.59
173.11,125.00,2252.63,2809.06,1245.08,180.82,90.41
148.11,150.00,2254.99,2820.87,1255.35,182.45,91.23
123.11,175.00,2257.35,2832.69,1265.62,184.09,92.04
98.11,200.00,2259.70,2844.50,1275.92,185.73,92.87
73.11,225.00,2262.04,2856.31,1286.22,187.38,93.69
48.11,250.00,2264.38,2868.13,1296.54,189.03,94.52
23.11,275.00,2266.70,2879.94,1306.87,190.69,95.35
-1.89,300.00,2269.03,2891.75,1317.22,192.36,96.18
-26.89,325.00,2271.34,2903.56,1327.58,194.02,97.01
-51.89,350.00,2273.65,2915.38,1337.94,195.70,97.85
-76.89,375.00,2275.95,2927.19,1348.32,197.37,98.69
-101.89,400.00,2278.24,2939.00,1358.71,199.06,99.53
-126.89,425.00,2280.52,2950.81,1369.11,200.75,100.37
-151.89,450.00,2282.80,2962.63,1379.52,202.44,101.22
-176.89,475.00,2285.07,2974.44,1389.93,204.14,102.07
-201.89,500.00,2287.34,2986.25,1400.36,205.84,102.92
-226.89,525.00,2289.60,2998.07,1410.79,207.56,103.78
-251.89,550.00,2291.85,3009.88,1421.23,209.27,104.63
-276.89,575.00,2294.09,3021.69,1431.68,210.99,105.50
-301.89,600.00,2296.33,3033.50,1442.13,212.71,106.36
-326.89,625.00,2298.56,3045.32,1452.59,214.45,107.22
```

(continues on next page)

(continued from previous page)

```
-351.89,650.00,2300.79,3057.13,1463.06,216.18,108.09
-376.89,675.00,2303.01,3068.94,1473.53,217.93,108.96
-401.89,700.00,2305.22,3080.75,1484.00,219.67,109.84
-426.89,725.00,2307.43,3092.57,1494.48,221.43,110.71
```

Listing 2.5: Contents of `NC.JGR.csv` in the `profiles` output directory.

```
Elevation,Depth,Density,Vp,Vs,Qp,Qs
260.71,0.00,2669.73,1501.00,337.70,33.21,16.60
235.71,25.00,2669.73,1625.01,392.36,42.94,21.47
210.71,50.00,2669.73,1750.00,456.56,54.13,27.07
185.71,75.00,2669.73,1875.00,528.97,66.48,33.24
160.71,100.00,2669.73,2000.00,608.68,79.78,39.89
135.71,125.00,2669.73,2125.00,694.81,93.85,46.93
110.71,150.00,2669.73,2250.00,786.53,108.59,54.29
85.71,175.00,2669.73,2375.00,883.06,123.88,61.94
60.71,200.00,2669.73,2500.00,983.63,139.68,69.84
35.71,225.00,2669.73,2625.00,1087.52,155.96,77.98
10.71,250.00,2669.73,2750.00,1194.06,172.73,86.37
-14.29,275.00,2669.73,2875.00,1302.60,190.02,95.01
-39.29,300.00,2669.73,3000.00,1412.53,207.85,103.93
-64.29,325.00,2669.73,3125.00,1523.29,226.29,113.15
-89.29,350.00,2669.73,3250.00,1634.33,245.39,122.70
-114.29,375.00,2669.73,3375.00,1745.18,265.20,132.60
-139.29,400.00,2669.73,3500.00,1855.36,285.75,142.87
-164.29,425.00,2669.73,3625.00,1964.46,307.08,153.54
-189.29,450.00,2669.73,3750.00,2072.11,329.21,164.61
-214.29,475.00,2669.73,3875.00,2177.95,352.14,176.07
-239.29,500.00,2669.73,3990.52,2273.91,374.02,187.01
-264.29,525.00,2669.73,4032.44,2308.26,382.11,191.06
-289.29,550.00,2669.73,4064.94,2334.75,388.43,194.21
-314.29,575.00,2669.73,4097.44,2360.94,394.83,197.42
-339.29,600.00,2669.73,4129.93,2387.09,401.25,200.62
-364.29,625.00,2669.73,4162.43,2412.93,407.75,203.87
-389.29,650.00,2669.73,4194.93,2438.72,414.26,207.13
-414.29,675.00,2669.73,4227.42,2464.20,420.84,210.42
-439.29,700.00,2669.73,4259.92,2489.61,427.44,213.72
-464.29,725.00,2669.73,4292.42,2514.71,434.11,217.05
-489.29,750.00,2669.73,4324.92,2539.74,440.79,220.40
-514.29,775.00,2669.73,4357.41,2564.45,447.54,223.77
-539.29,800.00,2669.73,4389.91,2589.08,454.30,227.15
-564.29,825.00,2669.73,4422.41,2613.39,461.13,230.56
-589.29,850.00,2669.73,4454.90,2637.62,467.96,233.98
-614.29,875.00,2669.73,4487.40,2661.52,474.85,237.42
-639.29,900.00,2669.73,4519.90,2685.33,481.75,240.88
-664.29,925.00,2669.73,4552.40,2708.81,488.70,244.35
-689.29,950.00,2669.73,4584.89,2732.20,495.66,247.83
-714.29,975.00,2669.73,4617.39,2755.26,502.66,251.33
-739.29,1000.00,2669.73,4649.89,2778.23,509.68,254.84
```

Model creation

Classes

Applications

App

Full name: geomodelgrids.create.apps.create_model.App

Data Members

- **config** *(dict)* Configuration information.
- **model** *(core.model.Model)* Model information.
- **show_progress** *(bool)* If True, print progress to stdout.

Methods

App

Constructor.

main(**kwargs)

Application driver.

Keyword arguments:

- **config[in]** *(str)* Name of configuration file.
- **show_parameters[in]** *(bool)*, If True, print parameters to stdout (default: False)
- **import_domain[in]** *(bool)*, If True, write domain information to model (default: False)
- **import_surfaces[in]** *(bool)* If True, write surfaces information to model (default: False)
- **import_blocks[in]** *(bool)* If True, write block information to model (default: False)
- **all[in]** *(bool)* If True, equivalent to import_domain=True, import_surfaces=True, import_blocks=True (default: False)
- **show_progress[in]** *(bool)* If True, print progress to stdout (default: True)
- **log_filename[in]** *(str)*, Name of log file (default: create_model.log)
- **debug[in]** *(bool)* Print additional debugging information to log file (default: False)

initialize(config_filenames)

Set parameters from config file.

- **config_filenames[in]** *(list of str)* Name of configuration file(s) with parameters.

Core

Model Python Object

Full name: geomodelgrids.create.core.model.Model

Data Members

- **top_surface** *(Surface)* Surface for top of model.
- **topo_bathy** *(Surface)* Surface for top of solid material.
- **blocks** *(list of Block)* Blocks in model.
- **config** *(dict)* Model parameters.
- **storage** *(HDF5)* Interface for storing model.
- **metadata** *(ModelMetadata)* Model metadata.

Methods

- *Model(config)*
- *save_domain()*
- *init_top_surface()*
- *save_top_surface(elevation, batch)*
- *init_topography_bathymetry()*
- *save_topography_bathymetry(elevation, batch)*
- *init_block(block)*
- *save_block(block, values, batch)*
- *update_metadata()*
- *get_attributes()*

Model(config)

Constructor.

- **config[in]** *(dict)* Model parameters.

save_domain()

Write domain information to storage.

init_top_surface()

Create `top_surface` in storage

save_top_surface(elevation, batch=None)

Write `top_surface` information to storage.

- **elevation[in]** *(numpy.array [Nx, Ny])* Elevation of top surface.
- **batch[in]** *(BatchGenerator2D)* Current batch of surface points.

init_topography_bathymetry()

Create topography/bathymetry in storage.

save_topography_bathymetry(elevation, batch=None)

Write `topography_bathymetry` information to storage.

- **elevation[in]** *(numpy.array [Nx, Ny])* Elevation of topography/bathymetry surface.
- **batch[in]** *(BatchGenerator2D)* Current batch of surface points.

init_block(block)

Create block in storage.

- **block[in]** *(Block)* Block information.

save_block(block, values, batch=None)

Write block information to storage.

- **block[in]** *(Block)* Block information.
- **values[in]** *(numpy.array [Nx, Ny, Nz, Nv])* Gridded data associated with block.
- **batch[in]** *(BatchGenerator3D)* Current batch of points in domain.

update_metadata()

Update all metadata for model using current model configuration.

get_attributes()

Get attributes for model.

- **returns** Tuple of model attributes.

Block Python Object

Full name: geomodelgrids.create.core.model.Block

Data Members

- **name** *(str)* Name of block.
- **model_metadata** *(ModelMetadata)* Metadata for model domain containing the block.
- **x_resolution** *(float)* Resolution in x direction if uniform resolution, otherwise `None`.
- **y_resolution** *(float)* Resolution in y direction if uniform resolution, otherwise `None`.
- **z_resolution** *(float)* Resolution in z direction if uniform resolution, otherwise `None`.
- **x_coordinates** *(tuple)* Coordinates in x direction if variable resolution, otherwise `None`.
- **y_coordinates** *(tuple)* Coordinates in y direction if variable resolution, otherwise `None`.
- **z_coordinates** *(tuple)* Coordinates in z direction if variable resolution, otherwise `None`.
- **z_top** *(float)* Elevation of top of block.
- **z_bot** *(float)* Elevation of bottom of block.
- **z_top_offset** *(float)* Vertical offset of top slice of points below top of block.
- **chunk_size** *(tuple)* Dimensions of dataset chunk.

Methods

- *Block(name, model_metadata, config)*
- *get_dims()*
- *get_batches()*
- *generate_points(top_surface, batch)*
- *get_surface(surface, batch)*
- *get_attributes()*

Block(name, model_metadata, config)

- **name[in]** *(str) Name of block.
- **model_metadata[in]** *(ModelMetadata)* Metadata for model domain containing the block.
- **config[in]** *(dict)* Block parameters as a dictionary.
  - `x_resolution` *(float)* Resolution in x direction (if uniform resolution).
  - `y_resolution` *(float)* Resolution in y direction (if uniform resolution).
  - `z_resolution` *(float)* Resolution in z direction (if uniform resolution).
  - `x_coordinates` *(float)* Coordinates in x direction (if variable resolution).
  - `y_coordinates` *(float)* Coordinates in y direction (if variable resolution).
  - `z_coordinates` *(float)* Coordinates in z direction (if variable resolution).
  - `z_top` *(float)* Elevation of top of block if uniform resolution in z direction.
  - `z_bot` *(float)* Elevation of bottom of block if uniform resolution in z direction.
  - `z_top_offset` *(float)* Vertical offset of top slice of points below top of block (used to avoid roundoff errors).
  - `chunk_size` *(tuple)* Dimensions of dataset chunk (should be about 10Kb - 1Mb).

get_dims()

Get number of points in block along each dimension.

- **returns** Tuple of number of points along each dimension (num_x, num_y, num_z).

get_batches(batch_size)

Get batch generator for block.

- **returns** BatchGenerator3D for block.

generate_points(top_surface, batch)

Generate grid of points in block.

- **top_surface[in]** *(Surface)* Elevation of top surface of model domain.
- **batch[in]** *(BatchGenerator3D)* Current batch of points in block.
- **returns** 3D array (Nx*Ny*Nz,3) of point locations in block.

get_surface(surface, batch=None)

Get surface grid for block.

- **surface** *(Surface)* Surface in model domain.
- **batch** *(BatchGenerator3D)* Current batch of points in block.
- **returns** *(numpy array)* [Nx,Ny] with elevation of surface for current batch in block.

get_attributes()

Get attributes associated with block.

- **returns** Array of tuples with attributes for block.

Surface Python Object

Full name: geomodelgrids.create.core.model.Surface

Data Members

- **name** *(str)* Name of surface.
- **model_metadata** *(ModelMetadata)* Metadata for model domain containing the block.
- **x_resolution** *(float)* Resolution in x direction if uniform resolution, otherwise `None`.
- **y_resolution** *(float)* Resolution in y direction if uniform resolution, otherwise `None`.
- **x_coordinates** *(tuple)* Coordinates in x direction if variable resolution, otherwise `None`.
- **y_coordinates** *(tuple)* Coordinates in y direction if variable resolution, otherwise `None`.
- **chunk_size** *(tuple)* Dimensions of dataset chunk.
- **storage** *(HDF5)* Storage for surface.

Methods

- *Surface(name, model_metadata, config, storage)*
- *get_dims()*
- *get_batches(batch_size)*
- *generate_points(batch)*
- *get_attributes()*

Surface(name, model_metadata, config, storage)

Constructor.

- **name[in]** *(str) Name of block.
- **model_metadata[in]** *(ModelMetadata)* Metadata for model domain containing the block.
- **config[in]** *(dict)* Block parameters as a dictionary.
  - `x_resolution` *(float)* Resolution in x direction (if uniform resolution).
  - `y_resolution` *(float)* Resolution in y direction (if uniform resolution).
  - `x_coordinates` *(float)* Coordinates in x direction (if variable resolution).
  - `y_coordinates` *(float)* Coordinates in y direction (if variable resolution).
  - `chunk_size` *(tuple)* Dimensions of dataset chunk (should be about 10Kb - 1Mb).
- **storage[in]** *(HDF5)* Storage for surface.

get_dims()

Get number of points in surface along each dimension.

- **returns** Tuple of number of points along each dimension (num_x, num_y).

get_batches(batch_size)

Get generator for batches of points.

- **returns** Current batch of points.

generate_points(batch=None)

Generate points for surface.

- **batch** *(BatchGenerator2D)* Current batch of points for elevation data.
- **returns** *(numpy.array [Nx*Ny, 2])** 2D array of point locations for ground surface.

get_attributes()

Get attributes for surface.

- **returns** Tuple of surface attributes.

DataSrc Python Object

Full name: geomodelgrids.create.core.datasrc.DataSrc

Abstract base class defining the data source interface.

Methods

- *DataSrc()*
- *initialize()*
- *get_metadata()*
- *get_top_surface(points)*
- *get_topography_bathymetry(points)*
- *get_values(block, top_surface, topo_bathy, batch)*

DataSrc()

Constructor.

initialize()

Initialize data source.

get_metadata()

Get any additional metadata provided by data source.

- **returns** Dict with additional metadata.

get_top_surface(points)

Query model for elevation of top surface at points.

- **points[in]** *(numpy.array [Nx, Ny])* Coordinates of points in model coordinates.
- **returns** Elevation of top surface at points.

get_topography_bathymetry(points)

Query model for elevation of topography/bathymetry at points.

- **points[in]** *(numpy.array [Nx, Ny])* Coordinates of points in model coordinates.
- **returns** Elevation of topography/bathymetry surface at points.

get_values(block, top_surface, topo_bathy, batch=None)

Query model for values at points.

- **block[in]** *(Block)* Block information.
- **top_surface[in]** *(Surface)* Top surface.
- **topo_bathy[in]** *(Surface)* Topography/bathymetry surface to define depth.
- **batch[in]** *(BatchGenerator3D)* Current batch of points in block.
- **returns** Values at points.

Data Sources

CSV Data Source

CSV DataSrc Python Object

Full name: geomodelgrids.create.data_srcs.csv.datasrc.CSVFile

CSV file data source.

Data Members

- **config** *(dict)* Model parameters.

Methods

- *CSVFile(config)*
- *initialize()*
- *get_metadata()*
- *get_top_surface(points)*
- *get_topography_bathymetry(points)*
- *get_values(block, top_surface, topo_bathy, batch)*

CSVFile(config)

Constructor.

- **config** *(dict)* Model parameters.

initialize()

Initialize data source.

get_metadata()

Get any additional metadata provided by data source.

- **returns** Dict with additional metadata.

get_top_surface(points)

Query model for elevation of top surface at points.

Note: We assume a flat top surface at an elevation of 0, so there is not top surface.

- **points[in]** *(numpy.array [Nx, Ny])* Coordinates of points in model coordinates.
- **returns** `None`

get_topography_bathymetry(points)

Query model for elevation of topography/bathymetry at points.

Note: We assume a flat top surface at an elevation of 0, so there is not top surface.

- **points[in]** *(numpy.array [Nx, Ny])* Coordinates of points in model coordinates.
- **returns** `None`

get_values(block, top_surface, topo_bathy, batch=None)

Query model for values at points.

Warning: Batches with CSV files are not implemented.

- **block[in]** *(Block)* Block information.
- **top_surface[in]** *(Surface)* Top surface.
- **topo_bathy[in]** *(Surface)* Topography/bathymetry surface to define depth.
- **batch[in]** *(BatchGenerator3D)* Current batch of points in block.
- **returns** Values at points.

IRIS EMC Data Source

IRIS EMC DataSrc Python Object

Full name: geomodelgrids.create.data_srcs.iris_emc.datasrc.EMCNetCDF

IRIS EMC data source using NetCDF file.

Important: This data source requires the netCDF4 Python module.

Data Members

- **config** *(dict)* Model parameters.

Methods

- *EMCNetCDF(config)*
- *initialize()*
- *get_metadata()*
- *get_top_surface(points)*
- *get_topography_bathymetry(points)*
- *get_values(block, top_surface, topo_bathy, batch)*

EMCNetCDF(config)

Constructor.

- **config** *(dict)* Model parameters.

initialize()

Initialize data source.

get_metadata()

Get any additional metadata provided by data source.

- **returns** Dict with additional metadata.

get_top_surface(points)

Query model for elevation of top surface at points.

Note: We assume a flat top surface at an elevation of 0, so there is not top surface.

- **points[in]** *(numpy.array [Nx, Ny])* Coordinates of points in model coordinates.
- **returns** `None`

get_topography_bathymetry(points)

Query model for elevation of topography/bathymetry at points.

Note: We assume a flat top surface at an elevation of 0, so there is not top surface.

- **points[in]** *(numpy.array [Nx, Ny])* Coordinates of points in model coordinates.
- **returns** `None`

get_values(block, top_surface, topo_bathy, batch=None)

Query model for values at points.

- **block[in]** *(Block)* Block information.
- **top_surface[in]** *(Surface)* Top surface.
- **topo_bathy[in]** *(Surface)* Topography/bathymetry surface to define depth.
- **batch[in]** *(BatchGenerator3D)* Current batch of points in block.
- **returns** Values at points.

EarthVision Data Source

EarthVision DataSrc Python Object

Full name: geomodelgrids.create.data_srcs.earthvision.datasrc.RulesDataSrc

EarthVision model constructed from rules applies to fault blocks and zones.

Important: This data source requires the EarthVision software and a corresponding license.

Data Members

- **config** *(dict)* Model parameters.
- **model_dir** *(str)* Relative or absolute path to directory containing EarthVision model.
- **api** *(api)* EarthVision API.
- **faultblock_ids** *(dict)* Mapping of fault block names to ids.
- **zone_ids** *(dict)* Mapping of zone names to ids.

Methods

- *RulesDataSrc(config)*
- *initialize()*
- *get_metadata()*
- *get_top_surface(points)*
- *get_topography_bathymetry(points)*
- *get_values(block, top_surface, topo_bathy, batch)*

DataSrc(config)

Constructor.

- **config** *(dict)* Model parameters.

initialize()

Initialize data source.

get_metadata()

Get any additional metadata provided by data source.

- **returns** Dict with additional metadata.

get_top_surface(points)

Query model for elevation of top surface at points.

- **points[in]** *(numpy.array [Nx, Ny])* Coordinates of points in model coordinates.
- **returns** `None`

get_topography_bathymetry(points)

Query model for elevation of topography/bathymetry at points.

- **points[in]** *(numpy.array [Nx, Ny])* Coordinates of points in model coordinates.
- **returns** None

get_values(block, top_surface, topo_bathy, batch=None)

Query model for values at points.

- **block[in]** *(Block)* Block information.
- **top_surface[in]** *(Surface)* Top surface.
- **topo_bathy[in]** *(Surface)* Topography/bathymetry surface to define depth.
- **batch[in]** *(BatchGenerator3D)* Current batch of points in block.
- **returns** Values at points.

EarthVision API Python Object

Full name: geomodelgrids.create.data_srcs.earthvision.api.EarthVisionAPI

API for running some specific EarthVision programs.

Data Members

- **model_dir** *(str)* Relative or absolute path of directory containing EarthVision model.
- **env** *(dict) Environment variables for accessing EarthVision executables and libraries.

Methods

- *EarthVisionAPI(model_dir, env)*
- *ev_facedump(filename_faces)*
- *ev_label(filename_values, filename_points, filename_model, dtype, converters)*
- *ev_fp(formula, filename_out)*

EarthVisionAPI(model_dir, env)

Constructor.

- **model_dir[in]** *(str)* Relative or absolute path of directory containing EarthVision model.
- **env[in]** *(dict) Environment variables for accessing EarthVision executables and libraries.

ev_facedump(filename_faces)

Run 'ev_facedump {filename_faces}'.

- **filename_faces[in]** *(str)* Name of faces file.
- **returns** Output of `ev_facedump` as list of lines.

ev_label(filename_values, filename_points, filename_model, dtype, converters)

Run 'ev_label -m FILENAME_MODEL -o FILENAME_VALUES FILE'.

- **filename_values** *(str)* Name of file for output values.
- **filename_points** *(str)* Name of file with input points.
- **filename_model** *(str)* Name of EarthVision model (.seq) file.
- **dtype** *(dict)* Mapping of output columns to numpy type.
- **converters** *(dict)* Mapping of output columns to function to convert to dtype.
- **returns** numpy.array with output.

ev_fp(formula, filename_out)

Run 'ev_fp < {formula}'.

- **formula** *(str)* Formula for EarthVision formula processor.
- **filename_out** *(str)* Name of output file in formula.
- **returns** numpy.array with output.

Input/Output

HDF5Storage Python Object

Full name: geomodelgrids.create.io.hdf5.HDF5Storage

HDF5 file for storing gridded model.

Data Members

- **filename** *(str)* Name of HDF5 file.

Methods

- *HDF5Storage(filename)*
- *save_domain(domain)*
- *create_surface(surface)*
- *save_surface_metadata(surface)*
- *save_surface(surface, elevation, batch)*
- *load_surface(surface, batch)*
- *create_block(block)*
- *save_block_metadata(block)*
- *save_block(block, data, batch)*

HDF5Storage(filename)

Constructor.

- **filename[in]** *(str) Name for HDF5 file.

save_domain(domain)

Write domain attributes to HDF5 file.

- **domain[in]** *(Model)* Model domain.

create_surface(surface)

Create surface in HDF5 file.

- **surface[in]** *(Surface)* Model surface.

save_surface_metadata(surface)

Write surface metadata to HDF5 file.

- **surface[in]** *(Surface)* Model surface

save_surface(surface, elevation, batch=None)

Write surface to HDF5 file.

- **surface** *(Surface)* Model surface.
- **elevation** *(numpy.array)* Elevation of surface.
- **batch** *(BatchGenerator2D)* Current batch of surface points.

load_surface(surface, batch=None)

Load surface from HDF5 file.

- **surface** *(Surface)* Model surface.
- **batch** *(BatchGenerator2D)* Current batch of surface points.
- **returns** numpy.array with elevation of surface.

create_block(block)

Create block in HDF5 file.

- **block** *(Block)* Block in model.

save_block_metadata(block)

Write block metadata to HDF5 file.

- **block[in]** *(Block)* Block associated with gridded data.

save_block(block, data, batch=None)

Write block data to HDF5 file.

- **block** *(Block)* Block in model.
- **data** *(numpy.array) [Nx, Ny, Nz, Nv]* Array of gridded data.
- **batch** *(BatchGenerator3D)* Current batch of block points.

Utilities

Batch Python Objects

BatchGenerator2D

Full name: geomodelgrids.create.utils.batch.BatchGenerator2D

Iterator for batches of points for 2D domains.

```
for batch in BatchGenerator2D(num_x, num_y, max_nvalues):
    # Use batch
```

Data Members

- **num_x** *(int)* Number of points in x direction.
- **num_y** *(int)* Number of points in y direction.
- **bnum_x** *(int)* Number of points in x direction for current batch.
- **bnum_y** *(int)* Number of points in y direction for current batch.
- **nbatch_x** *(int)* Number of batches in x direction.
- **nbatch_y** *(int)* Number of batches in y direction.
- **ix** *(int)* Batch index in x direction.
- **iy** *(int)* Batch index in y direction.
- **x_range** *(tuple)* Starting and ending indices in x direction of current batch.
- **y_range** *(tuple)* Starting and ending indices in y direction of current batch.

Methods

- *BatchGenerator2D(num_x, num_y, max_nvalues)*
- *__str__()*
- *__iter__()*
- *__next__()*

BatchGenerator2D(num_x, num_y, max_nvalues=None)

Constructor.

- **num_x** *(int)* Number of points in x direction.
- **num_y** *(int)* Number of points in y direction.
- **max_nvalues** *(int)* Maximum number of points in a batch.

__str__()

Get string representation.

- **returns** Batch as a string.

__iter__()

Iteration.

- **returns** Batch object.

__next__()

Get next batch.

- **returns** Batch object with next batch.

BatchGenerator3D

Full name: geomodelgrids.create.utils.batch.BatchGenerator3D

Iterator for batches of points for 3D domains.

```
for batch in BatchGenerator3D(num_x, num_y, num_z, max_nvalues):
    # Use batch
```

Data Members

- **num_x** *(int)* Number of points in x direction.
- **num_y** *(int)* Number of points in y direction.
- **num_z** *(int)* Number of points in z direction.
- **bnum_x** *(int)* Number of points in x direction for current batch.
- **bnum_y** *(int)* Number of points in y direction for current batch.
- **bnum_z** *(int)* Number of points in z direction for current batch.
- **nbatch_x** *(int)* Number of batches in x direction.
- **nbatch_y** *(int)* Number of batches in y direction.
- **nbatch_z** *(int)* Number of batches in z direction.
- **ix** *(int)* Batch index in x direction.
- **iy** *(int)* Batch index in y direction.
- **iz** *(int)* Batch index in z direction.
- **x_range** *(tuple)* Starting and ending indices in x direction of current batch.
- **y_range** *(tuple)* Starting and ending indices in y direction of current batch.
- **z_range** *(tuple)* Starting and ending indices in z direction of current batch.

Methods

- *BatchGenerator3D(num_x, num_y, num_z, max_nvalues)*
- *__str__()*
- *__iter__()*
- *__next__()*

BatchGenerator3D(num_x, num_y, num_z, max_nvalues=None)

Constructor.

- **num_x** *(int)* Number of points in x direction.
- **num_y** *(int)* Number of points in y direction.
- **num_z** *(int)* Number of points in z direction.
- **max_nvalues** *(int)* Maximum number of points in a batch.

__str__()

Get string representation.

- **returns** Batch as a string.

__iter__()

Iteration.

- **returns** Batch object.

__next__()

Get next batch.

- **returns** Batch object with next batch.

Configuration Python Functions

Functions

- *string_to_list(list_string, delimiter)*
- *get_config(filenames, keep_case, verbose)*

string_to_list(list_string, delimiter=",")

Convert object as string into a list of objects.

If the object is already a list, return the list.

- **list_string[in]** *(str)* List as string, e.g., "[a, b, c]".
- **delimiter[in]** *(str)* Character(s) separating strings.
- **returns** List of objects, e.g., ["a", "b", "c"]

get_config(filenames, keep_case=True, verbose=False)

Get configuration from `.cfg` files.

- **filenames[in]** *(list)* List of `.cfg` files to read.
- **keep_case[in]** *(bool)* If True, maintain case in section headings, otherwise convert to lowercase.
- **verbose[in]** *(bool)* If True, print out progress.
- **returns** Dictionary with configuration.

Units Python Functions

Functions

- *length_scale(name)*

length_scale(name)

Get length scale associated with units.

- **name** *(str)* Units of length, e.g., "m", "meter", "km", "kilometer", "ft", "feet".
- **returns** Length of unit in meters.

Examples

> **Warning:** We do not yet have examples for how to create GeoModelGrids models.

C++ API

Serial C++ API

Classes

All classes in the serial C++ API are in the `geomodelgrids::serial` namespace.

Query

Full name: geomodelgrids::serial::Query

Enumerated types

SquashingEnum

- **SQUASH_NONE** No squashing.
- **SQUASH_TOP_SURFACE** Squash relative to the top surface of the model.
- **SQUASH_TOPOGRPAHY_BATHYMETRY** Squash relative to the topography/bathymetry surface.

Methods

Query()

Constructor.

getErrorHandler()

Get the error handler.

initialize(const std::vector<std::string>& modelFilenames, const std::vector<std::string>& valueNames, const std::string& inputCRSString)

Setup for querying.

- **modelFilenames**[in] Array of model filenames (in query order).
- **valueNames**[in] Array of names of values to return in query.
- **inputCRSString**[in] Coordinate reference system (CRS) as string (PROJ, EPSG, WKT) for input points.

setSquashMinElev(const double value)

Set minimum elevation (m) above which vertical coordinate is given as -depth.

This option is used to adjust (squash) topography to sea level above `value`. Below `value` the original geometry of the model is maintained. For example, this maintains the original geometry of deeper structure.

- **value**[in] Minimum elevation (m) for squashing topography.

setSquashing(const SquashingEnum value)

Set type of squashing.

- **value**[in] True if squashing is on, false otherwise.

double queryTopElevation(const double x, const double y)

Query model for elevation of the top surface of the model at a point using bilinear interpolation (interpolation along each model axis).

- **x**[in] X coordinate of of point in (in input CRS).
- **y**[in] Y coordinate of of point in (in input CRS).
- **return value** Elevation (meters) of top surface at point.

double queryTopoBathyElevation(const double x, const double y)

Query model for elevation of the topography/bathymetry surface of the model at a point using bilinear interpolation (interpolation along each model axis). If the model does not contain a topography/bathymetry surface, then the top surface of the model is used.

- **x**[in] X coordinate of of point in (in input CRS).
- **y**[in] Y coordinate of of point in (in input CRS).
- **return value** Elevation (meters) of topography/bathymetry surface at point.

query(const double* values, const double x, const double y, const double z)

Query model for values at a point using trilinear interpolation (interpolation along each model axis).

- **values**[out] Array of values (must be preallocated).
- **x**[in] X coordinate of of point in (in input CRS).
- **y**[in] Y coordinate of of point in (in input CRS).
- **z**[in] Z coordinate of of point in (in input CRS).

finalize()

Cleanup after querying.

Model

Full name: geomodelgrids::serial::Model

Enums

ModelMode

- **READ** Read only mode.
- **READ_WRITE** Read/write mode.
- **READ_WRITE_TRUNCATE** Read/write mode, truncate upon opening.

DataLayout

- **VERTEX** Vertex-based data (values are specified at coordinates of vertices).
- **CELL** Cell-based data (values are specified at centers of grid cells).

Methods

Model()

Constructor.

setInputCRS(const std::string& value)

Set the coordinate reference system (CRS) of query input points.

- **value**[in] CRS of input points as string (PROJ, EPSG, WKT).

open(const char* filename, ModelMode mode)

Open the model for querying.

close()

Close the model after querying.

loadMetadata()

Load model metadata.

initialize()

Initialize the model.

const std::vector<std::string>& getValueNames()

Get names of values in the model.

- **returns** Array of names of values in the model.

const std::vector<std::string>& getValueUnits()

Get units of values in the model.

- **returns** Array of units of values in the model.

DataLayout getDataLayout()

Get data layout for the model.

- **returns** Data layout for values.

const double* getDims()

Get model dimension.

- **returns** Model dimensions (m) [x, y, z].

const double* getOrigin()

Get coordinates of model origin in geographic projection.

- **returns** Coordinates of model origin [x, y].

double getYAzimuth()

Get azimuth of y coordinate axis.

- **returns** Azimuth (degrees) of y coordinate axis.

const std::string& getCRSString()

Get coordinate reference system for model.

- **returns** Coordinate reference system for model as Proj, EPSG, or WKT.

const geomodelgrids::serial::ModelInfo& getInfo()

Get model description information.

- **returns** Model description.

const geomodelgrids::serial::Surface& getTopSurface()

Get top surface of model.

- **returns** Model surface.

const geomodelgrids::serial::Surface& getTopoBathy()

Get topography/bathymetry surface of model.

- **returns** Model surface.

const std::vector<geomodelgrids::serial::Block*>& getBlocks()

Get blocks in model.

- **returns** Array of blocks in model.

bool contains(const double x, const double y, const double z)

Does model contain given point?

- **x**[in] X coordinate of point (in input CRS).
- **y**[in] Y coordinate of point (in input CRS).
- **z**[in] Z coordinate of point (in input CRS).
- **returns** True if model contains given point, false otherwise.

double queryTopElevation(const double x, const double y)

Query model for elevation of the top surface at a point using bilinear interpolation.

- **x**[in] X coordinate of point (in input CRS).
- **y**[in] Y coordinate of point (in input CRS).
- **returns** Elevation (meters) of surface at point.

double queryTopoBathyElevation(const double x, const double y)

Query model for elevation of the topography/bathymetry surface at a point using bilinear interpolation.

- **x**[in] X coordinate of point (in input CRS).
- **y**[in] Y coordinate of point (in input CRS).
- **returns** Elevation (meters) of surface at point.

const double* query(const double x, const double y, const double z)

Query model for values at a point using bilinear interpolation

- **x**[in] X coordinate of point (in input CRS).
- **y**[in] Y coordinate of point (in input CRS).
- **z**[in] Z coordinate of point (in input CRS).
- **returns** Array of model values at point.

ModelInfo

Full name: geomodelgrids::serial::ModelInfo

Methods

ModelInfo()

Constructor.

const std::string& getTitle()

Get title.

- **returns** Title of model.

const std::string& getId()

Get identifier.

- **returns** Model identifier.

const std::string& getDescription()

Get description.

- **returns** Model description.

const std::vector<std::string>& getKeywords()

Get keywords describing model.

- **returns** Array of keywords.

const std::string& getHistory()

Get model history.

- **returns** Model history.

const std::string& getComment()

Get comment.

- **returns** Comment about model.

const std::string& getCreatorName()

Get name of creator.

- **returns** Name of creator.

const std::string& getCreatorInstitution()

Get institution of creator.

- **returns** Institution of creator.

const std::string& getCreatorEmail()

Get email of creator.

- **returns** Email of creator.

const std::string& getAcknowledgement()

Get acknowledgements for model.

- **returns** acknowledgements for model.

const std::vector<std::string>& getAuthors()

Get authors of model.

- **returns** Array of author names.

const std::vector<std::string>& getReferences()

Get references associated with model.

- **returns** Array of references.

const std::string& getRepositoryName()

Get name of repository holding model.

- **returns** Name of repository.

const std::string& getRepositoryURL()

Get URL of repository holding model.

- **returns** URL for repository.

const std::string& getRepositoryDOI()

Get DOI for model.

- **returns** Digital Object Identifier.

const std::string& getVersion()

Get model version.

- **returns** Model version.

const std::string& getLicense()

Get model license.

- **returns** Name of license for model.

const std::string& getAuxiliary()

Get auxiliary information.

- **returns** Auxiliary information as string.

load(geomodelgrids::serial::HDF5* const h5)

Load model information.

Surface

Full name: geomodelgrids::serial::Surface

Methods

Surface(const char* const name)

Constructor.

- **name**[in] Name of surface.

loadMetadata(geomodelgrids::serial::HDF5* const h5)

Load metadata from the model file.

double getResolutionX()

Get horizontal resolution along x axis. Only valid (nonzero) for uniform resolution.

- **returns** Resolution along x axis.

double getResolutionY()

Get horizontal resolution along y axis. Only valid (nonzero) for uniform resolution.

- **returns** Resolution along y axis.

double* getCoordinatesX(void)

Get coordinates along x axis.

- **returns** Array of coordinates along x axis. Only valid (non-null) if variable resolution.

double* getCoordinatesY(void)

Get coordinates along y axis.

- **returns** Array of coordinates along y axis. Only valid (non-null) if variable resolution.

const size_t* getDims()

Get number of values along each grid dimension.

- **returns** Array of values along each grid dimension.

setHyperslabDims(const size_t dims[], const size_t ndims)

Set hyperslab size.

- **dims**[in] Dimensions of hyperslab.
- **ndims**[in] Number of dimensions.

openQuery(geomodelgrids::serial::HDF5* const h5)

Prepare for querying.

- **h5**[in] HDF5 object with model.

closeQuery()

Cleanup after querying.

double query(const double x, const double y)

Query for elevation of ground surface at a point using bilinear interpolation.

- **x[in]** X coordinate of point in model coordinate system.
- **y[in]** Y coordinate of point in model coordinate system.
- **returns** Elevation of ground surface at point.

Block

Full name: geomodelgrids::serial::Block

Methods

Block(const char* name)

Constructor with name.

- **name**[in] Name of block.

loadMetadata(geomodelgrids::serial::HDF5* const h5)

Load metadata from the model file.

- **h5**[in] HDF5 object with model.

const std::string& getName()

Get the name of the block.

- **returns** Name of the block

double getResolutionX()

Get resolution along x axis. Only valid (nonzero) for uniform resolution.

- **returns** Resolution along x axis.

double getResolutionY()

Get resolution along y axis. Only valid (nonzero) for uniform resolution.

- **returns** Resolution along y axis.

double getResolutionZ()

Get resolution along z axis. Only valid (nonzero) for uniform resolution.

- **returns** Resolution along z axis.

double getZTop()

Get elevation of top of block in logical space.

- **returns** Elevation of top of block.

double getZBottom()

Get elevation of bottom of block in logical space.

- **returns** Elevation of bottom of block.

double* getCoordinatesX(void)

Get coordinates along x axis.

- **returns** Array of coordinates along x axis. Only valid (non-null) if variable resolution.

double* getCoordinatesY(void)

Get coordinates along y axis.

- **returns** Array of coordinates along y axis. Only valid (non-null) if variable resolution.

double* getCoordinatesZ(void)

Get coordinates along z axis.

- **returns** Array of coordinates along z axis. Only valid (non-null) if variable resolution.

const size_t getDims()

Get numebr of values along each grid dimension.

- **returns** Number of points along grid in each dimension [x, y, z].

size_t getNumValues()

Get number of values stored at each grid point.

- **returns** Number of values stored at each grid point.

setHyperslabDims(const size_t dims[], const size_t ndims)

Set hyperslab size.

- **dims**[in] Dimensions of hyperslab.
- **ndims**[in] Number of dimensions.

openQuery(geomodelgrids::serial::HDF5* const h5)

Prepare for querying.

- **h5** HDF5 object with model.

const double* query(const double x, const double y, const double z)

Query for values at a point using bilinear interpolation.

This low-level function returns all values stored at a point.

- **x[in]** X coordinate of point in model coordinate system.
- **y[in]** Y coordinate of point in model coordinate system.
- **z[in]** Z coordinate of point in model coordinate system.
- **returns** Array of values for model at specified point.

closeQuery()

Cleanup after querying.

bool compare(const Block* a, const Block* b)

Comparison for ordering blocks by vertical location.

- **a[in]** Block to compare.
- **b[in]** Block to compare.
- **returns** True if a.z_top > b.z_top, else false.

Hyperslab

Full name: geomodelgrids::serial::Hyperslab

Methods

Hyperslab(geomodelgrids::serial::HDF* const h5, const char* path, const hsize_t dims[], const size_t ndims)

Constructor.

- **h5**[in] HDF5 object with model.
- **path**[in] Full path to dataset.
- **dims**[in] Array of hyperslab dimensions.
- **ndims**[in] Number of dimensions of hyperslab (should match number of dimensions of dataset).

interpolate(double* const values, const double indexFloat[])

Compute values at point using bilinear interpolation.

- **values**[out] Preallocated array for interpolated values.
- **indexFloat**[in] Index of target point as floating point values.

HDF5

Full name: geomodelgrids::serial::HDF5

Methods

HDF5()

Constructor.

setCache(const size_t cacheSize, const size_t nslots, const double preemption)

Must be called BEFORE open().

The cache should be large enough to fit at least as many chunks as there are in a hyperslab. HDF5 uses a default cache size of 1 MB. We use a default of 16 MB.

The number of slots should be a prime number at least 10 times the number of chunks that can fit into the cache; usually 100 times that number of chunks provides maximum performance. HDF5 uses a default of 521. We use a default of 63997.

Chunk preemption policy for this dataset; value between 0 and 1 (default is 0.75).

See H5Pset_cache for more information.

- **cacheSize**[in] Dataset cache size in bytes.
- **nslots**[in] Number of chunk slots.
- **preemption**[in] Preemption policy value.

open(const char* filename, hid_t mode)

Open HDF5 file.

- **filename**[in] Name of HDF5 file.
- **mode**[in] Mode for HDF5 file.

close()

Close HDF5 file.

bool isOpen()

Check if HDF5 file is open.

bool hasGroup(const char* name)

Check if HDF5 file has group.

- **name**[in] Full path of group.
- **returns** True if group exists, false otherwise.

bool hasDataset(const char* name)

Check if HDF5 file has dataset.

- **name**[in] Full path of dataset.
- **returns** True if dataset exists, false otherwise.

getDatasetDims(hsize_t** dims, int* ndims, const char* path)

Get dimensions of dataset.

- **dims**[out] Array of dimensions.
- **ndims**[out] Number of dimensions.
- **path**[in] Full path of dataset.

getGroupDatasets(std::vector<std::string>* names, const char* parent)

Get names of datasets in group.

- **names**[out] Array of dataset names.
- **group**[in] Full path of group.

readAttribute(const char* path, const char* name, hid_t datatype, void* value)

Read scalar attribute.

- **path**[in] Full path to object with attribute.
- **name**[in] Name of attribute.
- **datastype**[in] Datatype of scalar.
- **value**[out] Attribute value.

readAttribute(const char* path, const char* name, hid_t datatype, void** value, size_t* valuesSize)

Read array attribute.

- **path[in]** Full path to object with attribute.
- **name[in]** Name of attribute.
- **datatype[in]** Datatype of array.
- **values[out]** Attribute value.

std::string readAttribute(const char* path, const char* name)

Read fixed or variable length string attribute.

- **path**[in] Full path to object with attribute.
- **path**[in] name Name of attribute.

readAttribute(const char* path, const char* name, std::vector<std::string>* values)

Read array of fixed or variable length strings attribute.

- **path**[in] Full path to object with attribute.
- **name**[in] Name of attribute.
- **values**[out] Array of strings.

readDatasetHyperslab(void* values, const char* path, const hsize_t* const origin, const hsize_t* const dims, int ndims, hid_t datatype)

Read hyperslab (subset of values) from dataset.

- **values**[out] Values of hyperslab.
- **path**[in] Full path to dataset.
- **orogin**[in] Origin of hyperslab in dataset.
- **dims**[in] Dimensions of hyperslab.
- **ndims**[in] Number of dimensions of hyperslab.
- **datatype**[in] Type of data in dataset.

Examples

Example using C++ API for querying models

See examples/cxx-api/query.c for the complete source code associated with this example.

We first show a complete example in a single code block and then discuss the individual pieces.

Source code

In this example, we hardwire the parameters for convenience. See the C++ query application for a more sophisticated interface.

```
#include "geomodelgrids/serial/Query.hh"
#include "geomodelgrids/utils/ErrorHandler.hh"
#include "geomodelgrids/utils/constants.hh"

int main(int argc, char* argv[]) {
    // Models to query.
    static const std::vector<std::string>& filenames{
        "../../tests/data/one-block-topo.h5",
        "../../tests/data/three-blocks-flat.h5",
    };

    // Values and order to be returned in queries.
    static const std::vector<std::string>& valueNames{ "two", "one" };
    static const size_t numValues = valueNames.size();

    // Coordinate reference system of points passed to queries.
    //
    // The string can be in the form of EPSG:XXXX, WKT, or Proj
    // parameters. In this case, we will specify the coordinates in
    // latitude, longitude, elevation in the WGS84 horizontal
    // datum. The elevation is with respect to the WGS84 ellipsoid.
    static const char* const crs = "EPSG:4326";
    static const size_t spaceDim = 3;

    // Create and initialize serial query object using the parameters
    // stored in local variables.
    geomodelgrids::serial::Query query;
    query.initialize(filenames, valueNames, crs);

    // Log warnings and errors to "error.log".
    geomodelgrids::utils::ErrorHandler& errorHandler = query.getErrorHandler();
    errorHandler.setLogFilename("error.log");

    // Coordinates of points for query (latitude, longitude, elevation).
    static const size_t numPoints = 5;
    static const double points[5*3] = {
        37.455, -121.941, 0.0,
        37.479, -121.734, -5.0e+3,
        37.381, -121.581, -3.0e+3,
        37.283, -121.959, -1.5e+3,
        37.262, -121.684, -4.0e+3,
    };

    // Query for values at points. We must preallocate the array holding the values.
    double values[numValues];
    for (size_t iPt = 0; iPt < numPoints; ++iPt) {
        const double latitude = points[iPt*spaceDim+0];
        const double longitude = points[iPt*spaceDim+1];
```

(continues on next page)

(continued from previous page)

```
        const double elevation = points[iPt*spaceDim+2];
        const int err = query.query(values, latitude, longitude, elevation);

        // Query for elevation of ground surface.
        const double groundElev = query.queryElevation(latitude, longitude);

        // Use the values returned in the query.
    }

    return 0;
}
```

Header files

We include the header files for the query and error handler interfaces as well as defined constants.

```
#include "geomodelgrids/serial/Query.hh"
#include "geomodelgrids/utils/ErrorHandler.hh"
#include "geomodelgrids/utils/constants.hh"
```

Set query parameters

As mentioned earlier, in this example we hardwire all of the query parameters using local variables.

Set the parameters indicating which model to query. Multiple models can be queried and they will be accessed in the order given. Once values are found in one of the models, the rest of the models are skipped.

```
static const std::vector<std::string>& filenames{
   "../../tests/data/one-block-topo.h5",
   "../../tests/data/three-blocks-flat.h5",
};
```

Set the parameters for the values to be returned in queries. The order specified is the order in which they will be returned in queries.

```
static const std::vector<std::string>& valueNames{ "two", "one" };
static const size_t numValues = valueNames.size();
```

Set the coordinate system that is used for the points passed to the query function. The coordinate system is specified using a string corresponding to any coordinate reference system (CRS) recognized by the Proj library. This includes EPGS codes, Proj parameters, and Well-Known Text (WKT). In this case, we specify the coordinates as longitude, latitude, elevation in the WGS horizontal datum, which corresponds to `EPSG:4326`. Elevation is meters above the WGS84 ellipsoid.

```
static const char* const crs = "EPSG:4326";
static const size_t spaceDim = 3;
```

Create and initialize the query

We create the query object and initialize it with the query parameters.

```
geomodelgrids::serial::Query query;
query.initialize(filenames, valueNames, crs);
```

Setup error handler

We get the error handler from the query object and set the name of the log file where warnings and errors will be written.

```
geomodelgrids::utils::ErrorHandler& errorHandler = query.getErrorHandler();
errorHandler.setLogFilename("error.log");
```

Set points for query

In this example, we hardwire the query points using a local variable. The coordinate system is the one specified by the `crs` variable.

```
static const size_t numPoints = 5;
static const double points[5*3] = {
   37.455, -121.941, 0.0,
   37.479, -121.734, -5.0e+3,
   37.381, -121.581, -3.0e+3,
   37.283, -121.959, -1.5e+3,
   37.262, -121.684, -4.0e+3,
};
```

Query model

We query the model looping over the points. We must preallocate the array holding the values returned in the queries. We do not do anything with the values returned as indicated by the comment at the end of the `for` loop.

```
double values[numValues];
for (size_t iPt = 0; iPt < numPoints; ++iPt) {
    const double latitude = points[iPt*spaceDim+0];
    const double longitude = points[iPt*spaceDim+1];
    const double elevation = points[iPt*spaceDim+2];
    const int err = query.query(values, latitude, longitude, elevation);

    // Query for elevation of ground surface.
    const double groundElev = query.queryElevation(latitude, longitude);

    // Use the values returned in the query.
}
```

Parallel C++ API

All classes in the parallel C++ API are in the `geomodelgrids::parallel` namespace.

> **Warning:** This functionality is not yet implemented.

Utilities C++ API

All classes in the utilities C++ API are in the `geomodelgrids::utils` namespace.

Classes

CRSTransformer

Full name: geomodelgrids::utils::CRSTransformer

Methods

- *CRSTransformer()*
- *setSrc(const char* value)*
- *setDest(const char* value)*
- *initialize()*
- *transform(double* destX, double* destY, const double* destZ, const double srcX, const double srcY, const double srcZ)*
- *inverse_transform(double* srcX, double* srcY, const double* srcZ, const double destX, const double destY, const double destZ)*
- *createGeoToXYAxisOrder(const char*)*

CRSTransformer()

Constructor.

setSrc(const char* value)

Set source coordinate system. String can be EPSG:XXXX, WKT, or Proj parameters.

- **value**[in] String specifying source coordinate system.

setDest(const char* value)

Set destination coordinate system. String can be EPSG:XXXX, WKT, or Proj parameters.

- **value**[in] String specifying destination coordinate system.

initialize()

Initialize transformer.

transform(double* destX, double* destY, const double* destZ, const double srcX, const double srcY, const double srcZ)

Transform coordinates from source to destination coordinate system. If `destZ` is `nullptr`, then the z coordinate in the destination coordinate system is not computed.

- **destX**[out] X coordinate in destination coordinate system.
- **destY**[out] Y coordinate in destination coordinate system.
- **destZ**[out] Z coordinate in destination coordinate system.
- **srcX**[in] X coordinate in source coordinate system.
- **srcY**[in] Y coordinate in source coordinate system.
- **srcZ**[in] Z coordinate in source coordinate system.

inverse_transform(ouble* srcX, double* srcY, double* srcZ, const double destX, const double destY, const double destZ)

Transform coordinates from destination to source coordinate system.

- **srcX[out]** X coordinate in source coordinate system.
- **srcY[out]** Y coordinate in source coordinate system.
- **srcZ[out]** Z coordinate in source coordinate system (can be `nullptr`).
- **destX[in]** X coordinate in destination coordinate system.
- **destY[in]** Y coordinate in destination coordinate system.
- **destZ[in]** Z coordinate in destination coordinate system.

CRSTransformer* createGeoToXYAxisOrder(const char* crsString)

Create CRSTransformer that transforms axis order from geo to xy order.

- **crsString[in]** CRS for coordinate system.
- **returns** CRSTransformer.

Indexing

Classes

- *Indexing*
- *IndexingUniform*
- *IndexingVariable*

Indexing

Full name: geomodelgrids::utils::Indexing

Methods

- *Indexing()*
- *getIndex(const double x)*

Indexing()

Constructor.

getIndex(const double x)

Get index.

- **x[in]** Coordinate value.
- **returns** Index for coordinate value.

IndexingUniform

Full name: geomodelgrids::utils::IndexingUniform

Methods

- *IndexingUniform()*
- *getIndex(const double x)*

IndexingUniform()

Constructor.

getIndex(const double x)

Get index.

- **x[in]** Coordinate value.
- **returns** Index for coordinate value.

IndexingVariable

Full name: geomodelgrids::utils::IndexingVariable

Enums

- **ASCENDING** Order coordinates in ascending order.
- **DESCENDING** Order coordinates in descending order.

Methods

- *IndexingVariable(const double*, const size_t, SortOrder)*
- *getIndex(const double x)*

IndexingVariable(const double* x, const size_t numX, SortOrder sortOrder=ASCENDING)

Constructor.

- **x[in]** Array of coordinates along axis.
- **numX[in]** Number of coordinates along axis.
- **sortOrder[in]** Order of coordinate indexing.

getIndex(const double x)

Get index.

- **x[in]** Coordinate value.
- **returns** Index for coordinate value.

ErrorHandler

Full name: geomodelgrids::utils::ErrorHandler

Enums

StatusEnum

- **OK** No errors.
- **WARNING** Non-fatal error.
- **ERROR** Fatal error.

Methods

- *ErrorHandler()*
- *setLogFilename(const char* filename)*
- *getLogFilename()*
- *setLoggingOn(const bool value)*
- *resetStatus()*
- *getStatus()*
- *getMessage()*
- *setError(const char* msg)*
- *setWarning(const char* msg)*
- *logMessage(const char* msg)*

ErrorHandler()

Constructor.

setLogFilename(const char* filename)

Set filename for logging and enable logging.

- **filename**[in] Name of file.

const char* const getLogFilename(void) const

Get filename used in logging.

- **returns** Name of file.

setLoggingOn(const bool value)

Turn logging on/off.

The log filename must have been set for logging to work if it is turned on.

Turning logging on after it has been turned off will cause subsequent messages to be appended to the log file.

- **value**[in] True to turn on logging, false to turn logging off

resetStatus()

Reset error status and clear any error message.

StatusEnum getStatus() const

Get status.

- **returns** Status of errors.

const char* getMessage(void) const

Get warning/error message.

- **returns** Warning/error message.

setError(const char* msg)

Set status to error and store error message.

- **msg**[in] Error message.

setWarning(const char* msg)

Set status to warning and store warning message.

- **msg** Warning message.

logMessage(const char* msg)

Write message to log file.

- **msg**[in] Message to write to log file.

C API

Serial C API

Functions

Serial Query functions

These functions are prefixed by `geomodelgrids_squery`.

Preprocessor macros

- **GEOMODELGRIDS_NODATA_VALUE** Value assigned to points with no data.
- **GEOMODELGRIDS_SQUASH_NONE** Disable squashing.
- **GEOMODELGRIDS_SQUASH_TOP_SURFACE** Squash relative to the top surface of the model.
- **GEOMODELGRIDS_SQUASH_TOPOGRAPHY_BATHYMETRY** Squash relative to the topography/bathymetry surface.

Functions

void* geomodelgrids_squery_create()

Create C++ query object.

- **returns** Pointer to C++ query object (`NULL` on failure).

geomodelgrids_squery_destroy(void** handle)

Destroy C++ query object.

void* geomodelgrids_squery_getErrorHandler()

Get the error handler.

- **handle**[in] Pointer to C++ query object.
- **returns** Pointer to C++ error handler.

int geomodelgrids_squery_initialize(void* handle, const char* const modelFilenames[], const int modelFilenamesSize, const char* const valueNames[], const int valueNamesSize, const char* const inputCRSString)

Setup for querying.

- **handle**[in] Pointer to C++ query object.
- **modelFilenames**[in] Array of model filenames (in query order).
- **modelFilenamesSize**[in] Size of model filenames array.
- **valueNames**[in] Array of names of values to return in query.
- **valueNamesSize**[in] Size of names of values array.
- **inputCRSString**[in] Coordinate reference system (CRS) as string (PROJ, EPSG, WKT) for input points.
- **returns** GeomodelgridsStatusEnum for error status.

int geomodelgrids_squery_setSquashMinElev(const double value)

Set minimum elevation (m) above which vertical coordinate is given as -depth.

This option is used to adjust (squash) topography to sea level above `value`. Below `value` the original geometry of the model is maintained. For example, this maintains the original geometry of deeper structure.

- **handle**[in] Pointer to C++ query object.
- **value**[in] Minimum elevation (m) for squashing topography.
- **returns** GeomodelgridsStatusEnum for error status.

int geomodelgrids_squery_setSquashing(const int value)

Set squashing type.

- **handle**[in] Pointer to C++ query object.
- **value**[in] Valid values are `GEOMODELGRIDS_SQUASH_NONE`, `GEOMODELGRIDS_SQUASH_TOP_SURFACE`, and `GEOMODELGRIDS_SQUASH_TOPOGRAPHY_BATHYMETRY`.
- **returns** GeomodelgridsStatusEnum for error status.

double geomodelgrids_squery_queryTopElevation(const double x, const double y)

Query model for elevation of the top surface of the model at a point.

- **handle**[in] Pointer to C++ query object.
- **x**[in] X coordinate of of point in (in input CRS).
- **y**[in] Y coordinate of of point in (in input CRS).
- **return value** Elevation (meters) of surface at point.

double geomodelgrids_squery_queryTopoBathyElevation(const double x, const double y)

Query model for elevation of the topography/bathymetry surface at a point.

- **handle**[in] Pointer to C++ query object.
- **x**[in] X coordinate of of point in (in input CRS).
- **y**[in] Y coordinate of of point in (in input CRS).
- **return value** Elevation (meters) of surface at point.

int geomodelgrids_squery_query(const double* values, const double x, const double y, const double z)

Query model for values at a point.

- **handle**[in] Pointer to C++ query object.
- **values**[out] Array of values (must be preallocated).
- **x**[in] X coordinate of of point in (in input CRS).
- **y**[in] Y coordinate of of point in (in input CRS).
- **z**[in] Z coordinate of of point in (in input CRS).
- **returns** GeomodelgridsStatusEnum for error status.

geomodelgrids_squery_finalize()

Cleanup after querying.

- **handle**[in] Pointer to C++ query object.

Examples

Example using C API for querying models

See examples/c-api/query.c for the complete source code associated with this example.

We first show a complete example in a single code block and then discuss the individual pieces.

Source code

In this example, we hardwire the parameters for convenience. See the C++ query application for a more sophisticated interface.

```
#include "geomodelgrids/serial/cquery.h"
#include "geomodelgrids/utils/cerrorhandler.h"

int main(int argc, char* argv[]) {
    /* Models to query. */
    static const size_t numModels = 1;
    static const char* const filenames[1] = {
```

(continues on next page)

(continued from previous page)

```
        "../../tests/data/one-block-topo.h5",
    };

    /* Values and order to be returned in queries.
     */
    static const size_t numValues = 2;
    static const char* const valueNames[2] = { "two", "one" };

    /* Coordinate reference system of points passed to queries.
     *
     * The string can be in the form of EPSG:XXXX, WKT, or Proj
     * parameters. In this case, we will specify the coordinates in
     * latitude, longitude, elevation in the WGS84 horizontal
     * datum. The elevation is with respect to the WGS84 ellipsoid.
     */
    static const char* const crs = "EPSG:4326";
    static const size_t spaceDim = 3;

    /* Create and initialize serial query object using the parameters
     * stored in local variables.
     */
    void* query = geomodelgrids_squery_create();
    int err = geomodelgrids_squery_initialize(query, filenames, numModels, valueNames,␣
↪numValues, crs);

    /* Log warnings and errors to "error.log". */
    void* errorHandler = geomodelgrids_squery_getErrorHandler(query);
    geomodelgrids_cerrorhandler_setLogFilename(errorHandler, "error.log");

    /* Coordinates of points for query (latitude, longitude, elevation). */
    static const size_t numPoints = 5;
    static const double points[5*3] = {
        37.455, -121.941, 0.0,
        37.479, -121.734, -5.0e+3,
        37.381, -121.581, -3.0e+3,
        37.283, -121.959, -1.5e+3,
        37.262, -121.684, -4.0e+3,
    };

    /* Query for values at points. We must preallocate the array holding the values. */
    double values[numValues];
    for (size_t iPt = 0; iPt < numPoints; ++iPt) {
        const double latitude = points[iPt*spaceDim+0];
        const double longitude = points[iPt*spaceDim+1];
        const double elevation = points[iPt*spaceDim+2];
        const int err = geomodelgrids_squery_query(query, values, latitude, longitude,␣
↪elevation);

        /* Query for elevation of ground surface. */
        const double groundElev = geomodelgrids_squery_queryElevation(query, latitude,␣
↪longitude);
```

(continues on next page)

(continued from previous page)

```
        /* Use the values returned in the query. */
    }

    /* Destroy query object. */
    geomodelgrids_squery_destroy(&query);

    return 0;
}
```

Header files

We include the header files for the query and error handler interfaces.

```
#include "geomodelgrids/serial/cquery.h"
#include "geomodelgrids/utils/cerrorhandler.h"
```

Set query parameters

As mentioned earlier, in this example we hardwire all of the query parameters using local variables.

Set the parameters indicating which model to query. Multiple models can be queries and they will be accessed in the order given.

```
static const size_t numModels = 1;
static const char* const filenames[1] = {
   "../../tests/data/one-block-topo.h5",
};
```

Set the parameters for the values to be returned in queries. The order specified is the order in which they will be returned in queries.

```
static const size_t numValues = 2;
static const char* const valueNames[2] = { "two", "one" };
```

Set the coordinate system that is used for the points passed to the query function. The coordinate system is specified using a string corresponding to any coordinate reference system (CRS) recognized by the Proj library. This includes EPGS codes, Proj parameters, and Well-Known Text (WKT). In this case, we specify the coordinates as longitude, latitude, elevation in the WGS horizontal datum, which corresponds to `EPSG:4326`. Elevation is meters above the WGS84 ellipsoid.

```
static const char* const crs = "EPSG:4326";
static const size_t spaceDim = 3;
```

Create and initialize the query

We create the query object and initialize it with the query parameters.

```
void* query = geomodelgrids_squery_create();
int err = geomodelgrids_squery_initialize(query, filenames, numModels, valueNames,␣
↪numValues, crs);
```

Setup error handler

We get the error handler from the query object and set the name of the log file where warnings and errors will be written.

```
void* errorHandler = geomodelgrids_squery_getErrorHandler(query);
geomodelgrids_cerrorhandler_setLogFilename(errorHandler, "error.log");
```

Set points for query

In this example, we hardwire the query points using a local variable. The coordinate system is the one specified by the `crs` variable.

```
static const size_t numPoints = 5;
static const double points[5*3] = {
   37.455, -121.941, 0.0,
   37.479, -121.734, -5.0e+3,
   37.381, -121.581, -3.0e+3,
   37.283, -121.959, -1.5e+3,
   37.262, -121.684, -4.0e+3,
};
```

Query model

We query the model looping over the points. We must preallocate the array holding the values returned in the queries. We do not do anything with the values returned as indicated by the comment at the end of the `for` loop.

```
double values[numValues];
for (size_t iPt = 0; iPt < numPoints; ++iPt) {
    const double latitude = points[iPt*spaceDim+0];
    const double longitude = points[iPt*spaceDim+1];
    const double elevation = points[iPt*spaceDim+2];
    const int err = geomodelgrids_squery_query(query, values, latitude, longitude,␣
↪elevation);

    /* Query for elevation of ground surface. */
    const double groundElev = geomodelgrids_squery_queryElevation(query, latitude,␣
↪longitude);

    /* Use the values returned in the query. */
}
```

Cleanup after query

After using the query object, we must destroy it.

```
geomodelgrids_squery_destroy(&query);
```

Parallel C API

> **Warning:** This functionality is not yet implemented.

Utilities C API

Functions

Error Handler

All of these functions are prefixed by `geomodelgrids_cerrorhandler`.

Enums

GeomodelgridsStatusEnum

- **GEOMODELGRIDS_OK** No errors.
- **GEOMODELGRIDS_WARNING** Non-fatal error.
- **GEOMODELGRIDS_ERROR** Fatal error.

Functions

geomodelgrids_cerrorhandler_setLogFilename(void* handle, const char* filename)

Set filename for logging and enable logging.

- **handle**[in] Pointer to C++ error handler object.
- **filename**[in] Name of file.

const char* const geomodelgrids_cerrorhandler_getLogFilename(void* handle) const

Get filename used in logging.

- **handle**[in] Pointer to C++ error handler object.
- **returns** Name of file.

geomodelgrids_cerrorhandler_setLoggingOn(void* handle, const int value)

Turn logging on/off.

The log filename must have been set for logging to work if it is turned on.

Turning logging on after it has been turned off will cause subsequent messages to be appended to the log file.

- **handle**[in] Pointer to C++ error handler object.
- **value**[in] 1 to turn on logging, 0 to turn logging off

geomodelgrids_cerrorhandler_resetStatus(void* handle)

Reset error status and clear any error message.

- **handle**[in] Pointer to C++ error handler object.

GeomodelgridsStatusEnum geomodelgrids_cerrorhandler_getStatus(void* handle) const

Get status.

- **handle**[in] Pointer to C++ error handler object.
- **returns** Status of errors.

const char* geomodelgrids_cerrorhandler_getMessage(void* handle) const

Get warning/error message.

- **handle**[in] Pointer to C++ error handler object.
- **returns** Warning/error message.

2.2.4 Developer Guide

Code layout

The root directory contains basic information files, such as `README` and `LICENSE.md`.

- **bin**: Command line programs.
- **ci-config**: Continuous integration configuration files.
- **developer**: Configuration files for developer related tools.
- **docker**: Docker image configuration files.
- **docs**: Files used to generate the documentation.
- **examples**: Source code illustrating use of the C and C++ APIs.
- **geomodelgrids**: Python API
- **libsrc**: C and C++ API
- **models**: Files used to generate some specific models using geomodelgrids.
- **tests**: Code for continuous integration testing.

Command line programs

The `geomodelgrids_create_model` file is a Python script that uses the Python API. All of the other files are C++ files that use the C++ API.

```
bin
├── Makefile.am
├── borehole.cc
├── geomodelgrids_create_model
├── info.cc
├── isosurface.cc
├── query.cc
└── queryelev.cc
```

C/C++ API

The `apps` directory contains the code for the command line programs. The `serial` directory contains the code for the C/C++ serial API. The `utils` directory contains general C/C++ API utilities.

```
libsrc/
├── Makefile.am
└── geomodelgrids
    ├── Makefile.am
    ├── apps
    │   ├── Borehole.cc
    │   ├── Borehole.hh
    │   ├── Info.cc
    │   ├── Info.hh
    │   ├── Isosurface.cc
    │   ├── Isosurface.hh
    │   ├── Makefile.am
    │   ├── Query.cc
    │   ├── Query.hh
    │   ├── QueryElev.cc
    │   ├── QueryElev.hh
    │   └── appsfwd.hh
    ├── geomodelgrids_serial.hh
    ├── serial
    │   ├── Block.cc
    │   ├── Block.hh
    │   ├── HDF5.cc
    │   ├── HDF5.hh
    │   ├── Hyperslab.cc
    │   ├── Hyperslab.hh
    │   ├── Makefile.am
    │   ├── Model.cc
    │   ├── Model.hh
    │   ├── ModelInfo.cc
    │   ├── ModelInfo.hh
    │   ├── Query.cc
    │   ├── Query.hh
    │   ├── Surface.cc
```

(continues on next page)

(continued from previous page)

```
    │   ├── Surface.hh
    │   ├── cquery.cc
    │   ├── cquery.h
    │   └── serialfwd.hh
    └── utils
        ├── CRSTransformer.cc
        ├── CRSTransformer.hh
        ├── ErrorHandler.cc
        ├── ErrorHandler.hh
        ├── GeoTiff.cc
        ├── GeoTiff.hh
        ├── Indexing.cc
        ├── Indexing.hh
        ├── Makefile.am
        ├── TestDriver.cc
        ├── TestDriver.hh
        ├── cerrorhandler.cc
        ├── cerrorhandler.h
        ├── constants.hh
        └── utilsfwd.hh
```

Python interface

```
geomodelgrids
├── Makefile.am
├── __init__.py
└── create
    ├── __init__.py
    ├── apps
    │   ├── __init__.py
    │   └── create_model.py
    ├── core
    │   ├── __init__.py
    │   ├── datasrc.py
    │   └── model.py
    ├── data_srcs
    │   ├── __init__.py
    │   ├── csv
    │   │   ├── __init__.py
    │   │   └── datasrc.py
    │   ├── earthvision
    │   │   ├── __init__.py
    │   │   ├── api.py
    │   │   └── datasrc.py
    │   └── iris_emc
    │       ├── __init__.py
    │       └── datasrc.py
    ├── io
    │   ├── __init__.py
    │   ├── hdf5.py
```

(continues on next page)

(continued from previous page)

```
    │   └── scec_cca.py
    ├── testing
    │   ├── __init__.py
    │   └── datasrc.py
    └── utils
        ├── __init__.py
        ├── batch.py
        ├── config.py
        └── units.py
```

GeoModelGrids Docker Development Environment

The `geomodelgrids-devenv` Docker image https://registry.gitlab.com/baagaard-usgs/geomodelgrids/geomodelgrids-devenv provides all of the dependencies and defines the environment for GeoModelGrids development. It is built using the Ubuntu 20.04 Linux distribution. It is intended to be read only with a separate Docker volume for persistent storage and the GeoModelGrids development workspace. We separate the development "environment" from the "workspace" so that we can update the development environment without affecting the workspace and easily maintain a persistent workspace while starting and stopping the Docker container that holds the development environment.

Prerequisites

1. You need to have Docker installed and running on your computer.
2. You need to have a GitHub account.

Setup

You only need to run these setup steps once.

Fork repositories on GitHub

This creates a copy of the repository in your GitHub account.

1. Log in to your GitHub account, creating an account if you do not already have one.
2. Fork the GeoModelGrids repository: https://github.com/baagaard-usgs/geomodelgrids.

Create Docker volume for persistent storage

On your local machine, create a Docker volume for persistent storage.

```
docker volume create geomodelgrids-dev
```

Start GeoModelGrids development Docker container

Running the command below will:

1. Start (run) the Docker container using the `geomodelgrids-devenv` Docker image and assign it the name `geomodelgrids-dev-workspace`.
2. Mount the docker volume with persistent storage at `/opt/geomodelgrids`.
3. The `geomodelgrids-devenv` Docker image will be downloaded from the GitLab registry <registry.gitlab.com/baagaard-usgs/geomodelgrids>.

```
# Without EarthVision
docker run --name geomodelgrids-dev-workspace --rm -it \
    -v geomodelgrids-dev:/opt/geomodelgrids \
    registry.gitlab.com/baagaard-usgs/geomodelgrids/geomodelgrids-devenv
#
# With EarthVision in $TOOLS_DIR/earthvision-11
docker run --name geomodelgrids-dev-workspace --rm -it \
    -v geomodelgrids-dev:/opt/geomodelgrids \
    -v $TOOLS_DIR/earthvision-11:/opt/earthvision \
    -v $HOME/data/sfbay-geology:/data/sfbay-geology \
    -v $HOME/data/geomodelgrids/:/data/geomodelgrids \
    registry.gitlab.com/baagaard-usgs/geomodelgrids/geomodelgrids-devenv
```

Setup directory structure

We will use the directory following directory structure for the persistent storage.

```
/opt/geomodelgrids
    ├── src
    ├── build
    │   ├── debug
    │   └── opt
    └── dest
        ├── debug
        │   ├── bin
        │   ├── include
        │   ├── lib
        │   └── share
        └── opt
            ├── bin
            ├── include
            ├── lib
            └── share
```

This directory structure is setup for both a debugging version for development (debug directory) and an optimized version for performance testing (opt directory). For now, we will only setup the debugging version.

```
cd /opt/geomodelgrids
mkdir -p ${TOP_BUILDDIR}
mkdir -p ${INSTALL_DIR}
```

Clone repositories

This creates a local copy of the repository in the persistent storage volume of the GeoModelGrids development container.

```
cd /opt/geomodelgrids
git clone --recursive https://github.com/GITHUB_USERNAME/geomodelgrids.git src
```

Configure and build for development

```
cd ${TOP_BUILDDIR}
pushd ${TOP_SRCDIR} && autoreconf -if && popd
${TOP_SRCDIR}/configure \
  --prefix=${INSTALL_DIR} \
  --enable-python \
  --enable-gdal \
  --enable-testing \
  CPPFLAGS="-I${HDF5_INCDIR}" \
  LDFLAGS="-L${HDF5_LIBDIR}" \
  CC=gcc CXX=g++ CFLAGS="-g -Wall" CXXFLAGS="-g -Wall"
make install -j$(nproc)
make check -j$(nproc)
```

Install Visual Studio Code

Install VS Code for your computer.

Install the following extensions:

- Remote - Containers
- C/C++
- Docker
- Live Share
- Python
- Uncrustify

Additionally, we recommend also installing the following extensions:

- GitHub Pull Requests and Issues
- GitLens – Git supercharged
- Material Icon Theme

Running

Start the GeoModelGrids development Docker container

Whenever you need to restart the `geomodelgrids-dev-workspace` Docker container, simply run

```
# Without EarthVision
docker run --name geomodelgrids-dev-workspace --rm -it \
    -v geomodelgrids-dev:/opt/geomodelgrids \
    registry.gitlab.com/baagaard-usgs/geomodelgrids/geomodelgrids-devenv
```

Tip: Make sure Docker is running before you start the container.

Attach VS Code to the Docker container

1. Start VS Code.
2. Click on the Docker extension in the Activity Bar on the far left hand side.
3. Find the `geomodelgrids-dev-workspace` container. Verify that it is running.
4. Right-click on the container and select `Attach Visual Studio Code`. This will open a new window. You should see `Container registry.gitlab.com/baagaard-usgs...` at the left side of the status bar at the bottom of the window.