MDAL

Peter Petrik, Martin Dobias and others

Oct 18, 2021

CONTENTS

CHAPTER

ONE

WHAT IS MDAL?

Mesh Data Abstraction Library (MDAL) is a C++ library for handling unstructured mesh data released with MIT license. It provides a single data model for multiple supported data formats. MDAL is used by QGIS for data access for mesh layers.

MDAL is OSGeo Community Project

See *Software using MDAL*

CHAPTER

TWO

DOWNLOAD

2.1 Current Releases

https://github.com/lutraconsulting/MDAL/releases

2.2 Past Releases

https://github.com/lutraconsulting/MDAL/releases

2.3 Conda Installation

MDAL can be installed as a stand-alone package (i.e. outside of QGIS) using conda.

The package can installed by running :

```
conda install -c conda-forge mdal
```

This package provides the MDAL ABI through the mdal shared object(i.e. mdal.dll, libmdal.dylib or libmdal.so) and the mdalinfo CLI.

Note: A friendly note about versions. The conda package is usually targetted at the latest version of GDAL on conda-forge. This is usually a later version than used by QGIS. Therefore, there may be some subtle differences in behaviour when loading e.g. GRIB files.

2.3.1 UPM Installation

There is also a community supported Unity Package Manager (UPM) package for MDAL to allow MDAL to be used in Unity based projects. This builds on top of the Conda package.

https://openupm.com/packages/com.virgis.mdal/

2.4 Development Source

The main repository for MDAL is located on github at https://github.com/lutraconsulting/MDAL.

You can obtain a copy of the active source code by issuing the following command

```
git clone https://github.com/lutraconsulting/MDAL.git
```

CHAPTER

THREE

PROGRAMS

3.1 MDAL programs

3.1.1 mdalinfo

3.1.1.1 Synopsis

```
mdalinfo [--help]
```

3.1.1.2 Description

mdalinfo program lists various information about a MDAL supported files

The following command line parameters can appear in any order

-todo
todo

3.1.1.3 Example

```
./mdalinfo ./2dm/quad_and_triangle.2dm ./ascii_dat/quad_and_triangle_vertex_vector.dat
mdalinfo 0.5.90
Mesh File: ./2dm/quad_and_triangle.2dm
Mesh loaded: OK
  Driver: 2DM
  Vertex count: 5
  Edge count: 0
  Face count: 2
  Edge count: 0
  Projection: undefined
Dataset File: ./ascii_dat/quad_and_triangle_vertex_vector.dat
Datasets loaded: OK
  Groups count: 2
  Bed Elevation
  VertexVectorDataset ( Vector )
```

- *mdalinfo*: Lists information about a MDAL dataset.

CHAPTER

FOUR

MDAL DRIVERS

Short name	Long name	Creation	Geo-referencing	Build requirements
2dm	SMS mesh format	No	No	Built-in by default
3Di	NetCDF with Climate and Forecast (CF) metadata	No	No	Built-in by default
ADCIRC	NetCDF outputs from ADCIRC	No	No	Built-in by default
DAT	ASCII or Binary Dataset Files	No	No	Built-in by default
dfs2	DHI Grid Series File	No	No	???
dfsu	DHI flexible file format	No	No	???
Esri TIN	Elevation model as TIN	No	No	Built-in by default
FLO-2D	FLO-2D outputs format	No	No	Built-in by default
GRIB	WMO General Regularly-distributed Information in Binary form	No	No	Built-in by default
HEC-RAS	HEC-RAS outputs format	No	No	Built-in by default
NetCDF	Network Common Data Form	No	No	Built-in by default
PLY	ASCII Stanford Polygon Format	No	No	Built-in by default
SAGA Flow	Raster flow directions from SAGA GIS	No	No	Built-in by default
TELEMAC	TELEMAC outputs format	No	No	Built-in by default
SWW	NetCDF format for AnuGA	No	No	Built-in by default
UGRID	NetCDF with Climate and Forecast (CF) metadata	No	No	Built-in by default
XDMF	eXtensible Data Model and Format	No	No	Built-in by default
XMDF	eXtensible Model Data Format	No	No	Built-in by default
XMS TIN	ASCII TIN	No	No	Built-in by default

4.1 GRIB – WMO General Regularly-distributed Information in Binary form

Driver short name

GRIB

Driver built-in by default

This driver is built-in by default

MDAL supports GRIB1 (reading) and GRIB2 (reading and writing) format raster data, with support for common coordinate system, georeferencing and other metadata. GRIB format is commonly used for distribution of Meteorological information, and is propagated by the World Meteorological Organization.

4.2 NetCDF – Network Common Data Form

Driver short name

NetCDF

Driver built-in by default

This driver is built-in by default

MDAL supports NetCDF (Network Common Data Form) (reading) format through GDAL driver. NetCDF format is commonly used for distribution of scientific data.

4.3 2dm – SMS mesh format

Driver short name

2dm

Driver built-in by default

This driver is built-in by default

MDAL supports reading and writing to 2dm format. 2dm is a generic file format to save mesh finite element mesh information defined by SMS.

4.4 XDMF – eXtensible Data Model and Format

Driver short name

XDMF

Driver built-in by default

This driver is built-in by default

MDAL supports reading of the XDMF format generated by BASEMENT (version 3 or later)

4.5 XMDF – eXtensible Model Data Format

Driver short name

XMDF

Driver built-in by default

This driver is built-in by default

MDAL supports reading of the XMDF format generated by TUFLOW, HYDRO_AS-2D and other hydraulic modelling software applications.

4.6 DAT – ASCII or Binary Dataset Files

Driver short name

DAT

Driver built-in by default

This driver is built-in by default

MDAL supports reading of and writing to the DAT format. This format is generated by TUFLOW , BASEMENT and Hydrotec hydraulic modelling software applications.

4.7 SWW – NetCDF format for AnuGA

Driver short name

SWW

Driver built-in by default

This driver is built-in by default

MDAL supports reading the SWW format. This format is based on NetCDF and generated by AnuGA.

4.8 HDF – HEC-RAS outputs format

Driver short name

HEC-RAS

Driver built-in by default

This driver is built-in by default

MDAL supports reading of the HDF files generated by HEC-RAS.

4.9 3Di – NetCDF with Climate and Forecast (CF) metadata

Driver short name

3Di

Driver built-in by default

This driver is built-in by default

MDAL supports reading of the 3Di format. This format follows CF conventions.

4.10 PLY – ASCII Stanford Polygon Format

Driver short name

PLY

Driver built-in by default

This driver is built-in by default

MDAL supports reading and writing of PLY format in ASCII and BINARY formats. This format is used to store triangulated irregular networks (TINs) in ASCII format with arbitrary scalar data on the vertices, faces and/or edges.

Note that:

- This driver can read and write Meshes containing any (consistent) combination of vertices, faces and edges and can access Scalar and Vector datasets on vertices, faces & edges and Scalar (only) datasets on volumes.
- All MDAL Scalar datasets are stored in PLY as doubles,
- Vector Datasets are stored as PLY double lists. When Reading a PLY file, a list data type is converted into an MDAL Vector dataset with only the first two values of the list used,
- Datasets on Volumes are stored as TWO equal sized lists on the face elements (one containing the values using the name of the MDAL DatasetGroup and one containing the extrusion values which has "__vol" appended to the name). This is a format that is totally consistent with the PLY format but is unique to this driver.
- This driver implements an additional feature without breaking the standard. If it finds a line in the header starting "comment crs " it will use the rest of the line as the string to set the mesh projection.

The PLY format allows data to be attached both to edges and to faces in the same data set and this driver will successfully load that data. However, most host applications (like QGIS) will expect the dataset to be either a 1d mesh - with edges - or a 2D mwesh - with faces. There is no guarantee how the host application will process a dataset with both.

4.11 UGRID – NetCDF with Climate and Forecast (CF) metadata

Driver short name

UGRID

Driver built-in by default

This driver is built-in by default

MDAL supports reading of unstructured grid (mesh) data from NetCDF files based on UGRID format. The UGRID conventions are an extension to CF conventions to describe hybrid 1D-2D-3D meshes and unstructured mesh topology and geometry. UGRID is used for example by the Delft3D Flexible Mesh Suite, FVCom/VisIt, ADCIRC and BAW's unstructured grid tools. Additionally, MDAL supports detailed 1D network topologies and geometries.

4.12 Esri TIN – Elevation model as TIN

Driver short name

Esri TIN

Driver built-in by default

This driver is built-in by default

MDAL supports reading of Esri TIN format. This format is used to store elevation in triangulated irregular network (TIN).

4.13 HDF and ASCII – FLO-2D outputs format

Driver short name

FLO-2D

Driver built-in by default

This driver is built-in by default

MDAL supports reading of the HDF and ASCII files generated by FLO-2D.

This driver can be used to read FLO-2D mesh (1D, 2D) The required files for this format are :

- CADPTS.DAT file that contains the cells centre position
- for 2D mesh, FPLAIN.DAT that includes the 2D mesh topology
- for 1D mesh, CHAN.DAT that includes the 1D mesh topology and CHANBANK.DAT that includes the left/right bank relation

The 2D mesh is composed of cells which centres are contained in CADPTS.DAT files. FPLAIN.DAT defines the neighboring of the cells. 2D mesh datasets file are not required but can be :

- TIMDEP.hdf5, hdf5 format files
- TIMDEP.OUT
- DEPTH.OUT
- VELFP.OUT

hdf5 format extra datasets can also be read, and this driver supports persisting dataset with this format.

For 1D mesh, the cell centres are also defined in CADPTS.DAT file. Each vertex is linked to a cell from CADPTS.DAT that represents the position of the left bank, and each cell can be linked only with one left bank. So the topology of the 1D mesh is defined by the id of the left bank of each vertex. Vertices can also have (but not required) a right bank represented by another cell. For the right bank, one cell can be linked to several right banks of vertex, so to several vertices. When a vertex hasn't a right bank, its position is defined by the cell link to the left bank. When vertex has a right bank, the vertex position is the middle of the segment between the right and the left bank.

1D mesh datasets file is not required but is HYCHAN.OUT

4.14 Selafin/Serafin – TELEMAC outputs format

Driver short name

TELEMAC

Driver built-in by default

This driver is built-in by default

MDAL supports reading of the _Selafin/Serafin format generated by TELEMAC.

4.15 SAGA Flow – Raster flow directions from SAGA GIS

Driver short name

SAGA Flow

Driver built-in by default

This driver is built-in by default

MDAL supports reading of the SAGA Flow files. The file needs to be further processed by the Crayfish plugin.

4.16 ADCIRC – NetCDF outputs from ADCIRC

Driver short name

ADCIRC

Driver built-in by default

This driver is built-in by default

MDAL supports reading of the ADCIRC NetCDF format. The file needs to be further processed to comply with UGRID conventions.

4.17 XMS TIN – ASCII TIN

Driver short name

XMS TIN

Driver built-in by default

This driver is built-in by default

MDAL supports reading of XMS TIN format. This format is used to store triangulated irregular network (TIN) in ASCII format.

4.18 DFSU – DHI flexible file format

Driver short name

dfsu

MDAL supports reading of DFSU format. DFSU is DHI Flexible File Format defined by DHI. This is external driver that is not shipped directly with MDAL installations. To build the driver, follow instruction in README DFSU_README

Driver is available only on Windows.

4.19 DFS2 – DHI Grid Series File

Driver short name

dfs2

MDAL supports reading of DFS2 format. DFS2 is DHI Grid Series File defined by DHI. This is external driver that is not shipped directly with MDAL installations. To build the driver, follow instruction in README DFS2_README

Driver is available only on Windows.

CHAPTER

FIVE

USER ORIENTED DOCUMENTATION

5.1 MDAL Data Model

5.1.1 Data Model

In our context, a mesh is a collection of vertices, edges and faces in 2D or 3D space:

1. Vertices - XY(Z) points (in the layer's coordinate reference system).
2. Edges - connect pairs of vertices.
3. Faces - sets of edges forming a closed shape - typically triangles or quadrilaterals (quads), rarely polygons with higher number of vertices.

5.1.2 Mesh-examples

Mesh gives us information about the spatial structure. In addition to the mesh we have datasets that assign a value to every vertex. For example, ice cap thickness at particular moment of time. A single file may contain multiple datasets - typically multiple quantities (e.g. water depth, water flow) that may be varying in time (time being represented in discrete timesteps, so each quantity may have N arrays, one for each timestep). Datasets do not have to vary in time (e.g. maximum water depth over the whole simulation).

Here is an example of a triangular mesh with numbered vertices:

We can visualize the data by assigning colors to values (similarly to how it is done with "Singleband pseudocolor" raster rendering) and interpolating data between vertices according to the mesh topology. It is common that some quantities are 2D vectors rather than being simple scalar values (e.g. wind direction). For such quantities it is very desired to display arrows indicating vector direction.

5.2 Configuration options

This page discussed runtime configuration options for MDAL, and is distinct from options to the build-time configure script. Runtime configuration options apply on all platforms, and are evaluated at runtime. They can be set programmatically, by commandline switches or in the environment by the user.

Configuration options are normally used to alter the default behavior of MDAL drivers and in some cases the MDAL core. They are essentially global variables the user can set.

5.2.1 How to set configuration options ?

TODO

5.2.2 List of configuration options and where they apply

Note: This list is known to be incomplete. It depends on proper annotation of configuration options where they are mentionned elsewhere in the documentation. If you want to help to extend it, use the `:decl_configoption:`NAME`` syntax in places where a configuration option is mentionned.

CHAPTER

SIX

API

API is omitted in this PDF document. You can consult it on https://mdal.xyz/api/index.html

CHAPTER

SEVEN

TUTORIALS

7.1 Drivers

7.1.1 MDAL driver implementation tutorial

7.1.1.1 Overall Approach

In general new formats are added to MDAL by implementing format specific drivers. Drivers need to be derived class of *MDAL::Driver()* and be registered in *MDAL::DriverManager::DriverManager()*. Drivers can have different types, they can read (or write) the mesh frame or mesh datasets (or both). Some formats have separate file for mesh (frame) definition and data definition, some formats have multiple files per dataset, some formats use single for to store all the data in single file. All possibilities are supported.

7.1.1.2 Driver structure

There are two types of drivers in MDAL

A) Regular drivers: they are compiled into the MDAL library at compile time

B) Dynamic drivers: they are compiled separately, without linkage to MDAL library. MDAL loads the on runtime from folder defined by environment variable *MDAL_DRIVER_PATH*

7.1.1.3 Regular drivers

Each regular MDAL driver consists of files in

1. MDAL/docs/source/drivers/<driver_name>.rst where the driver documentation is preserved
2. MDAL/mdal/frmts/mdal_<driver_name>.cpp where the driver implementation is coded
3. MDAL/tests/test_<driver_name>.cpp where the test is implemented. Test coverage is aimed for more than 90% of the code base. Each driver should come with the driver test with size < 3MB that is stored under the same folder.

The new driver needs to also be registered in *mdal_driver_manager.cpp* file.

The driver can be implemented with lazy-loaded fashion, so the data is gathered only when requested (preferred) or in brute-force way, where everything is loaded in the memory. The latter options is easier to implement as is a usual practice to implement the all-in-memory way by usage of classes *MDAL::MemoryMesh* and *MDAL::MemoryDataset* first.

7.1.1.4 Dynamic drivers

Implementation of the dynamic drivers is a lot more challenging than the regular drivers. To be able to guarantee the binary stability, dynamic drivers needs to expose plain C API with set of exported functions.

The example implementation of the dynamic driver is in *MDAL/tools/externaldriver/*. Basically you need to implement and export all functions defined in *MDAL/tools/externaldriver/mdal_external_driver.h*. Please consult the functions description.

The dynamic driver should not link MDAL library function, so it is not dependent on the version of the library. Also, with the new release of MDAL you do not need to recompile your drive to be able to use it.

On runtime, MDAL scans the folder defined by environment variable *MDAL_DRIVER_PATH* and scans all libraries found in the folder. If any library contains the required *MDAL_DRIVER_** symbols, it is registered in runtime to mdal driver registry.

When the mesh is opened in MDAL, it tries *MDAL_DRIVER_canReadMesh()* function of the driver to find out if the driver shall be used to open the file.

The reading follows by calling *MDAL_DRIVER_M_** functions to get the information and structure of the mesh frame.

Afterwards, the dataset groups are read by *MDAL_DRIVER_G_** functions.

When client requests particular data, they are served through *MDAL_DRIVER_D_data*. Data could be lazy-loaded.

CHAPTER

EIGHT

COMMUNITY

MDAL's community interacts through *GitHub* and *Mails* Please feel welcome to ask questions and participate in all of the venues. The *GitHub* communication channel is for development activities, bug reports, and testing. The *Mails* communication channel is for usage, development discussion.

8.1 GitHub

Visit http://github.com/lutraconsulting/MDAL to file issues you might be having with the software. GitHub is also where you can obtain a current development version of the software in the git revision control system. The MDAL project is eager to take contributions in all forms, and we welcome those who are willing to roll up their sleeves and start filing tickets, pushing code, generating builds, and answering questions.

8.2 Mails

To discuss development and usage, please use mdal-developer@lists.osgeo.org You can subscribe on https://lists.osgeo.org/mailman/listinfo/mdal-developer

CHAPTER

NINE

HOW TO CONTRIBUTE?

9.1 Developer Contributions to MDAL

Install all required development packages: GNU make, g++, ...

Build:

```
cd gdal
./configure [options]
make -j8 -s
cd apps; make -s test_ogrsf; cd ..
```

Run command line utilities (without installing):

```
. scripts/setdevenv.sh
gdalinfo --version
```

Run autotest suite:

```
cd ../autotest
pip install -r requirements.txt
pytest
```

9.1.1 Git workflows with MDAL

This is not a git tutorial or reference manual by any means. This just collects a few best practice for git usage for MDAL development.

9.1.1.1 Commit message

Indicate a component name (eg a driver name), a short description and when relevant, a reference to a issue (with 'fixes #' if it actually fixes it)

```
COMPONENT_NAME: fix bla bla (fixes #1234)

Details here...
```

9.1.1.2 Initiate your work repository

Fork lutraconsulting/MDAL from GitHub UI, and then

```
git clone https://github.com/lutraconsulting/gdal
cd gdal
git remote add my_user_name https://github.com/my_user_name/gdal.git
```

9.1.1.3 Updating your local master against upstream master

```
git checkout master
git fetch origin
# Be careful: this will loose all local changes you might have done now
git reset --hard origin/master
```

9.1.1.4 Working with a feature branch

```
git checkout master
(potentially update your local master against upstream, as described above)
git checkout -b my_new_feature_branch

# do work. For example:
git add my_new_file
git add my_modifid_message
git rm old_file
git commit -a

# you may need to resynchronize against master if you need some bugfix
# or new capability that has been added since you created your branch
git fetch origin
git rebase origin/master

# At end of your work, make sure history is reasonable by folding non
# significant commits into a consistent set
git rebase -i master (use 'fixup' for example to merge several commits together,
and 'reword' to modify commit messages)

# or alternatively, in case there is a big number of commits and marking
# all them as 'fixup' is tedious
git fetch origin
git rebase origin/master
git reset --soft origin/master
git commit -a -m "Put here the synthetic commit message"

# push your branch
git push my_user_name my_new_feature_branch
From GitHub UI, issue a pull request
```

If the pull request discussion or Travis-CI/AppVeyor checks require changes, commit locally and push. To get a reasonable history, you may need to `git rebase -i master`, in which case you will have to force-push your branch with `git push -f my_user_name my_new_feature_branch`

9.1.1.5 Backporting bugfixes from master to a stable branch

```
git checkout master
With git log, identify the sha1sum of the commit you want to backport
git checkout 2.2 (if you want to backport to 2.2)
git pull origin 2.2
(git checkout -b branch_name: if you intend to submit the backport as a pull request)
git cherry-pick the_sha1_sum
git push ...
```

If changes are needed, do them and `git commit -a --amend`

9.1.1.6 Things you should NOT do

(For anyone with push rights to github.com/lutraconsulting/gdal) Never modify a commit or the history of anything that has been committed to https://github.com/lutraconsulting/gdal

9.2 Sphinx RST Style guide

This page contains syntax rules, tips, and tricks for using Sphinx and reStructuredText. For more information, please see this comprehensive guide to reStructuredText, as well as the Sphinx reStructuredText Primer.

9.2.1 Basic markup

9.2.2 Basic markup

A reStructuredText document is written in plain text. Without the need for complex formatting, one can be composed simply, just like one would any plain text document. For basic formatting, see this table:

Format	Syntax	Output
Italics	`*italics*` (single asterisk)	*italics*
Bold	`**bold**` (double asterisk)	**bold**
Monospace	`` ``monospace`` `` (double back quote)	`monospace`

Warning: Use of basic markup is **not recommend**! Where possible use sphinx inline directives to logically mark commands, parameters, options, input, and files. By using directives consistently these items can be styled appropriately.

9.2.3 Lists

There are two types of lists, bulleted lists and numbered lists. A **bulleted list** looks like this:

- An item
- Another item
- Yet another item

This is accomplished with the following code:

```
* An item
* Another item
* Yet another item
```

A **numbered list** looks like this:

1. First item
2. Second item
3. Third item

This is accomplished with the following code:

```
#. First item
#. Second item
#. Third item
```

Note that numbers are automatically generated, making it easy to add/remove items.

9.2.4 List-tables

Bulleted lists can sometimes be cumbersome and hard to follow. When dealing with a long list of items, use list-tables. For example, to talk about a list of options, create a table that looks like this:

Shapes	Description
Square	Four sides of equal length, 90 degree angles
Rectangle	Four sides, 90 degree angles

This is done with the following code:

```
.. list-table::
   :widths: 20 80
   :header-rows: 1

   * - Shapes
     - Description
   * - Square
     - Four sides of equal length, 90 degree angles
   * - Rectangle
     - Four sides, 90 degree angles
```

9.2.5 Page labels

Ensure every page has a label that matches the name of the file. For example if the page is named `foo_bar.rst` then the page should have the label:

```
.. _foo_bar:
```

Other pages can then link to that page by using the following code:

```
:ref:`foo_bar`
```

9.2.6 Linking

Links to other pages should never be titled as "here". Sphinx makes this easy by automatically inserting the title of the linked document.

Bad More information about linking can be found *here*.

Good For more information, please see the section on *Linking*.

To insert a link to an external website:

```
`Text of the link <http://example.com>`__
```

The resulting link would look like this: Text of the link

Warning: It is very easy to have two links with the same text resulting in the following error:

```
**(WARNING/2) Duplicate explicit target name:foo**
```

To avoid these warnings use of a double __ generates an anonymous link.

9.2.7 Sections

Use sections to break up long pages and to help Sphinx generate tables of contents.

```
================================================================================
Document title
================================================================================

First level
-----------

Second level
++++++++++++

Third level
***********

Fourth level
~~~~~~~~~~~~
```

9.2.8 Notes and warnings

When it is beneficial to have a section of text stand out from the main text, Sphinx has two such boxes, the note and the warning. They function identically, and only differ in their coloring. You should use notes and warnings sparingly, however, as adding emphasis to everything makes the emphasis less effective.

Here is an example of a note:

Note: This is a note.

This note is generated with the following code:

```
.. note:: This is a note.
```

Similarly, here is an example of a warning:

Warning: Beware of dragons.

This warning is generated by the following code:

```
.. warning:: Beware of dragons.
```

9.2.9 Images

Add images to your documentation when possible. Images, such as screenshots, are a very helpful way of making documentation understandable. When making screenshots, try to crop out unnecessary content (browser window, desktop, etc). Avoid scaling the images, as the Sphinx theme automatically resizes large images. It is also helpful to include a caption underneath the image.:

```
.. figure:: image.png
   :align: center

   *Caption*
```

In this example, the image file exists in the same directory as the source page. If this is not the case, you can insert path information in the above command. The root `/` is the directory of the `conf.py` file.:

```
.. figure:: /../images/gdalicon.png
```

9.2.10 External files

Text snippets, large blocks of downloadable code, and even zip files or other binary sources can all be included as part of the documentation.

To include link to sample file, use the `download` directive:

```
:download:`An external file <example.txt>`
```

The result of this code will generate a standard link to an `external file`

To include a the contents of a file, use `literalinclude` directive:

```
Example of :command:`gdalinfo` use:

.. literalinclude:: example.txt
```

Example of **gdalinfo** use:

```
example file
```

The `literalinclude` directive has options for syntax highlighting, line numbers and extracting just a snippet:

```
Example of :command:`gdalinfo` use:

.. literalinclude:: example.txt
   :language: txt
   :linenos:
   :emphasize-lines: 2-6
   :start-after: Coordinate System is:
   :end-before: Origin =
```

9.2.11 Reference files and paths

Use the following syntax to reference files and paths:

```
:file:`myfile.txt`
```

This will output: `myfile.txt`.

You can reference paths in the same way:

```
:file:`path/to/myfile.txt`
```

This will output: `path/to/myfile.txt`.

For Windows paths, use double backslashes:

```
:file:`C:\\myfile.txt`
```

This will output: `C:\myfile.txt`.

If you want to reference a non-specific path or file name:

```
:file:`{your/own/path/to}/myfile.txt`
```

This will output: *your/own/path/to*`/myfile.txt`

9.2.12 Reference commands

Reference commands (such as **gdalinfo**) with the following syntax:

```
:program:`gdalinfo`
```

Use `option` directive for command line options:

```
.. option:: -json

   Display the output in json format.
```

Use `describe` to document create parameters:

```
.. describe:: WORLDFILE=YES

   Force the generation of an associated ESRI world file (with the extension .wld).
```

CHAPTER

TEN

FAQ

10.1 What does MDAL stand for?

MDAL - Mesh Data Abstraction Library

10.2 When would I use it?

Most often this kind of representation is used when preparing data for simulation software or when viewing results of physical simulations, typically for meteorology, oceanography, hydrological or hydraulic models. All computation in such software is done on meshes, with values (physical quantities) usually stored in vertices (less commonly in edges or faces). Results usually comprise of various quantities (e.g. wind speed, water depth) which may be also time-varying (e.g. calculated water flow estimates in 5 minute intervals).

Some of the modelling software packages are free and open source (e.g. AnuGA, EPANET), there are some freeware packages (e.g. Basement, HEC-RAS) as well as many commercial pieces of software.

10.3 When was the MDAL project started?

In 2012, Peter Wells and Saber Razmjooei started QGIS Crayfish plugin.

In 2018, Lutra Consulting Ltd. ported and rewritten some parts of Crayfish plugin for separate C++ library that was used in the early QGIS 3.x released as base for QGIS's Mesh Layer.

10.4 Is MDAL proprietary software?

No, MDAL is a Free and Open Source Software.

10.5 What license does MDAL use?

See *License*

10.6 What operating systems does MDAL run on?

You can use MDAL on all modern flavors of Unix: Linux, FreeBSD, Mac OS X; all supported versions of Microsoft Windows; mobile environements (Android and iOS).

10.7 Is there a graphical user interface to MDAL?

See *Software using MDAL*

10.7.1 Software using MDAL

- QGIS A cross platform desktop GIS.
- ViRGIS A Unity based GIS in VR platform.

10.8 What compiler can I use to build MDAL?

MDAL can be compiled with a C++11 capable compiler.

10.9 I have a question that's not answered here. Where can I get more information?

See *Community*

Keep in mind, the quality of the answer you get does bear some relation to the quality of the question. If you need more detailed explanation of this, you can find it in essay How To Ask Questions The Smart Way by Eric S. Raymond.

10.10 How do I add support for a new format?

To some extent this is now covered by the *MDAL driver implementation tutorial*

10.11 How do I cite MDAL ?

See CITATION

CHAPTER

ELEVEN

LICENSE

11.1 License

MIT License

Copyright (c) 2018 Lutra Consulting Ltd.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The full licensing terms are available in the *LICENSE* file.

INDEX

Symbols

M

T