SEGY

TGS

May 07, 2024

GETTING STARTED

This project is under active development, expect breaking changes the to API - *March, 2024*

This is an efficient and comprehensive SEG-Y parsing library.

See the documentation for more information.

This is not an official TGS product.

CHAPTER

ONE

FEATURES

The library utilizes `numpy` and `fsspec`, includes the reading from various local and remote resources at a high speed. It also allows the users to build their own SEG-Y specifications.

CHAPTER

TWO

INSTALLING SEGY

Clone the repo and install it using pip:

Simplest way to install `segy` is via pip from PyPI:

```
$ pip install segy
```

or install `segy` via conda from conda-forge:

```
$ conda install -c conda-forge segy
```

Extras must be installed separately on `Conda` environments.

For details, please see the *installation instructions* in the documentation.

CHAPTER

THREE

USING SEGY

Please see the *Command-line Usage* for details.

For Python API please see the *API Reference* for details.

3.1 Reading Capabilities

It supports reading from local and cloud files (object store). It can read:

- Sequential traces (fastest)
- Disjoint sequential regions (fast)
- Random traces (slow)

3.2 High Performance

The performance is high and to be proven with upcoming benchmarks. The initial subjective benchmarks is very acceptable.

3.3 Flexibility

The library provides a fully flexible, schematized SEG-Y structure, including data models and JSON schema parsing and validation.

3.4 Predefined SEG-Y Standards

It supports predefined SEG-Y "standards" for various versions. However, some versions are still in progress:

- [x] Rev 0 (1975)
- [x] Rev 1 (2002)
- [ ] Rev 2 (2017)
- [ ] Rev 2.1 (2023)

3.5 Custom SEG-Y Standards

You can build your own SEG-Y "standard" with composition of specs for:

- Text header (file + extended)
- Binary header
- Traces (header + extended header + samples)

CHAPTER

FOUR

CONTRIBUTING TO SEGY

Contributions are very welcome. To learn more, see the *Contributor Guide*.

CHAPTER

FIVE

LICENSING

Distributed under the terms of the *Apache 2.0 license*. `segy` is free and open source software.

CHAPTER

SIX

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

CHAPTER

SEVEN

CREDITS

This project was established at TGS. Current maintainer is Altay Sansal with the support of many more great colleagues.

The CI/CD tooling is loosely based on Hypermodern Python Cookiecutter with more modern tooling applied elsewhere.

7.1 Installation

There are different ways to install `segy`:

- Install the latest release via *`pip`* or *`conda`*.
- Building package *from source*.

Note: We strongly recommend using a virtual environment `venv` or `conda` to avoid potential conflicts with other Python packages.

7.1.1 Using `pip` and `virtualenv`

Install the 64-bit version of Python 3 from https://www.python.org.

Then we can create a `venv` and install `segy`.

```
$ python -m venv segy-venv
$ segy-venv/Scripts/activate
$ pip install -U segy
```

You can also install some optional dependencies (extras) like this:

```
$ pip install segy[cloud]
```

`cloud` installs fsspec backed I/O libraries for AWS' S3, Google's GCS, and Azure ABS.\

To check if installation was successful see *checking installation*.

7.1.2 Using conda

segy can also be installed in a conda environment.

Note: segy is hosted in the conda-forge channel. Make sure to always provide the -c conda-forge when running conda install or else it won't be able to find the package.

We first run the following to create and activate an environment:

```
$ conda create -n segy-env
$ conda activate segy-env
```

Then we can to install with conda:

```
$ conda install -c conda-forge segy
```

The above command will install segy into your conda environment.

Note: segy extras must be installed separately when using conda.

7.1.3 Checking Installation

After installing segy, run the following:

```
$ python -c "import segy; print(segy.__version__)"
```

You should see the version of segy printed to the screen.

7.1.4 Building from Source

All dependencies of segy are Python packages, so the build process is very simple. To install from source, we need to clone the repo first and then install locally via pip.

```
$ git clone https://github.com/TGSAI/segy.git
$ cd segy
$ pip install .
```

We can also install the extras in a similar way, for example:

```
$ pip install .[cloud]
```

If you want an editable version of segy then we could install it with the command below. This does allow you to make code changes on the fly.

```
$ pip install --editable .[cloud]
```

To check if installation was successful see *checking installation*.

7.2 Command-Line Usage

7.2.1 Introduction

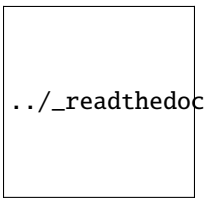
`segy` comes with a useful CLI tool to interrogate SEG-Y files either on disk or any remote store.

7.2.2 Command Line Usage

SEG-Y provides a convenient command-line-interface (CLI) to do various tasks.

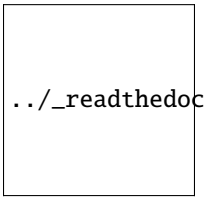
For each command / subcommand you can provide `--help` argument to get information about usage.

At the highest level, the `segy` command line offers various options to choose from. Below you can see the usage for the main entry point.



Dumping Data

When we use `segy dump` subcommand, we have some options to choose from. As usual, the `uri` (local or remote paths) will allow us to use the same toolkit for local and cloud / web files.



For the CLI demos, we will use a public SEG-Y file located in Amazon Web Services' (AWS) Simple Storage Service (S3), also known as a cloud object store.

This dataset, the Stratton 3D is made available for worldwide education and training by the Bureau of Economic Geology at the University of Texas at Austin. Available information and data acquisition details are accessible via the SEG Wiki.

We will take a look at the 3D unprocessed shot gathers (swath 1).

Note: To run the below examples, set your S3 configuration. More details in *Configuration Options*.

```
export SEGY__STORAGE_OPTIONS='{"anon": true}'
```

Basic Info

Now that we can access public S3 buckets anonymously, we can output a basic summary of the file using the `info` command.

```
$ segy dump info \
    s3://open.source.geoscience/open_data/stratton/segy/navmerged/swath_1_geometry.sgy
{
  "uri": "s3://open.source.geoscience/open_data/stratton/segy/navmerged/swath_1_geometry.
↪sgy",
  "segyStandard": 0.0,
  "numTraces": 136530,
  "samplesPerTrace": 3000,
  "sampleInterval": 2000,
  "fileSize": 1671130800
}
```

File Text Header

Let's take a look at the text header.

```
$ segy dump text-header \
    s3://open.source.geoscience/open_data/stratton/segy/navmerged/swath_1_geometry.sgy
C 1 CLIENT: BUREAU OF ECONOMIC GEOLOGY  COMPANY: HALLIBURTON GS CREW: #1768
C 2 SURVEY: WARDNER LEASE 3-D (STRATTON FIELD)       AREA: NUECES CO, TEXAS
C 3 RECORDING DATE: 1992
C 4 2MS SAMPLE INTERVAL   3000 SAMPLES/TRACE   4 BYTES/SAMPLE
C 5
C 6 DATA STORED AS SEG-Y FORMAT #1 (IBM FLOATING POINT)
C 7 KEY STANDARD TRACE HEADERS USED:
C 8 FFID =  9-12
C 9 SOURCE X = 73-76  SOURCE Y = 77-80 SOURCE Z = 45-48
C10    REC X = 81-84     REC Y = 85-88    REC Z = 41-44
C11 COORD SCALER = 71-72   ELEV. SCALER = 69-70
C12
C13 NOTE: X = NORTHING, Y = EASTING (RIGHT-HAND Z-DOWN COORDINATES)
C14
C15 NON-STANDARD TRACE HEADERS:
C16 CHANNEL  = 25-28
C17    CHANNELS 1-720 ARE LIVE DATA, 996-999 ARE AUXILIARY TRACES
C18 RECEIVER LINE = 181-184  RECEIVER NUMBER = 185-188
C19 SOURCE LINE = 189-192  SOURCE NUMBER = 193-196
C20
C21 PROCESSED BY EGL: EXPLORATION GEOPHYSICS LABORATORY (PAUL E MURRAY)
C22 BUREAU OF ECONOMIC GEOLOGY, JACKSON SCHOOL OF GEOSCIENCES, UT - AUSTIN
C23 **********PROCESSING**********
C24 1) FIELD SEG-Y FILES REFORMAT TO EGLTOOLS SDF FORMAT
C25 2) CHANNEL RENUMBERING AND GEOMETRY LOADED TO HEADERS
C26 3) PADDED TRACES, BAD AND TEST RECORDS REMOVED
C27 4) REFORMAT TO SEG-Y
C28
C29
```

(continues on next page)

(continued from previous page)

```
C30 SWATH 1 OF 4 CONTAINS FFIDS 1-262
C31
C32 COORDINATES ARE IN FEET
C33 ELLIPSOID: CLARKE 1866
C34 DATUM = NAD27
C35 TEXAS STATE PLANE SOUTH ZONE, LAMBERT PROJECTION
C36 FALSE NORTHING =  485012.85
C37 FALSE EASTING =  2000000.00
C38
C39 written from EGLTools for Matlab on 14-Dec-2009
C40 END EBCDIC
```

File Binary Header

```
$ segy dump binary-header \
    s3://open.source.geoscience/open_data/stratton/segy/navmerged/swath_1_geometry.sgy
{
  "job_id": 0,
  "line_no": 0,
  "reel_no": 1,
  "data_traces_ensemble": 724,
  "aux_traces_ensemble": 0,
  "sample_interval": 2000,
  "sample_interval_orig": 2000,
  "samples_per_trace": 3000,
  "samples_per_trace_orig": 3000,
  "data_sample_format": 1,
  "ensemble_fold": 724,
  "trace_sorting": 1,
  "vertical_sum": 0,
  "sweep_freq_start": -30480,
  "sweep_freq_end": -2692,
  "sweep_length": 0,
  "sweep_type": 0,
  "sweep_trace_no": 0,
  "sweep_taper_start": 0,
  "sweep_taper_end": 0,
  "taper_type": 0,
  "correlated_traces": 0,
  "binary_gain": 0,
  "amp_recovery_method": 0,
  "measurement_system": 0,
  "impulse_signal_polarity": 0,
  "vibratory_polarity": 0
}
```

Trace Header

This is how we can get three header fields for a few traces.

```
$ segy dump trace-header s3://open.source.geoscience/open_data/stratton/segy/navmerged/
↪swath_1_geometry.sgy \
    --index 100 --index 101 --index 500 --index 501 \
    --field src_x --field src_y \
    --field rec_x --field rec_y \
    --field scalar_apply_coords
                src_x      src_y     rec_x      rec_y  scalar_apply_coords
trace_index
100          70628086  219412572  70616707  218875760                 -100
101          70628086  219412572  70616695  218864765                 -100
500          70650057  219412488  70880968  219271571                 -100
501          70650057  219412488  70880940  219260587                 -100
```

7.2.3 Configuration Options

When accessing public datasets from S3, we need to set `SegyFileSettings().storage_options = {"anon": True}` for anonymous access. *SegyFileSettings* exposes all configuration options as environment variables. We just need to set `storage_options` with the `JSON` string `{"anon": true}`. On Linux you can do this by the command below. Environment variables can be configured in many ways, please refer to the options for your specific Operating System (OS).

```
export SEGY__STORAGE_OPTIONS='{"anon": true}'
```

See also:

Settings Management

7.3 Tutorials

7.3.1 Read Data from the Cloud

Altay Sansal

May 07, 2024

9 min read

In this tutorial, we will use a public SEG-Y file located in Amazon Web Services' (AWS) Simple Storage Service (S3), also known as a cloud object store.

This dataset, the Parihaka 3D full angle stack (4.7 GB per volume including full angle and near, mid, and far stacks), is provided by New Zealand Petroleum and Minerals (NZPM). Available information and data acquisition details are accessible via the SEG Wiki, the New Zealand GNS website, and the NZPM data portal.

Important: For plotting, the notebook requires Matplotlib as a dependency. Please install it before executing using `pip install matplotlib` or `conda install matplotlib`.

Let's start by importing some modules we will be using.

```
import json

import matplotlib.pyplot as plt
from IPython.display import JSON
from numpy.random import default_rng

from segy import SegyFile
from segy.config import SegyFileSettings
from segy.schema import StructuredFieldDescriptor
from segy.standards import rev1_segy
```

You can (but don't) download the SEG-Y directly clicking the HTTP link from the website.

This link is convenient as the `segy` library supports HTTP and we can directly use it without downloading as well. Hovewer, For demonstration purposes, we'll use the corresponding S3 link (or called bucket and prefix):

`s3://open.source.geoscience/open_data/newzealand/Taranaiki_Basin/PARIHAKA-3D/Parihaka_PSTM_full_angle.sgy`

It's important to note that the file isn't downloaded but rather read on demand from the S3 object store with the `segy` library.

The `SegyFile` class uses information from the binary file header to construct a SEG-Y descriptor, allowing it to read the file. The SEG-Y Revision is inferred from the binary header by default, but can be manually set by adjusting the `revision` setting.

Since this is a public bucket and an object, we need to tell `S3` that we want anonymous access, which is done by configuring `storage_options` in settings.

```
path = "s3://open.source.geoscience/open_data/newzealand/Taranaiki_Basin/PARIHAKA-3D/
↪Parihaka_PSTM_full_angle.sgy"

# Alternatively via HTTP
# path = "http://s3.amazonaws.com/open.source.geoscience/open_data/newzealand/Taranaiki_
↪Basin/PARIHAKA-3D/Parihaka_PSTM_full_angle.sgy"

settings = SegyFileSettings(storage_options={"anon": True})

sgy = SegyFile(path, settings=settings)
```

Let's investigate the JSON version of the descriptor.

Some things to note:

1. The opening processed inferred Revision 1 from the binary header automatically.
2. It generated the schema using **default** SEG-Y Revision 1 headers.
3. Some headers can be defined in the wrong byte locations, we can check that.
4. Way to many headers to deal with in the default schema.

Note that we can build this JSON independently, load it into the descriptor and open any SEG-Y with a shema.

```
JSON(json.loads(sgy.spec.model_dump_json()))
```

```
<IPython.core.display.JSON object>
```

Let's check the file size, number of traces, sample rate, etc. As expected, the file size matches what was in the description. We also observe there are ~ 1 million traces.

```
print(f"file size: {sgy.file_size / 1024**3:0.2f} GiB")
print(f"num traces: {sgy.num_traces:,}")
print(f"sample rate: {sgy.sample_interval}")
print(f"num samples: {sgy.samples_per_trace}")
print(f"sample labels: {sgy.sample_labels // 1000}")  # microsecond to millisecond
```

```
file size: 4.75 GiB
num traces: 1,038,162
sample rate: 3000
num samples: 1168
sample labels: [   0    3    6 ... 3495 3498 3501]
```

Using the `SegyFile` we can read SEG-Y components.

Here we read:

- Textual file header
- Binary file header
- 1,000 traces (headers + data) from somewhere in the middle of the file

```
text_header = sgy.text_header
binary_header = sgy.binary_header

start = 500_000
stop = start + 1_000

traces = sgy.trace[start:stop]

trace_headers = traces.header
trace_data = traces.sample
```

This should take around one second or less, based on internet connection.

Let's print the textual header. There are a not many headers of interest. The available headers appear to be in the Revision 1 byte locations.

```
print(text_header)
```

```
C 1 3D VOLUME
C 2 HEADER BYTE LOCATIONS AND TYPES:
C 3     3D INLINE : 189-192 (4-BYTE INT)    3D CROSSLINE: 193-196 (4-BYTE INT)
C 4     ENSEMBLE X: 181-184 (4-BYTE INT)    ENSEMBLE Y  : 185-188 (4-BYTE INT)
C 5
C 6 SAMPLES/TRACE     :       1168
C 7 SAMPLE INTERVAL   :       3000 microseconds
C 8 FIRST SAMPLE AT   :          0 ms
C 9 VERTICAL DIMENSION: TWT (ms)
C10 SAMPLE RECORDING FORMAT: IBM FLOATING POINT (4-BYTE)
C11
C12
C13
```

(continues on next page)

(continued from previous page)

```
C14
C15
C16
C17
C18
C19
C20
C21
C22
C23
C24
C25
C26
C27
C28
C29
C30
C31
C32
C33
C34
C35
C36
C37
C38 WRITTEN BY INSIGHT VERSION 3.0 (405040)              http://www.dugsw.com/
C39 SEG Y REV1
C40 END TEXTUAL HEADER
```

We can look at headers (by default it is a Pandas `DataFrame`) in a nicely formatted table.

We can also do typical Pandas analytics (like plots, statistics, etc.) but it won't be shown here.

```
binary_header.to_dataframe()
```

```
   job_id  line_no  reel_no  data_traces_ensemble  aux_traces_ensemble  \
0       0        0        0                     0                    0

   sample_interval  sample_interval_orig  samples_per_trace  \
0             3000                     0               1168

   samples_per_trace_orig  data_sample_format  ...  correlated_traces  \
0                       0                   1  ...                  0

   binary_gain  amp_recovery_method  measurement_system  \
0            0                    0                   0

   impulse_signal_polarity  vibratory_polarity  seg_y_revision  \
0                        0                   0             256

   fixed_length_trace_flag  extended_textual_headers  additional_trace_headers
0                        1                         0                         0

[1 rows x 31 columns]
```

```
trace_headers.to_dataframe()
```

```
     trace_seq_line  trace_seq_file  field_rec_no  trace_no_field_rec  \
0                 0               0             0                   0
1                 0               0             0                   0
2                 0               0             0                   0
3                 0               0             0                   0
4                 0               0             0                   0
..              ...             ...           ...                 ...
995               0               0             0                   0
996               0               0             0                   0
997               0               0             0                   0
998               0               0             0                   0
999               0               0             0                   0

     energy_src_pt  cdp_ens_no  trace_no_ens  trace_id  vert_sum  horiz_stack  \
0                0           0             0         0         0            0
1                0           0             0         0         0            0
2                0           0             0         0         0            0
3                0           0             0         0         0            0
4                0           0             0         0         0            0
..             ...         ...           ...       ...       ...          ...
995              0           0             0         0         0            0
996              0           0             0         0         0            0
997              0           0             0         0         0            0
998              0           0             0         0         0            0
999              0           0             0         0         0            0

     ...  transduction_constant_exponent  transduction_units  device_trace_id  \
0    ...                               0                  0                0
1    ...                               0                  0                0
2    ...                               0                  0                0
3    ...                               0                  0                0
4    ...                               0                  0                0
..   ...                             ...                ...              ...
995  ...                               0                  0                0
996  ...                               0                  0                0
997  ...                               0                  0                0
998  ...                               0                  0                0
999  ...                               0                  0                0

     times_scalar  source_type_orientation  source_energy_direction_mantissa  \
0               0                        0                                 0
1               0                        0                                 0
2               0                        0                                 0
3               0                        0                                 0
4               0                        0                                 0
..            ...                      ...                               ...
995             0                        0                                 0
996             0                        0                                 0
997             0                        0                                 0
998             0                        0                                 0
```

(continues on next page)

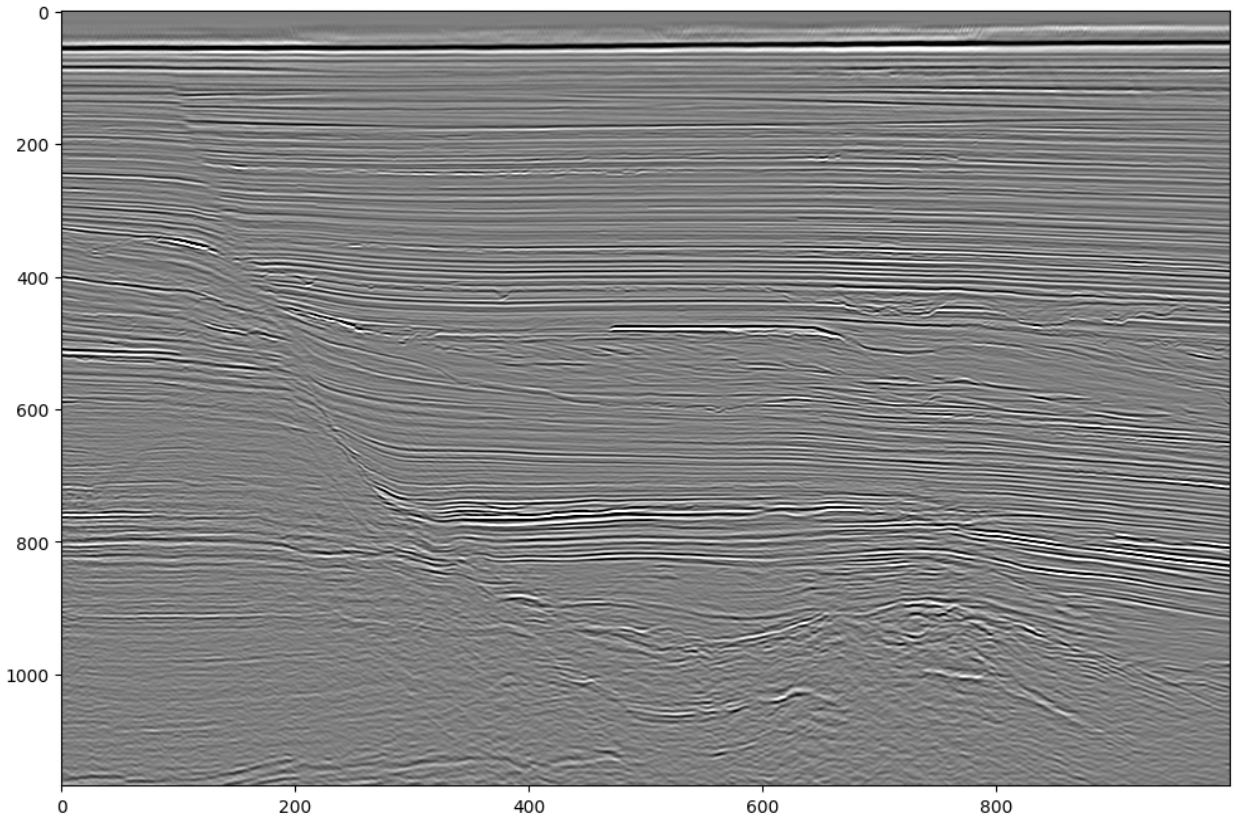
(continued from previous page)

```
999             0                         0                                    0

     source_energy_direction_exponent  source_measurement_mantissa  \
0                                   0                            0
1                                   0                            0
2                                   0                            0
3                                   0                            0
4                                   0                            0
..                                ...                          ...
995                                 0                            0
996                                 0                            0
997                                 0                            0
998                                 0                            0
999                                 0                            0

     source_measurement_exponent  source_measurement_unit
0                              0                       0
1                              0                       0
2                              0                       0
3                              0                       0
4                              0                       0
..                           ...                     ...
995                            0                       0
996                            0                       0
997                            0                       0
998                            0                       0
999                            0                       0

[1000 rows x 89 columns]
```

Let's plot the traces.

Note that they are all parsed from IBM floats to IEEE floats (decoded) in the background.

```
plt.figure(figsize=(12, 8))
plot_kw = {"aspect": "auto", "cmap": "gray_r", "interpolation": "bilinear"}
plt.imshow(trace_data.T, vmin=-1000, vmax=1000, **plot_kw);
```



With Custom Schema

We will create a new custom schema based on SEG-Y revision 1 with different binary and trace headers. This way we will parse ONLY the parts we want, with correct byte locations.

A user can define a completely custom SEG-Y schema from scratch as well, but for convenience, we are customizing Revision 1 schema with the parts that we want to customize.

Note that doing this will also modify the `segyStandard` field to `"custom"` to make sure we don't assume the file schema is standard after doing this.

From the binary file header we will read:

- Number of samples
- Sample rate

From the trace headers, we will read:

- Inline
- Crossline
- CDP-X
- CDP-Y
- Coordinate scalar

Based on the text header lines:

```
C 2 HEADER BYTE LOCATIONS AND TYPES:
C 3     3D INLINE : 189-192 (4-BYTE INT)    3D CROSSLINE: 193-196 (4-BYTE INT)
C 4     ENSEMBLE X: 181-184 (4-BYTE INT)    ENSEMBLE Y  : 185-188 (4-BYTE INT)
```

And we know by SEG-Y Rev1 definition, the coordinate scalars are at byte 71.

```
custom_spec = rev1_segy.customize(
    binary_header_fields=[
        StructuredFieldDescriptor(name="sample_interval", offset=16, format="int16"),
        StructuredFieldDescriptor(name="samples_per_trace", offset=20, format="int16"),
    ],
    trace_header_fields=[
        StructuredFieldDescriptor(name="inline", offset=188, format="int32"),
        StructuredFieldDescriptor(name="crossline", offset=192, format="int32"),
        StructuredFieldDescriptor(name="cdp-x", offset=180, format="int32"),
        StructuredFieldDescriptor(name="cdp-y", offset=184, format="int32"),
        StructuredFieldDescriptor(name="coordinate_scalar", offset=70, format="int16"),
    ],
)

sgy = SegyFile(path, spec=custom_spec, settings=settings)
```

Now let's look at the JSON for the desciptor again. It is a lot more compact.

```
JSON(json.loads(sgy.spec.model_dump_json()))
```

```
<IPython.core.display.JSON object>
```

As mentioned earlier, the JSON can be laded into the descriptor from a file too.

```
from segy.schema.segy import SegyDescriptor
import os

json_path = "..."

with open(json_path, mode="r") as fp:
    data = fp.read()
    spec = SegyDescriptor.model_validate_json(data)
```

Let's do something a little more interesting now. Let's try to plot a time slice by randomly sampling the file.

We will read 5,000 random traces. This should take about 15-20 seconds, based on your internet connection speed.

```
text_header = sgy.text_header
binary_header = sgy.binary_header

rng = default_rng()
indices = rng.integers(0, sgy.num_traces, size=5_000).tolist()

traces = sgy.trace[indices]
```

```
binary_header.to_dataframe()
```

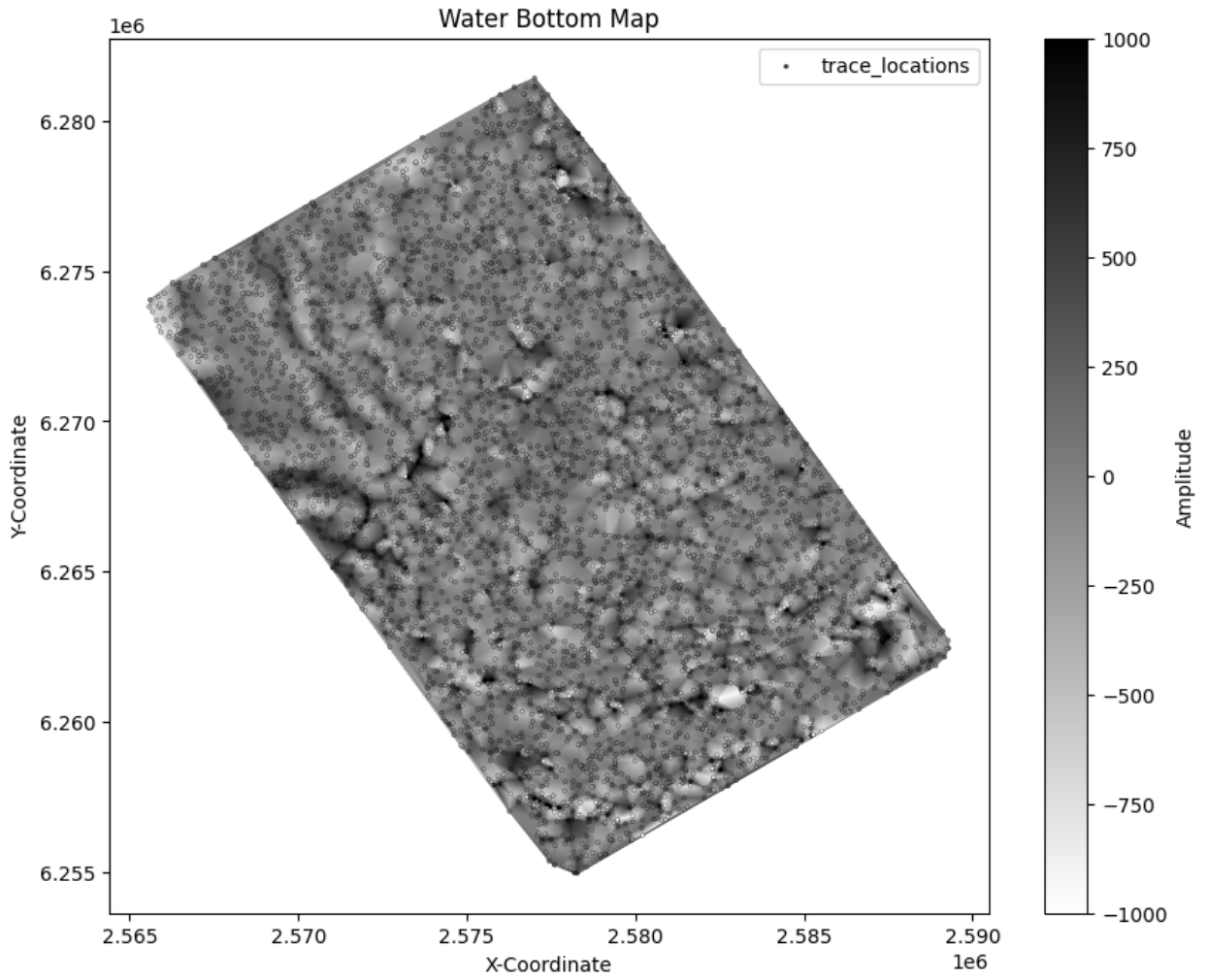
```
   sample_interval  samples_per_trace
0             3000               1168
```

```
trace_headers = traces.header.to_dataframe()

trace_headers["cdp-x"] /= trace_headers["coordinate_scalar"].abs()
trace_headers["cdp-y"] /= trace_headers["coordinate_scalar"].abs()

trace_headers
```

```
      inline  crossline      cdp-x      cdp-y  coordinate_scalar
0       1736       4246  2578274.0  6254964.0                  1
1       1736       4431  2580213.0  6256224.0                  1
2       1736       4673  2582750.0  6257871.0                  1
3       1736       4865  2584763.0  6259178.0                  1
4       1736       5261  2588914.0  6261874.0                  1
...      ...        ...        ...        ...                ...
4985    2656       4691  2570412.0  6277283.0                  1
4986    2657       4297  2566268.0  6274621.0                  1
4987    2657       4383  2567169.0  6275207.0                  1
4988    2657       4416  2567515.0  6275432.0                  1
4989    2657       5004  2573680.0  6279435.0                  1

[4990 rows x 5 columns]
```

Now we can plot the time slice on real coordinates from the headers, and even see a hint of the outline of the data! Since we significantly down- sampled the data, the time slice is aliased and not very useful, but this shows us the concept of making maps.

```
plt.figure(figsize=(10, 8))

z_slice_index = 500

x, y, z = (
    trace_headers["cdp-x"],
    trace_headers["cdp-y"],
    traces.sample[:, z_slice_index],
)

scatter_kw = {"ec": [0.0, 0.0, 0.0, 0.5], "linewidth": 0.5}
color_kw = {"cmap": "gray_r", "vmin": -1000, "vmax": 1000}

plt.tripcolor(x, y, z, shading="gouraud", **color_kw)
plt.scatter(x, y, s=4, c=z, label="trace_locations", **scatter_kw, **color_kw)
plt.title("Water Bottom Map")
plt.colorbar(label="Amplitude")
plt.xlabel("X-Coordinate")
plt.ylabel("Y-Coordinate")
plt.legend();
```



7.3.2 Creating a New SEG-Y

Altay Sansal

May 07, 2024

2 min read

In this tutorial, we create a new SEG-Y file from spec.

Let's start by importing some modules we will be using.

```
from segy.factory import SegyFactory
from segy.standards.rev1 import rev1_segy
```

We will take the default SEG-Y Revision 1 specification.

The `SegyFactory` takes the spec, number of samples, and sample interval as inputs. By using its creation functions, we can make the encoded (ready to write to disk) bytes for file headers (text header and binary header).

```
SAMPLE_INTERVAL = 4000  # in microseconds
SAMPLES_PER_TRACE = 101
```

(continues on next page)

(continued from previous page)

```
factory = SegyFactory(
    rev1_segy, sample_interval=SAMPLE_INTERVAL, samples_per_trace=SAMPLES_PER_TRACE
)

txt = factory.create_textual_header()
bin_ = factory.create_binary_header()
```

Let's create 15 traces and populate their values. Headers by default will be populated by sample rate and number of samples. We will set some fake headers. We will also fill in the trace samples with `trace_no + sample_index`.

```
TRACE_COUNT = 15

headers = factory.create_trace_header_template(size=TRACE_COUNT)
samples = factory.create_trace_sample_template(size=TRACE_COUNT)

for trace_idx in range(TRACE_COUNT):
    headers[trace_idx]["trace_seq_file"] = trace_idx + 1
    headers[trace_idx]["x_coordinate"] = 1_000
    headers[trace_idx]["y_coordinate"] = 10_000 + trace_idx * 50
    headers[trace_idx]["inline_no"] = 10
    headers[trace_idx]["crossline_no"] = 100 + trace_idx

    samples[trace_idx] = range(SAMPLES_PER_TRACE)  # sample index
    samples[trace_idx] += trace_idx  # trace no
```

Now we can create the encoded binary values for traces (ready to write).

```
traces = factory.create_traces(samples=samples, headers=headers)
```

We can now compose a binary SEG-Y file from pieces.

We create a new `my_segy.sgy` file and write the pieces we built.

```
from pathlib import Path

with Path("my_segy.sgy").open(mode="wb") as fp:
    fp.write(txt)
    fp.write(bin_)
    fp.write(traces)
```

Opening New SEG-Y

Now we can open it with `SegyFile`.

Note that our factory correctly populated the revision number in the header so the spec is automatically inferred!

```
from segy.file import SegyFile

file = SegyFile("my_segy.sgy")
print(file.text_header)
```

```
C01 File written by the open-source segy library.
C02
C03 Website: https://segy.readthedocs.io
C04 Source: https://github.com/TGSAI/segy
C05
C06
C07
C08
C09
C10
C11
C12
C13
C14
C15
C16
C17
C18
C19
C20
C21
C22
C23
C24
C25
C26
C27
C28
C29
C30
C31
C32
C33
C34
C35
C36
C37
C38
C39
C40 END TEXTUAL HEADER
```

```
file.binary_header.to_dataframe()
```

```
   job_id  line_no  reel_no  data_traces_ensemble  aux_traces_ensemble  \
0       0        0        0                     0                    0

   sample_interval  sample_interval_orig  samples_per_trace  \
0             4000                  4000                101

   samples_per_trace_orig  data_sample_format  ...  correlated_traces  \
0                     101                   0  ...                  0
```

(continues on next page)

(continued from previous page)

```
   binary_gain  amp_recovery_method  measurement_system  \
0            0                    0                   0

   impulse_signal_polarity  vibratory_polarity  seg_y_revision  \
0                        0                   0             256

   fixed_length_trace_flag  extended_textual_headers  additional_trace_headers
0                        0                         0                         0

[1 rows x 31 columns]
```

```
file.sample[:]
```

```
array([[  0.,   1.,   2., ...,  98.,  99., 100.],
       [  1.,   2.,   3., ...,  99., 100., 101.],
       [  2.,   3.,   4., ..., 100., 101., 102.],
       ...,
       [ 12.,  13.,  14., ..., 110., 111., 112.],
       [ 13.,  14.,  15., ..., 111., 112., 113.],
       [ 14.,  15.,  16., ..., 112., 113., 114.]], dtype=float32)
```

```
show_fields = [
    "trace_seq_file",
    "x_coordinate",
    "y_coordinate",
    "inline_no",
    "crossline_no",
]

file.header[:][show_fields].to_dataframe()
```

```
    trace_seq_file  x_coordinate  y_coordinate  inline_no  crossline_no
0                1          1000         10000         10           100
1                2          1000         10050         10           101
2                3          1000         10100         10           102
3                4          1000         10150         10           103
4                5          1000         10200         10           104
5                6          1000         10250         10           105
6                7          1000         10300         10           106
7                8          1000         10350         10           107
8                9          1000         10400         10           108
9               10          1000         10450         10           109
10              11          1000         10500         10           110
11              12          1000         10550         10           111
12              13          1000         10600         10           112
13              14          1000         10650         10           113
14              15          1000         10700         10           114
```

7.4 Settings Management

Altay Sansal

May 07, 2024

3 min read

7.4.1 `SegyFileSettings` Class

The *SegyFileSettings* is a configuration object for the *SegyFile* in the environment. It allows you to customize various aspects of SEG-Y file parsing according to your needs and the specifics of your project.

It is composed of various sub-settings isolated by SEG-Y components and various topics.

- **binary**: The *SegyBinaryHeaderSettings* is used for binary header configuration while reading a SEG-Y file.
- **endian**: This setting determines the byte order that is being used in the SEG-Y file. The possible options are `"big"` or `"little"` based on *Endianness*. If left as None, the system defaults to Big Endian (`"big"`).
- **revision**: This setting is used to specify the SEG-Y revision number. If left as None, the system will automatically use the revision mentioned in the SEG-Y file.
- **use_pandas**: This setting is a boolean that decides whether to use pandas for headers or not. Does not apply to trace data. The trace data is always returned as Numpy arrays. The option to use Numpy for headers is currently disabled and will be available at a later release (as of March 2024).

7.4.2 Usage

You initialize an instance of *SegyFileSettings* like any other Python object, optionally providing initial values for the settings. For example:

```
from segy.config import SegyBinaryHeaderSettings
from segy.config import SegyFileSettings
from segy.schema import Endianness


# Override extended text header count to zero
binary_header_settings = SegyBinaryHeaderSettings(
    extended_text_header={"value": 0}
)

settings = SegyFileSettings(
    binary=binary_header_settings,
    endian=Endianness.LITTLE,
    revision=1,
)
```

Then this can be passed to *SegyFile* directly.

```
from segy import SegyFile

file = SegyFile(uri="...", settings=settings)
```

If no settings are provided to *SegyFile*, it will take the default values.

See also:

SegyFileSettings, *SegyFile*, *Endianness*

7.4.3 Environment Variables

Environment variables that follow the `SEGY__VARIABLE__SUBVARIABLE` format will be automatically included in your *SegyFileSettings* instance:

```
export SEGY__BINARY__SAMPLES_PER_TRACE__VALUE=1001
export SEGY__BINARY__SAMPLE_INTERVAL__KEY="my_custom_key_in_schema"
export SEGY__ENDIAN="big"
export SEGY__REVISION=0.0
```

The environment variables will override the defaults in the *SegyFileSettings* configuration, unless user overrides it again within Python.

7.5 API Reference

7.5.1 SEG-Y File

class segy.**SegyFile**(*url*, *spec=None*, *settings=None*)

A SEG-Y file class that has various accessors.

Parameters

- **url** (*str*) – Path to SEG-Y file on disk or remote store.
- **spec** (*SegyDescriptor | None*) – The schema / spec describing the SEG-Y file. This is optional and by default it will try to infer the SEG-Y standard from the binary header.
- **settings** (*SegyFileSettings | None*) – A settings instance to configure / override the SEG-Y parsing logic. Optional.

property binary_header: HeaderArray

Read binary header from store, based on spec.

property file_size: int

Return file size in bytes.

property header: HeaderIndexer

Way to access the file to fetch trace headers only.

property num_ext_text: int

Return number of extended text headers.

property num_traces: int

Return number of traces in file based on size and spec.

property sample: AbstractIndexer

Way to access the file to fetch trace data only.

property sample_interval: int

Return samples interval in file based on spec.

property sample_labels: NDArray[np.int32]

Return sample axis labels.

property samples_per_trace: int

Return samples per trace in file based on spec.

property text_header: str

Return textual file header.

property trace: TraceIndexer

Way to access the file to fetch full traces (headers + data).

7.5.2 SEG-Y Factory

class segy.**SegyFactory**(*spec*, *sample_interval=4000*, *samples_per_trace=1500*)

Factory class for composing SEG-Y by components.

Parameters

- **spec** (SegyDescriptor) – SEG-Y specification.
- **sample_interval** (*int*) – Sample interval to use in new file.
- **samples_per_trace** (*int*) – Number of samples per trace.

create_binary_header()

Create a binary header for the SEG-Y file.

Returns
Bytes containing the encoded binary header, ready to write.

Return type
bytes

create_textual_header(*text=None*)

Create a textual header for the SEG-Y file.

The length of the text should match the rows and columns in the spec's TextHeaderDescriptor. The newlines must also be in the text to separate the rows.

Parameters
text (*str | None*) – String containing text header rows. If left as None, a default textual header will be created.

Returns
Bytes containing the encoded text header, ready to write.

Return type
bytes

create_trace_header_template(*size=1*)

Create a trace header template array that conforms to the SEG-Y spec.

Parameters
size (*int*) – Number of headers for the template.

Returns
Array containing the trace header template.

Return type
NDArray[Any]

`create_trace_sample_template`(*size=1*)

Create a trace data template array that conforms to the SEG-Y spec.

Parameters
size (*int*) – Number of traces for the template.

Returns
Array containing the trace data template.

Return type
NDArray[Any]

`create_traces`(*headers*, *samples*)

Convert trace data and header to bytes conforming to SEG-Y spec.

The rows (length) of the headers and traces must match. The headers must be a (num_traces,) shape array and data must be a (num_traces, num_samples) shape array. They can be created via the *create_trace_header_template* and *create_trace_sample_template* methods.

Parameters

- **headers** (*NDArray[Any]*) – Header array.
- **samples** (*NDArray[Any]*) – Data array.

Returns
Bytes containing the encoded traces, ready to write.

Raises

- **AttributeError** – if data dimensions are wrong (not 2D trace,samples).
- **ValueError** – if there is a shape mismatch between headers.
- **ValueError** – if there is a shape mismatch number of samples.

Return type
bytes

property segy_revision: *SegyStandard* | **None**

Revision of the SEG-Y file.

property trace_sample_format: *ScalarType*

Trace sample format of the SEG-Y file.

7.5.3 Configuration

pydantic settings segy.config.**SegyFileSettings**

SEG-Y file parsing settings.


```
{
   "title": "SegyFileSettings",
   "description": "SEG-Y file parsing settings.",
   "type": "object",
   "properties": {
      "binary": {
         "$ref": "#/$defs/SegyBinaryHeaderSettings"
```


(continues on next page)

(continued from previous page)


```
      },
      "endianness": {
         "allOf": [
            {
               "$ref": "#/$defs/Endianness"
            }
         ],
         "default": "big"
      },
      "revision": {
         "anyOf": [
            {
               "type": "integer"
            },
            {
               "type": "number"
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "title": "Revision"
      },
      "storage_options": {
         "title": "Storage Options",
         "type": "object"
      },
      "apply_transforms": {
         "default": true,
         "title": "Apply Transforms",
         "type": "boolean"
      }
   },
   "$defs": {
      "Endianness": {
         "description": "Enumeration class with three possible endianness values.\n\
↪nExamples:\n    >>> endian = Endianness.BIG\n    >>> print(endian.symbol)\n    >",
         "enum": [
            "big",
            "little",
            "native"
         ],
         "title": "Endianness",
         "type": "string"
      },
      "ExtendedTextHeaderSetting": {
         "description": "Configuration for extended textual headers parsing.",
         "properties": {
            "key": {
               "default": "extended_textual_headers",
               "title": "Key",
```


(continues on next page)

(continued from previous page)

```
            "type": "string"
         },
         "value": {
            "anyOf": [
               {
                  "type": "integer"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "title": "Value"
         }
      },
      "title": "ExtendedTextHeaderSetting",
      "type": "object"
   },
   "SampleIntervalSetting": {
      "description": "Configuration for samples interval parsing.",
      "properties": {
         "key": {
            "default": "sample_interval",
            "title": "Key",
            "type": "string"
         },
         "value": {
            "anyOf": [
               {
                  "type": "integer"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "title": "Value"
         }
      },
      "title": "SampleIntervalSetting",
      "type": "object"
   },
   "SamplesPerTraceSetting": {
      "description": "Configuration for samples per trace parsing.",
      "properties": {
         "key": {
            "default": "samples_per_trace",
            "title": "Key",
            "type": "string"
         },
         "value": {
            "anyOf": [
```

(continues on next page)

(continued from previous page)


```
                {
                   "type": "integer"
                },
                {
                   "type": "null"
                }
             ],
             "default": null,
             "title": "Value"
          }
       },
       "title": "SamplesPerTraceSetting",
       "type": "object"
    },
    "SegyBinaryHeaderSettings": {
       "description": "SEG-Y binary header parsing settings.",
       "properties": {
          "samples_per_trace": {
             "allOf": [
                {
                   "$ref": "#/$defs/SamplesPerTraceSetting"
                }
             ],
             "default": {
                "key": "samples_per_trace",
                "value": null
             }
          },
          "sample_interval": {
             "allOf": [
                {
                   "$ref": "#/$defs/SampleIntervalSetting"
                }
             ],
             "default": {
                "key": "sample_interval",
                "value": null
             }
          },
          "extended_text_header": {
             "allOf": [
                {
                   "$ref": "#/$defs/ExtendedTextHeaderSetting"
                }
             ],
             "default": {
                "key": "extended_textual_headers",
                "value": null
             }
          }
       },
       "title": "SegyBinaryHeaderSettings",
```


(continues on next page)

(continued from previous page)

```
         "type": "object"
      }
   }
}
```

field binary: *SegyBinaryHeaderSettings* **[Optional]**

field endianness: *Endianness* **= Endianness.BIG**

field revision: int | float | None **= None**

field storage_options: dict[str, Any] **[Optional]**

field apply_transforms: bool **= True**

pydantic settings segy.config.**SegyBinaryHeaderSettings**

SEG-Y binary header parsing settings.

```
{
   "title": "SegyBinaryHeaderSettings",
   "description": "SEG-Y binary header parsing settings.",
   "type": "object",
   "properties": {
      "samples_per_trace": {
         "allOf": [
            {
               "$ref": "#/$defs/SamplesPerTraceSetting"
            }
         ],
         "default": {
            "key": "samples_per_trace",
            "value": null
         }
      },
      "sample_interval": {
         "allOf": [
            {
               "$ref": "#/$defs/SampleIntervalSetting"
            }
         ],
         "default": {
            "key": "sample_interval",
            "value": null
         }
      },
      "extended_text_header": {
         "allOf": [
            {
               "$ref": "#/$defs/ExtendedTextHeaderSetting"
            }
         ],
         "default": {
            "key": "extended_textual_headers",
```

(continues on next page)

(continued from previous page)


```
            "value": null
         }
      }
   },
   "$defs": {
      "ExtendedTextHeaderSetting": {
         "description": "Configuration for extended textual headers parsing.",
         "properties": {
            "key": {
               "default": "extended_textual_headers",
               "title": "Key",
               "type": "string"
            },
            "value": {
               "anyOf": [
                  {
                     "type": "integer"
                  },
                  {
                     "type": "null"
                  }
               ],
               "default": null,
               "title": "Value"
            }
         },
         "title": "ExtendedTextHeaderSetting",
         "type": "object"
      },
      "SampleIntervalSetting": {
         "description": "Configuration for samples interval parsing.",
         "properties": {
            "key": {
               "default": "sample_interval",
               "title": "Key",
               "type": "string"
            },
            "value": {
               "anyOf": [
                  {
                     "type": "integer"
                  },
                  {
                     "type": "null"
                  }
               ],
               "default": null,
               "title": "Value"
            }
         },
         "title": "SampleIntervalSetting",
         "type": "object"
```


(continues on next page)

(continued from previous page)

```
      },
      "SamplesPerTraceSetting": {
         "description": "Configuration for samples per trace parsing.",
         "properties": {
            "key": {
               "default": "samples_per_trace",
               "title": "Key",
               "type": "string"
            },
            "value": {
               "anyOf": [
                  {
                     "type": "integer"
                  },
                  {
                     "type": "null"
                  }
               ],
               "default": null,
               "title": "Value"
            }
         },
         "title": "SamplesPerTraceSetting",
         "type": "object"
      }
   }
}
```

field samples_per_trace: *SamplesPerTraceSetting* = **SamplesPerTraceSetting(key='samples_per_trace', value=None)**

field sample_interval: *SampleIntervalSetting* = **SampleIntervalSetting(key='sample_interval', value=None)**

field extended_text_header: *ExtendedTextHeaderSetting* = **ExtendedTextHeaderSetting(key='extended_textual_headers', value=None)**

pydantic settings segy.config.**ExtendedTextHeaderSetting**

Configuration for extended textual headers parsing.

```
{
   "title": "ExtendedTextHeaderSetting",
   "description": "Configuration for extended textual headers parsing.",
   "type": "object",
   "properties": {
      "key": {
         "default": "extended_textual_headers",
         "title": "Key",
         "type": "string"
      },
      "value": {
         "anyOf": [
            {
```

(continues on next page)

(continued from previous page)


```
                "type": "integer"
            },
            {
                "type": "null"
            }
        ],
        "default": null,
        "title": "Value"
    }
  }
}
```


field key: `str = 'extended_textual_headers'`

field value: `int | None = None`

pydantic settings segy.config.**SampleIntervalSetting**

Configuration for samples interval parsing.


```
{
   "title": "SampleIntervalSetting",
   "description": "Configuration for samples interval parsing.",
   "type": "object",
   "properties": {
      "key": {
         "default": "sample_interval",
         "title": "Key",
         "type": "string"
      },
      "value": {
         "anyOf": [
            {
               "type": "integer"
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "title": "Value"
      }
   }
}
```


field key: `str = 'sample_interval'`

field value: `int | None = None`

pydantic settings segy.config.**SamplesPerTraceSetting**

Configuration for samples per trace parsing.


```
{
   "title": "SamplesPerTraceSetting",
```


(continues on next page)

(continued from previous page)

```
   "description": "Configuration for samples per trace parsing.",
   "type": "object",
   "properties": {
      "key": {
         "default": "samples_per_trace",
         "title": "Key",
         "type": "string"
      },
      "value": {
         "anyOf": [
            {
               "type": "integer"
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "title": "Value"
      }
   }
}
```

field key: `str = 'samples_per_trace'`

field value: `int | None = None`

7.6 SEG-Y File

Altay Sansal

May 07, 2024

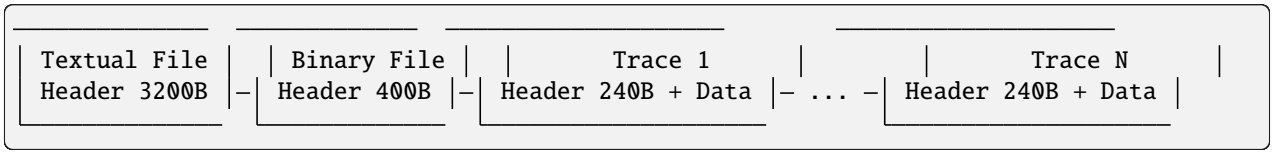
5 min read

7.6.1 SEG-Y Descriptor: A Conceptual Overview

The *SegyDescriptor* is a structured model used to define the structure and content of a SEG-Y file. SEG-Y is a standard file format used in the geophysical industry for recording digital seismic data. In essence, this model serves as a blueprint for what a SEG-Y file should look like.

This class and its components provide a specified and flexible way to work with SEG-Y seismic data files programmatically, from defining the file structure and read/write operations, to customization for specialised use cases.

Conceptually a SEG-Y Revision 0 file looks like this on disk.



Key Components

This descriptor model consists of several important components. Each of these components represents a particular section of a SEG-Y file.

SEGY-Standard

This attribute, *segy_standard*, corresponds to the specific SEG-Y standard that is being used. SEG-Y files can be of different revisions or standards, including custom ones.

It must be set to one of the allowed *SegyStandard* values.

Text File Header

The *text_file_header* stores the information required to parse the textual file header of the SEG-Y file. This includes important metadata that pertains to the seismic data in human-readable format.

Binary File Header

The *binary_file_header* item talks about the binary file header of the SEG-Y file. It is a set of structured and important information about the data in the file, stored in binary format for machines to read and process quickly and efficiently.

Binary headers are defined as *StructuredDataTypeDescriptor*s and are built by specifying header fields in the *StructuredFieldDescriptor* format.

Extended Text Header

The *extended_text_header* is an optional attribute that provides space for extra information that can't be fit within the regular text file header. This extended header can be used for additional human-readable metadata about the data.

Note: Extended text headers are were added in SEG-Y Revision 1.0.

Trace

The *trace* component is a descriptor for both the trace headers and the associated data. Trace headers contain specific information about each individual seismic trace in the dataset, and the trace data contains the actual numerical seismic data.

See also:

TraceDescriptor

The Customize Method

The *customize* method is a way for users to tailor an existing SEG-Y descriptor to meet their specific requirements. It's an optional tool that provides a way to update the various parts of the descriptor including the text header, binary header, extended text header, trace header and trace data. Note that the SEGY standard is always set to custom when using this method.

7.6.2 Reference

pydantic model segy.schema.segy.**SegyDescriptor**

A descriptor class for a SEG-Y file.


```
{
   "title": "SegyDescriptor",
   "description": "A descriptor class for a SEG-Y file.",
   "type": "object",
   "properties": {
      "segyStandard": {
         "anyOf": [
            {
               "$ref": "#/$defs/SegyStandard"
            },
            {
               "type": "null"
            }
         ],
         "description": "SEG-Y Revision / Standard. Can also be custom."
      },
      "textFileHeader": {
         "allOf": [
            {
               "$ref": "#/$defs/TextHeaderDescriptor"
            }
         ],
         "description": "Textual file header descriptor."
      },
      "binaryFileHeader": {
         "allOf": [
            {
               "$ref": "#/$defs/StructuredDataTypeDescriptor"
            }
         ],
         "description": "Binary file header descriptor."
      },
      "extendedTextHeader": {
         "anyOf": [
            {
               "$ref": "#/$defs/TextHeaderDescriptor"
            },
            {
               "type": "null"
            }
```


(continues on next page)

(continued from previous page)


```
        ],
        "default": null,
        "description": "Extended textual header descriptor."
     },
     "trace": {
        "allOf": [
           {
              "$ref": "#/$defs/TraceDescriptor"
           }
        ],
        "description": "Trace header + data descriptor."
     },
     "endianness": {
        "anyOf": [
           {
              "$ref": "#/$defs/Endianness"
           },
           {
              "type": "null"
           }
        ],
        "default": null,
        "description": "Endianness of SEG-Y file."
     }
  },
  "$defs": {
     "Endianness": {
        "description": "Enumeration class with three possible endianness values.\n\
↪nExamples:\n    >>> endian = Endianness.BIG\n    >>> print(endian.symbol)\n    >",
        "enum": [
           "big",
           "little",
           "native"
        ],
        "title": "Endianness",
        "type": "string"
     },
     "ScalarType": {
        "description": "A class representing scalar data types.",
        "enum": [
           "ibm32",
           "int64",
           "int32",
           "int16",
           "int8",
           "uint64",
           "uint32",
           "uint16",
           "uint8",
           "float64",
           "float32",
           "float16"
```


(continues on next page)

(continued from previous page)


```
         ],
         "title": "ScalarType",
         "type": "string"
      },
      "SegyStandard": {
         "description": "Allowed values for SEG-Y standards in SegyDescriptor.",
         "enum": [
            0.0,
            1.0,
            2.0,
            2.1
         ],
         "title": "SegyStandard",
         "type": "numeric"
      },
      "StructuredDataTypeDescriptor": {
         "description": "A class representing a descriptor for a structured data-
↪type.\n\nExamples:\n    Let's build a structured data type from scratch!\n\n    ␣
↪We will define three fields with different names, data-types, and\n    starting␣
↪offsets.\n\n    >>> field1 = StructuredFieldDescriptor(\n    >>>     name=\"foo\",
↪\n    >>>     format=\"int32\",\n    >>>     offset=0,\n    >>> )\n    >>> field2␣
↪= StructuredFieldDescriptor(\n    >>>     name=\"bar\",\n    >>>     format=\
↪"int16\",\n    >>>     offset=4,\n    >>> )\n    >>> field3 =␣
↪StructuredFieldDescriptor(\n    >>>     name=\"fizz\",\n    >>>     format=\
↪"int32\",\n    >>>     offset=16,\n    >>> )\n\n    Note that the fields span the␣
↪following byte ranges:\n\n    * `field1` between bytes `[0, 4)`\n    * `field2`␣
↪between bytes `[4, 6)`\n    * `field3` between bytes `[16, 20)`\n\n    The gap␣
↪between `field2` and `field3` will be padded with `void`. In\n    this case we␣
↪expect to see an item size of 20-bytes (total length of\n    the struct).\n\n    >
↪>> struct_dtype = StructuredDataTypeDescriptor(\n    >>>     fields=[field1,␣
↪field2, field3],\n    >>> )\n\n    Now let's look at its data type:\n\n    >>>␣
↪struct_dtype.dtype\n    dtype({'names': ['foo', 'bar', 'fizz'], 'formats': ['<i4',
↪ '<i2', '<i4'], 'offsets': [0, 4, 16], 'itemsize': 20})\n\n    If we wanted to␣
↪pad the end of the struct (to fit a specific byte range),\n    we would provide␣
↪the item_size in the descriptor. If we set it to 30,\n    this means that we␣
↪padded the struct by 10 bytes at the end.\n\n    >>> struct_dtype =␣
↪StructuredDataTypeDescriptor(\n    >>>     fields=[field1, field2, field3],\n    >
↪>>     item_size=30,\n    >>> )\n\n    Now let's look at its data type:\n\n    >>>
↪ struct_dtype.dtype\n    dtype({'names': ['foo', 'bar', 'fizz'], 'formats': ['<i4
↪', '<i2', '<i4'], 'offsets': [0, 4, 16], 'itemsize': 30})\n\n    To see what's␣
↪going under the hood, we can look at a lower level numpy\n    description of the␣
↪`dtype`. Here we observe all the gaps (void types).\n\n    >>> struct_dtype.dtype.
↪descr\n    [('foo', '<i4'), ('bar', '<i2'), ('', '|V10'), ('fizz', '<i4'), ('',
↪'|V10')]",
         "properties": {
            "description": {
               "anyOf": [
                  {
                     "type": "string"
                  },
                  {
                     "type": "null"
```


(continues on next page)

(continued from previous page)

```
                }
            ],
            "default": null,
            "description": "Description of the field.",
            "title": "Description"
         },
         "fields": {
            "description": "A list of descriptors for a structured data-type.",
            "items": {
               "$ref": "#/$defs/StructuredFieldDescriptor"
            },
            "title": "Fields",
            "type": "array"
         },
         "itemSize": {
            "anyOf": [
               {
                  "type": "integer"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "description": "Expected size of the struct.",
            "title": "Itemsize"
         },
         "offset": {
            "anyOf": [
               {
                  "minimum": 0,
                  "type": "integer"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "description": "Starting byte offset.",
            "title": "Offset"
         },
         "endianness": {
            "anyOf": [
               {
                  "$ref": "#/$defs/Endianness"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "description": "Endianness of structured data type."
```

(continues on next page)

(continued from previous page)


```
            }
         },
         "required": [
            "fields"
         ],
         "title": "StructuredDataTypeDescriptor",
         "type": "object"
      },
      "StructuredFieldDescriptor": {
         "description": "A class representing a descriptor for a structured data-
↪type field.\n\nExamples:\n    A named float at offset 8-bytes:\n\n    >>> data_
↪type = StructuredFieldDescriptor(\n    >>>     name=\"my_var\",\n    >>>     ␣
↪format=\"float32\",\n    >>>     offset=8,\n    >>> )\n\n    The name and offset␣
↪fields will only be used if the structured\n    field is used within the context␣
↪of a :class:`StructuredDataTypeDescriptor`.\n\n    >>> data_type.name\n    my_var\
↪n    >>> data_type.offset\n    8\n\n    The `dtype` property is inherited from␣
↪:class:`DataTypeDescriptor`.\n\n    >>> data_type.dtype\n    dtype('float32')",
         "properties": {
            "description": {
               "anyOf": [
                  {
                     "type": "string"
                  },
                  {
                     "type": "null"
                  }
               ],
               "default": null,
               "description": "Description of the field.",
               "title": "Description"
            },
            "format": {
               "allOf": [
                  {
                     "$ref": "#/$defs/ScalarType"
                  }
               ],
               "description": "The data type of the field."
            },
            "name": {
               "description": "The short name of the field.",
               "title": "Name",
               "type": "string"
            },
            "offset": {
               "description": "Starting byte offset.",
               "minimum": 0,
               "title": "Offset",
               "type": "integer"
            }
         },
         "required": [
```


(continues on next page)

(continued from previous page)


```
            "format",
            "name",
            "offset"
         ],
         "title": "StructuredFieldDescriptor",
         "type": "object"
      },
      "TextHeaderDescriptor": {
         "description": "A descriptor class for SEG-Y textual headers.",
         "properties": {
            "description": {
               "anyOf": [
                  {
                     "type": "string"
                  },
                  {
                     "type": "null"
                  }
               ],
               "default": null,
               "description": "Description of the field.",
               "title": "Description"
            },
            "rows": {
               "description": "Number of rows in text header.",
               "title": "Rows",
               "type": "integer"
            },
            "cols": {
               "description": "Number of columns in text header.",
               "title": "Cols",
               "type": "integer"
            },
            "encoding": {
               "allOf": [
                  {
                     "$ref": "#/$defs/TextHeaderEncoding"
                  }
               ],
               "description": "String encoding."
            },
            "format": {
               "allOf": [
                  {
                     "$ref": "#/$defs/ScalarType"
                  }
               ],
               "description": "Type of string."
            },
            "offset": {
               "anyOf": [
                  {
```


(continues on next page)

(continued from previous page)


```
                    "minimum": 0,
                    "type": "integer"
                },
                {
                    "type": "null"
                }
            ],
            "default": null,
            "description": "Starting byte offset.",
            "title": "Offset"
        }
    },
    "required": [
        "rows",
        "cols",
        "encoding",
        "format"
    ],
    "title": "TextHeaderDescriptor",
    "type": "object"
},
"TextHeaderEncoding": {
    "description": "Supported textual header encodings.",
    "enum": [
        "ascii",
        "ebcdic"
    ],
    "title": "TextHeaderEncoding",
    "type": "string"
},
"TraceDescriptor": {
    "description": "A descriptor class for a Trace (Header + Data).",
    "properties": {
        "description": {
            "anyOf": [
                {
                    "type": "string"
                },
                {
                    "type": "null"
                }
            ],
            "default": null,
            "description": "Description of the field.",
            "title": "Description"
        },
        "headerDescriptor": {
            "allOf": [
                {
                    "$ref": "#/$defs/StructuredDataTypeDescriptor"
                }
            ],
```


(continues on next page)

(continued from previous page)

```
            "description": "Trace header descriptor."
         },
         "extendedHeaderDescriptor": {
            "anyOf": [
               {
                  "$ref": "#/$defs/StructuredDataTypeDescriptor"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "description": "Extended trace header descriptor."
         },
         "sampleDescriptor": {
            "allOf": [
               {
                  "$ref": "#/$defs/TraceSampleDescriptor"
               }
            ],
            "description": "Trace data descriptor."
         },
         "offset": {
            "anyOf": [
               {
                  "type": "integer"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "description": "Starting offset of the trace.",
            "title": "Offset"
         },
         "endianness": {
            "anyOf": [
               {
                  "$ref": "#/$defs/Endianness"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "description": "Endianness of traces and headers."
         }
      },
      "required": [
         "headerDescriptor",
         "sampleDescriptor"
      ],
```

(continues on next page)

(continued from previous page)

```
         "title": "TraceDescriptor",
         "type": "object"
      },
      "TraceSampleDescriptor": {
         "description": "A descriptor class for a Trace Samples.",
         "properties": {
            "description": {
               "anyOf": [
                  {
                     "type": "string"
                  },
                  {
                     "type": "null"
                  }
               ],
               "default": null,
               "description": "Description of the field.",
               "title": "Description"
            },
            "format": {
               "allOf": [
                  {
                     "$ref": "#/$defs/ScalarType"
                  }
               ],
               "description": "Format of trace samples."
            },
            "samples": {
               "anyOf": [
                  {
                     "type": "integer"
                  },
                  {
                     "type": "null"
                  }
               ],
               "default": null,
               "description": "Number of samples in trace. It can be variable, then␣
↪it must be read from each trace header.",
               "title": "Samples"
            }
         },
         "required": [
            "format"
         ],
         "title": "TraceSampleDescriptor",
         "type": "object"
      }
   },
   "required": [
      "segyStandard",
      "textFileHeader",
```

(continues on next page)

(continued from previous page)

```
      "binaryFileHeader",
      "trace"
   ]
}
```

field segyStandard: *SegyStandard* | None **[Required]**

SEG-Y Revision / Standard. Can also be custom.

field textFileHeader: *TextHeaderDescriptor* **[Required]**

Textual file header descriptor.

field binaryFileHeader: *StructuredDataTypeDescriptor* **[Required]**

Binary file header descriptor.

field extendedTextHeader: *TextHeaderDescriptor* | None = None

Extended textual header descriptor.

field trace: *TraceDescriptor* **[Required]**

Trace header + data descriptor.

field endianness: *Endianness* | None = None

Endianness of SEG-Y file.

customize(*text_header_spec=None*, *binary_header_fields=None*, *extended_text_spec=None*, *trace_header_fields=None*, *trace_data_spec=None*)

Customize an existing SEG-Y descriptor.

Parameters

- **text_header_spec** (TextHeaderDescriptor | *None*) – New text header specification.
- **binary_header_fields** (*list*[StructuredFieldDescriptor] | *None*) – List of custom binary header fields.
- **extended_text_spec** (TextHeaderDescriptor | *None*) – New extended text header specification.
- **trace_header_fields** (*list*[StructuredFieldDescriptor] | *None*) – List of custom trace header fields.
- **trace_data_spec** (*TraceSampleDescriptor* | *None*) – New trace data specification.
- **self** (SegyDescriptor)

Returns

A modified SEG-Y descriptor with "custom" segy standard.

Return type

SegyDescriptor

pydantic model segy.schema.header.**TextHeaderDescriptor**

A descriptor class for SEG-Y textual headers.

```
{
   "title": "TextHeaderDescriptor",
   "description": "A descriptor class for SEG-Y textual headers.",
   "type": "object",
```

(continues on next page)

(continued from previous page)


```
  "properties": {
     "description": {
        "anyOf": [
           {
              "type": "string"
           },
           {
              "type": "null"
           }
        ],
        "default": null,
        "description": "Description of the field.",
        "title": "Description"
     },
     "rows": {
        "description": "Number of rows in text header.",
        "title": "Rows",
        "type": "integer"
     },
     "cols": {
        "description": "Number of columns in text header.",
        "title": "Cols",
        "type": "integer"
     },
     "encoding": {
        "allOf": [
           {
              "$ref": "#/$defs/TextHeaderEncoding"
           }
        ],
        "description": "String encoding."
     },
     "format": {
        "allOf": [
           {
              "$ref": "#/$defs/ScalarType"
           }
        ],
        "description": "Type of string."
     },
     "offset": {
        "anyOf": [
           {
              "minimum": 0,
              "type": "integer"
           },
           {
              "type": "null"
           }
        ],
        "default": null,
        "description": "Starting byte offset.",
```


(continues on next page)

(continued from previous page)


```
            "title": "Offset"
         }
      },
      "$defs": {
         "ScalarType": {
            "description": "A class representing scalar data types.",
            "enum": [
               "ibm32",
               "int64",
               "int32",
               "int16",
               "int8",
               "uint64",
               "uint32",
               "uint16",
               "uint8",
               "float64",
               "float32",
               "float16"
            ],
            "title": "ScalarType",
            "type": "string"
         },
         "TextHeaderEncoding": {
            "description": "Supported textual header encodings.",
            "enum": [
               "ascii",
               "ebcdic"
            ],
            "title": "TextHeaderEncoding",
            "type": "string"
         }
      },
      "required": [
         "rows",
         "cols",
         "encoding",
         "format"
      ]
}
```


field rows: *int* **[Required]**

Number of rows in text header.

field cols: *int* **[Required]**

Number of columns in text header.

field encoding: **TextHeaderEncoding [Required]**

String encoding.

field format: *ScalarType* **[Required]**

Type of string.

field offset: int | None = None

Starting byte offset.

Constraints

- **ge** = 0

property dtype: dtype[Any]

Get numpy dtype.

property itemsize: int

Number of bytes for the data type.

field description: str | None = None

Description of the field.

class segy.schema.segy.**SegyStandard**

Allowed values for SEG-Y standards in SegyDescriptor.

REV0 = 0.0

REV1 = 1.0

REV2 = 2.0

REV21 = 2.1

pydantic model segy.schema.segy.**SegyInfo**

Concise and useful information about SEG-Y files.


```
{
   "title": "SegyInfo",
   "description": "Concise and useful information about SEG-Y files.",
   "type": "object",
   "properties": {
      "uri": {
         "description": "URI of the SEG-Y file.",
         "title": "Uri",
         "type": "string"
      },
      "segyStandard": {
         "anyOf": [
            {
               "$ref": "#/$defs/SegyStandard"
            },
            {
               "type": "null"
            }
         ],
         "description": "SEG-Y Revision / Standard. Can also be custom."
      },
      "numTraces": {
         "description": "Number of traces.",
         "title": "Numtraces",
         "type": "integer"
      },
      "samplesPerTrace": {
```


(continues on next page)

(continued from previous page)


```
            "description": "Trace length in number of samples.",
            "title": "Samplespertrace",
            "type": "integer"
         },
         "sampleInterval": {
            "anyOf": [
               {
                  "type": "integer"
               },
               {
                  "type": "number"
               }
            ],
            "description": "Sampling rate from binary header.",
            "title": "Sampleinterval"
         },
         "fileSize": {
            "description": "File size in bytes.",
            "title": "Filesize",
            "type": "integer"
         }
      },
      "$defs": {
         "SegyStandard": {
            "description": "Allowed values for SEG-Y standards in SegyDescriptor.",
            "enum": [
               0.0,
               1.0,
               2.0,
               2.1
            ],
            "title": "SegyStandard",
            "type": "numeric"
         }
      },
      "required": [
         "uri",
         "segyStandard",
         "numTraces",
         "samplesPerTrace",
         "sampleInterval",
         "fileSize"
      ]
   }
```


field uri: `str` **[Required]**

URI of the SEG-Y file.

field segyStandard: *SegyStandard* | `None` **[Required]**

SEG-Y Revision / Standard. Can also be custom.

field numTraces: `int` **[Required]**

Number of traces.

field samplesPerTrace: int [Required]

Trace length in number of samples.

field sampleInterval: int | float [Required]

Sampling rate from binary header.

field fileSize: int [Required]

File size in bytes.

7.7 Traces

Altay Sansal

May 07, 2024

4 min read

7.7.1 Defining a Trace

The *TraceDescriptor* is a way to define the structure of a seismic trace as stored in SEG-Y files. It is composed of *Trace Header Descriptor* and *Trace Data Descriptor*. This information is combined using the *TraceDescriptor*.

The *TraceDescriptor* has fields for trace header, optional extended trace header, and trace data definitions. We also provide an optional *offset* field to define the beginning byte-location of the traces within a binary file. Most of the time this field gets populated automatically.

A custom trace descriptor can be built programmatically following a simple workflow. The same descriptor can be built from `JSON` as well. Navigate to *JSON Trace Descriptor* below for that.

Trace Header Descriptor

Trace headers are defined using *StructuredDataTypeDescriptor*. Each header field is a *StructuredFieldDescriptor*. We have an example workflow here. You can see more examples in the *Data Types* documentation.

We first do the required imports and then define header fields. By default, endianness is `big`, so we don't have to declare it.

```
from segy.schema.data_type import StructuredFieldDescriptor

trace_header_fields = [
    StructuredFieldDescriptor(
        name="inline",
        offset=188,
        format="int32",
    ),
    StructuredFieldDescriptor(
        name="crossline",
        offset=192,
        format="int32",
    ),
]
```

Then we create *StructuredDataTypeDescriptor* for trace headers. We know trace headers must be 240-bytes so we declare it. This will ensure we read/write with correct padding.

```
from segy.schema.data_type import StructuredDataTypeDescriptor

trace_header_descriptor = StructuredDataTypeDescriptor(
    fields=trace_header_fields,
    item_size=240,
)
```

Trace Data Descriptor

Trace data is described using *TraceDataDescriptor*. The data is mainly explained by its data type (*endianness* and *format*), and number of *samples*.

Continuing our previous example, we build the data descriptor. We assume that samples are encoded in 'ibm32' format and and they are big endian (again, default).

```

from segy.schema.trace import TraceDataDescriptor

trace_data_descriptor = TraceDataDescriptor(
    format="ibm32",
    samples=360
)
```

Trace Descriptor

Finally, since we have all components, we can create a descriptor for of a trace.

```
from segy.schema.trace import TraceDescriptor

trace_descriptor = TraceDescriptor(
    header_descriptor=trace_header_descriptor,
    data_descriptor=trace_data_descriptor,
    offset=3600  # just an example of possible offset
)
```

If we look at the Numpy data type of the trace, we can see how it will be decoded from raw bytes:

```
>>> trace_descriptor.dtype
dtype([('header', {'names': ['inline', 'crossline'], 'formats': ['>i4', '>i4'], 'offsets
↪': [188, 192], 'itemsize': 240}), ('data', '>u4', (360,))])
```

7.7.2 JSON Trace Descriptor

We can define the exact same trace descriptor above using `JSON`. This can either be defined as a `string` or can be read from a file. Both will work. Let's write the `JSON`.

```
{
  "headerDescriptor": {
    "fields": [
      {
```

(continues on next page)

(continued from previous page)

```
        "format": "int32",
        "name": "inline",
        "offset": 188
      },
      {
        "format": "int32",
        "name": "crossline",
        "offset": 192
      }
    ],
    "itemSize": 240
  },
  "dataDescriptor": {
    "format": "ibm32",
    "samples": 360
  },
  "offset": 3600
}
```

Then if we have our `JSON` as a `string` in the variable `json_str`, we can generate the same descriptor, with validation of all fields. If there are any errors in the `JSON`, there will be a validation error raised.

```
>>> trace_descriptor_from_json = TraceDescriptor.model_validate_json(json_str)
>>> trace_descriptor_from_json == trace_descriptor
True
```

7.7.3 Reference

pydantic model segy.schema.trace.**TraceDescriptor**

A descriptor class for a Trace (Header + Data).

```
{
   "title": "TraceDescriptor",
   "description": "A descriptor class for a Trace (Header + Data).",
   "type": "object",
   "properties": {
      "description": {
         "anyOf": [
            {
               "type": "string"
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "description": "Description of the field.",
         "title": "Description"
      },
      "headerDescriptor": {
         "allOf": [
```

(continues on next page)

(continued from previous page)

```
            {
               "$ref": "#/$defs/StructuredDataTypeDescriptor"
            }
         ],
         "description": "Trace header descriptor."
      },
      "extendedHeaderDescriptor": {
         "anyOf": [
            {
               "$ref": "#/$defs/StructuredDataTypeDescriptor"
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "description": "Extended trace header descriptor."
      },
      "sampleDescriptor": {
         "allOf": [
            {
               "$ref": "#/$defs/TraceSampleDescriptor"
            }
         ],
         "description": "Trace data descriptor."
      },
      "offset": {
         "anyOf": [
            {
               "type": "integer"
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "description": "Starting offset of the trace.",
         "title": "Offset"
      },
      "endianness": {
         "anyOf": [
            {
               "$ref": "#/$defs/Endianness"
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "description": "Endianness of traces and headers."
      }
   },
```

(continues on next page)

(continued from previous page)


```
  "$defs": {
     "Endianness": {
        "description": "Enumeration class with three possible endianness values.\n\
↪nExamples:\n    >>> endian = Endianness.BIG\n    >>> print(endian.symbol)\n    >",
        "enum": [
           "big",
           "little",
           "native"
        ],
        "title": "Endianness",
        "type": "string"
     },
     "ScalarType": {
        "description": "A class representing scalar data types.",
        "enum": [
           "ibm32",
           "int64",
           "int32",
           "int16",
           "int8",
           "uint64",
           "uint32",
           "uint16",
           "uint8",
           "float64",
           "float32",
           "float16"
        ],
        "title": "ScalarType",
        "type": "string"
     },
     "StructuredDataTypeDescriptor": {
        "description": "A class representing a descriptor for a structured data-
↪type.\n\nExamples:\n    Let's build a structured data type from scratch!\n\n    ␣
↪We will define three fields with different names, data-types, and\n    starting␣
↪offsets.\n\n    >>> field1 = StructuredFieldDescriptor(\n    >>>     name=\"foo\",
↪\n    >>>     format=\"int32\",\n    >>>     offset=0,\n    >>> )\n    >>> field2␣
↪= StructuredFieldDescriptor(\n    >>>     name=\"bar\",\n    >>>     format=\
↪"int16\",\n    >>>     offset=4,\n    >>> )\n    >>> field3 =␣
↪StructuredFieldDescriptor(\n    >>>     name=\"fizz\",\n    >>>     format=\
↪"int32\",\n    >>>     offset=16,\n    >>> )\n\n    Note that the fields span the␣
↪following byte ranges:\n\n    * `field1` between bytes `[0, 4)`\n    * `field2`␣
↪between bytes `[4, 6)`\n    * `field3` between bytes `[16, 20)`\n\n    The gap␣
↪between `field2` and `field3` will be padded with `void`. In\n    this case we␣
↪expect to see an item size of 20-bytes (total length of\n    the struct).\n\n    >
↪>> struct_dtype = StructuredDataTypeDescriptor(\n    >>>     fields=[field1,␣
↪field2, field3],\n    >>> )\n\n    Now let's look at its data type:\n\n    >>>␣
↪struct_dtype.dtype\n    dtype({'names': ['foo', 'bar', 'fizz'], 'formats': ['<i4',
↪ '<i2', '<i4'], 'offsets': [0, 4, 16], 'itemsize': 20})\n\n    If we wanted to␣
↪pad the end of the struct (to fit a specific byte range),\n    we would provide␣
↪the item_size in the descriptor. If we set it to 30,\n    this means that we␣
↪padded the struct by 10 bytes at the end.\n\n    >>> struct_dtype =␣
```


(continues on next page)

(continued from previous page)

```
↪StructuredDataTypeDescriptor(\n    >>>     fields=[field1, field2, field3],\n    >
↪>>     item_size=30,\n    >>> )\n\n    Now let's look at its data type:\n\n    >>>
↪ struct_dtype.dtype\n    dtype({'names': ['foo', 'bar', 'fizz'], 'formats': ['<i4
↪', '<i2', '<i4'], 'offsets': [0, 4, 16], 'itemsize': 30})\n\n    To see what's␣
↪going under the hood, we can look at a lower level numpy\n    description of the␣
↪`dtype`. Here we observe all the gaps (void types).\n\n    >>> struct_dtype.dtype.
↪descr\n    [('foo', '<i4'), ('bar', '<i2'), ('', '|V10'), ('fizz', '<i4'), ('',
↪'|V10')]",
        "properties": {
           "description": {
              "anyOf": [
                 {
                    "type": "string"
                 },
                 {
                    "type": "null"
                 }
              ],
              "default": null,
              "description": "Description of the field.",
              "title": "Description"
           },
           "fields": {
              "description": "A list of descriptors for a structured data-type.",
              "items": {
                 "$ref": "#/$defs/StructuredFieldDescriptor"
              },
              "title": "Fields",
              "type": "array"
           },
           "itemSize": {
              "anyOf": [
                 {
                    "type": "integer"
                 },
                 {
                    "type": "null"
                 }
              ],
              "default": null,
              "description": "Expected size of the struct.",
              "title": "Itemsize"
           },
           "offset": {
              "anyOf": [
                 {
                    "minimum": 0,
                    "type": "integer"
                 },
                 {
                    "type": "null"
                 }
```

(continues on next page)

(continued from previous page)


```
            ],
            "default": null,
            "description": "Starting byte offset.",
            "title": "Offset"
         },
         "endianness": {
            "anyOf": [
               {
                  "$ref": "#/$defs/Endianness"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "description": "Endianness of structured data type."
         }
      },
      "required": [
         "fields"
      ],
      "title": "StructuredDataTypeDescriptor",
      "type": "object"
   },
   "StructuredFieldDescriptor": {
      "description": "A class representing a descriptor for a structured data-
↪type field.\n\nExamples:\n    A named float at offset 8-bytes:\n\n    >>> data_
↪type = StructuredFieldDescriptor(\n    >>>     name=\"my_var\",\n    >>>     ␣
↪format=\"float32\",\n    >>>     offset=8,\n    >>> )\n\n    The name and offset␣
↪fields will only be used if the structured\n    field is used within the context␣
↪of a :class:`StructuredDataTypeDescriptor`.\n\n    >>> data_type.name\n    my_var\
↪n    >>> data_type.offset\n    8\n\n    The `dtype` property is inherited from␣
↪:class:`DataTypeDescriptor`.\n\n    >>> data_type.dtype\n    dtype('float32')",
      "properties": {
         "description": {
            "anyOf": [
               {
                  "type": "string"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "description": "Description of the field.",
            "title": "Description"
         },
         "format": {
            "allOf": [
               {
                  "$ref": "#/$defs/ScalarType"
               }
```


(continues on next page)

(continued from previous page)


```
            ],
            "description": "The data type of the field."
         },
         "name": {
            "description": "The short name of the field.",
            "title": "Name",
            "type": "string"
         },
         "offset": {
            "description": "Starting byte offset.",
            "minimum": 0,
            "title": "Offset",
            "type": "integer"
         }
      },
      "required": [
         "format",
         "name",
         "offset"
      ],
      "title": "StructuredFieldDescriptor",
      "type": "object"
   },
   "TraceSampleDescriptor": {
      "description": "A descriptor class for a Trace Samples.",
      "properties": {
         "description": {
            "anyOf": [
               {
                  "type": "string"
               },
               {
                  "type": "null"
               }
            ],
            "default": null,
            "description": "Description of the field.",
            "title": "Description"
         },
         "format": {
            "allOf": [
               {
                  "$ref": "#/$defs/ScalarType"
               }
            ],
            "description": "Format of trace samples."
         },
         "samples": {
            "anyOf": [
               {
                  "type": "integer"
               },
```


(continues on next page)

(continued from previous page)

```
                {
                   "type": "null"
                }
             ],
             "default": null,
             "description": "Number of samples in trace. It can be variable, then␣
↪it must be read from each trace header.",
             "title": "Samples"
          }
       },
       "required": [
          "format"
       ],
       "title": "TraceSampleDescriptor",
       "type": "object"
    }
 },
 "required": [
    "headerDescriptor",
    "sampleDescriptor"
 ]
}
```

field headerDescriptor: *StructuredDataTypeDescriptor* **[Required]**

Trace header descriptor.

field extendedHeaderDescriptor: *StructuredDataTypeDescriptor* | None = None

Extended trace header descriptor.

field sampleDescriptor: TraceSampleDescriptor **[Required]**

Trace data descriptor.

field offset: int | None = None

Starting offset of the trace.

field endianness: *Endianness* | None = None

Endianness of traces and headers.

property dtype: dtype[Any]

Get numpy dtype.

7.8 Data Types

Altay Sansal

May 07, 2024

0 min read

7.8.1 Intro

ScalarType	A class representing scalar data types.
StructuredDataTypeDescriptor	A class representing a descriptor for a structured data-type.
StructuredFieldDescriptor	A class representing a descriptor for a structured data-type field.
Endianness	Enumeration class with three possible endianness values.

class segy.schema.data_type.**ScalarType**

A class representing scalar data types.

IBM32 = 'ibm32'

INT64 = 'int64'

INT32 = 'int32'

INT16 = 'int16'

INT8 = 'int8'

UINT64 = 'uint64'

UINT32 = 'uint32'

UINT16 = 'uint16'

UINT8 = 'uint8'

FLOAT64 = 'float64'

FLOAT32 = 'float32'

FLOAT16 = 'float16'

property char: str

Returns the numpy character code for a given data type string.

pydantic model segy.schema.data_type.**StructuredDataTypeDescriptor**

A class representing a descriptor for a structured data-type.

Examples

Let's build a structured data type from scratch!

We will define three fields with different names, data-types, and starting offsets.

```
>>> field1 = StructuredFieldDescriptor(
>>>     name="foo",
>>>     format="int32",
>>>     offset=0,
>>> )
>>> field2 = StructuredFieldDescriptor(
```

(continues on next page)

(continued from previous page)

```
>>>     name="bar",
>>>     format="int16",
>>>     offset=4,
>>> )
>>> field3 = StructuredFieldDescriptor(
>>>     name="fizz",
>>>     format="int32",
>>>     offset=16,
>>> )
```

Note that the fields span the following byte ranges:

- *field1* between bytes *[0, 4)*
- *field2* between bytes *[4, 6)*
- *field3* between bytes *[16, 20)*

The gap between *field2* and *field3* will be padded with *void*. In this case we expect to see an item size of 20-bytes (total length of the struct).

```
>>> struct_dtype = StructuredDataTypeDescriptor(
>>>     fields=[field1, field2, field3],
>>> )
```

Now let's look at its data type:

```
>>> struct_dtype.dtype
dtype({'names': ['foo', 'bar', 'fizz'], 'formats': ['<i4', '<i2', '<i4'], 'offsets
↪': [0, 4, 16], 'itemsize': 20})
```

If we wanted to pad the end of the struct (to fit a specific byte range), we would provide the item_size in the descriptor. If we set it to 30, this means that we padded the struct by 10 bytes at the end.

```
>>> struct_dtype = StructuredDataTypeDescriptor(
>>>     fields=[field1, field2, field3],
>>>     item_size=30,
>>> )
```

Now let's look at its data type:

```
>>> struct_dtype.dtype
dtype({'names': ['foo', 'bar', 'fizz'], 'formats': ['<i4', '<i2', '<i4'], 'offsets
↪': [0, 4, 16], 'itemsize': 30})
```

To see what's going under the hood, we can look at a lower level numpy description of the *dtype*. Here we observe all the gaps (void types).

```
>>> struct_dtype.dtype.descr
[('foo', '<i4'), ('bar', '<i2'), ('', '|V10'), ('fizz', '<i4'), ('', '|V10')]
```

```
{
   "title": "StructuredDataTypeDescriptor",
   "description": "A class representing a descriptor for a structured data-type.\n\
↪nExamples:\n    Let's build a structured data type from scratch!\n\n    We will␣
```

(continues on next page)

(continued from previous page)

```
↪define three fields with different names, data-types, and\n    starting offsets.\
↪n\n    >>> field1 = StructuredFieldDescriptor(\n    >>>     name=\"foo\",\n    >>>
↪     format=\"int32\",\n    >>>     offset=0,\n    >>> )\n    >>> field2 =␣
↪StructuredFieldDescriptor(\n    >>>     name=\"bar\",\n    >>>     format=\"int16\
↪",\n    >>>     offset=4,\n    >>> )\n    >>> field3 = StructuredFieldDescriptor(\
↪n    >>>     name=\"fizz\",\n    >>>     format=\"int32\",\n    >>>     offset=16,
↪\n    >>> )\n\n    Note that the fields span the following byte ranges:\n\n    *␣
↪`field1` between bytes `[0, 4)`\n    * `field2` between bytes `[4, 6)`\n    *␣
↪`field3` between bytes `[16, 20)`\n\n    The gap between `field2` and `field3`␣
↪will be padded with `void`. In\n    this case we expect to see an item size of 20-
↪bytes (total length of\n    the struct).\n\n    >>> struct_dtype =␣
↪StructuredDataTypeDescriptor(\n    >>>     fields=[field1, field2, field3],\n    >
↪>> )\n\n    Now let's look at its data type:\n\n    >>> struct_dtype.dtype\n    ␣
↪dtype({'names': ['foo', 'bar', 'fizz'], 'formats': ['<i4', '<i2', '<i4'], 'offsets
↪': [0, 4, 16], 'itemsize': 20})\n\n    If we wanted to pad the end of the struct␣
↪(to fit a specific byte range),\n    we would provide the item_size in the␣
↪descriptor. If we set it to 30,\n    this means that we padded the struct by 10␣
↪bytes at the end.\n\n    >>> struct_dtype = StructuredDataTypeDescriptor(\n    >>>
↪     fields=[field1, field2, field3],\n    >>>     item_size=30,\n    >>> )\n\n  ␣
↪ Now let's look at its data type:\n\n    >>> struct_dtype.dtype\n    dtype({'names
↪': ['foo', 'bar', 'fizz'], 'formats': ['<i4', '<i2', '<i4'], 'offsets': [0, 4,␣
↪16], 'itemsize': 30})\n\n    To see what's going under the hood, we can look at a␣
↪lower level numpy\n    description of the `dtype`. Here we observe all the gaps␣
↪(void types).\n\n    >>> struct_dtype.dtype.descr\n    [('foo', '<i4'), ('bar', '
↪<i2'), ('', '|V10'), ('fizz', '<i4'), ('', '|V10')]",
  "type": "object",
  "properties": {
     "description": {
        "anyOf": [
           {
              "type": "string"
           },
           {
              "type": "null"
           }
        ],
        "default": null,
        "description": "Description of the field.",
        "title": "Description"
     },
     "fields": {
        "description": "A list of descriptors for a structured data-type.",
        "items": {
           "$ref": "#/$defs/StructuredFieldDescriptor"
        },
        "title": "Fields",
        "type": "array"
     },
     "itemSize": {
        "anyOf": [
           {
              "type": "integer"
```

(continues on next page)

(continued from previous page)


```
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "description": "Expected size of the struct.",
         "title": "Itemsize"
      },
      "offset": {
         "anyOf": [
            {
               "minimum": 0,
               "type": "integer"
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "description": "Starting byte offset.",
         "title": "Offset"
      },
      "endianness": {
         "anyOf": [
            {
               "$ref": "#/$defs/Endianness"
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "description": "Endianness of structured data type."
      }
   },
   "$defs": {
      "Endianness": {
         "description": "Enumeration class with three possible endianness values.\n\
↪nExamples:\n    >>> endian = Endianness.BIG\n    >>> print(endian.symbol)\n    >",
         "enum": [
            "big",
            "little",
            "native"
         ],
         "title": "Endianness",
         "type": "string"
      },
      "ScalarType": {
         "description": "A class representing scalar data types.",
         "enum": [
            "ibm32",
```


(continues on next page)

(continued from previous page)


```
            "int64",
            "int32",
            "int16",
            "int8",
            "uint64",
            "uint32",
            "uint16",
            "uint8",
            "float64",
            "float32",
            "float16"
         ],
         "title": "ScalarType",
         "type": "string"
      },
      "StructuredFieldDescriptor": {
         "description": "A class representing a descriptor for a structured data-
↪type field.\n\nExamples:\n    A named float at offset 8-bytes:\n\n    >>> data_
↪type = StructuredFieldDescriptor(\n    >>>     name=\"my_var\",\n    >>>     ␣
↪format=\"float32\",\n    >>>     offset=8,\n    >>> )\n\n    The name and offset␣
↪fields will only be used if the structured\n    field is used within the context␣
↪of a :class:`StructuredDataTypeDescriptor`.\n\n    >>> data_type.name\n    my_var\
↪n    >>> data_type.offset\n    8\n\n    The `dtype` property is inherited from␣
↪:class:`DataTypeDescriptor`.\n\n    >>> data_type.dtype\n    dtype('float32')",
         "properties": {
            "description": {
               "anyOf": [
                  {
                     "type": "string"
                  },
                  {
                     "type": "null"
                  }
               ],
               "default": null,
               "description": "Description of the field.",
               "title": "Description"
            },
            "format": {
               "allOf": [
                  {
                     "$ref": "#/$defs/ScalarType"
                  }
               ],
               "description": "The data type of the field."
            },
            "name": {
               "description": "The short name of the field.",
               "title": "Name",
               "type": "string"
            },
            "offset": {
```


(continues on next page)

(continued from previous page)

```
                "description": "Starting byte offset.",
                "minimum": 0,
                "title": "Offset",
                "type": "integer"
             }
          },
          "required": [
             "format",
             "name",
             "offset"
          ],
          "title": "StructuredFieldDescriptor",
          "type": "object"
       }
    },
    "required": [
       "fields"
    ]
}
```

field fields: `list[StructuredFieldDescriptor]` **[Required]**

A list of descriptors for a structured data-type.

field itemSize: `int | None` **= None**

Expected size of the struct.

field offset: `int | None` **= None**

Starting byte offset.

Constraints

- **ge** = 0

field endianness: `Endianness | None` **= None**

Endianness of structured data type.

property dtype: `dtype[Any]`

Converts the names, data types, and offsets of the object into a NumPy dtype.

property itemsize: `int`

Number of bytes for the data type.

field description: `str | None` **= None**

Description of the field.

pydantic model `segy.schema.data_type.`**StructuredFieldDescriptor**

A class representing a descriptor for a structured data-type field.

Examples

A named float at offset 8-bytes:

```
>>> data_type = StructuredFieldDescriptor(
>>>     name="my_var",
>>>     format="float32",
>>>     offset=8,
>>> )
```

The name and offset fields will only be used if the structured field is used within the context of a *StructuredDataTypeDescriptor*.

```
>>> data_type.name
my_var
>>> data_type.offset
8
```

The *dtype* property is inherited from `DataTypeDescriptor`.

```
>>> data_type.dtype
dtype('float32')
```


```
{
   "title": "StructuredFieldDescriptor",
   "description": "A class representing a descriptor for a structured data-type␣
↪field.\n\nExamples:\n    A named float at offset 8-bytes:\n\n    >>> data_type =␣
↪StructuredFieldDescriptor(\n    >>>     name=\"my_var\",\n    >>>     format=\
↪"float32\",\n    >>>     offset=8,\n    >>> )\n\n    The name and offset fields␣
↪will only be used if the structured\n    field is used within the context of a␣
↪:class:`StructuredDataTypeDescriptor`.\n\n    >>> data_type.name\n    my_var\n  ␣
↪>>> data_type.offset\n    8\n\n    The `dtype` property is inherited from␣
↪:class:`DataTypeDescriptor`.\n\n    >>> data_type.dtype\n    dtype('float32')",
   "type": "object",
   "properties": {
      "description": {
         "anyOf": [
            {
               "type": "string"
            },
            {
               "type": "null"
            }
         ],
         "default": null,
         "description": "Description of the field.",
         "title": "Description"
      },
      "format": {
         "allOf": [
            {
               "$ref": "#/$defs/ScalarType"
            }
         ],
```


(continues on next page)

(continued from previous page)


```
            "description": "The data type of the field."
         },
         "name": {
            "description": "The short name of the field.",
            "title": "Name",
            "type": "string"
         },
         "offset": {
            "description": "Starting byte offset.",
            "minimum": 0,
            "title": "Offset",
            "type": "integer"
         }
      },
      "$defs": {
         "ScalarType": {
            "description": "A class representing scalar data types.",
            "enum": [
               "ibm32",
               "int64",
               "int32",
               "int16",
               "int8",
               "uint64",
               "uint32",
               "uint16",
               "uint8",
               "float64",
               "float32",
               "float16"
            ],
            "title": "ScalarType",
            "type": "string"
         }
      },
      "required": [
         "format",
         "name",
         "offset"
      ]
}
```


field name: `str` **[Required]**

The short name of the field.

field offset: `int` **[Required]**

Starting byte offset.

Constraints

- **ge** = 0

property dtype: `dtype[Any]`

Converts the byte order and data type of the object into a NumPy dtype.

property itemsize: `int`

Number of bytes for the data type.

field format: *ScalarType* **[Required]**

The data type of the field.

field description: `str | None = None`

Description of the field.

class segy.schema.data_type.**Endianness**

Enumeration class with three possible endianness values.

Examples

```
>>> endian = Endianness.BIG
>>> print(endian.symbol)
>
```

BIG = 'big'

LITTLE = 'little'

NATIVE = 'native'

property symbol: `Literal['<', '>', '=']`

Get the numpy symbol for the endianness from mapping.

7.9 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the Apache 2.0 license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- *Code of Conduct*

7.9.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

7.9.2 How to request a feature

Request features on the Issue Tracker.

7.9.3 How to set up your development environment

You need Python 3.9+ and the following tools:

- Poetry
- Nox
- nox-poetry

Another alternative is to use a Development Container has been setup to provide an environment with the required dependencies. This facilitates development on different systems.

This should seamlessly enable development for users of VS Code on systems with docker installed.

Known Issues

- `git config --global --add safe.directory $(pwd)` might be needed inside the container.

7.9.4 How to Install and Run segy

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run segy
```

7.9.5 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the *tests* directory, and are written using the pytest testing framework.

7.9.6 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

7.10 Contributor Covenant Code of Conduct

7.10.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

7.10.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

7.10.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

7.10.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

7.10.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at opensource@tgs.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

7.10.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

7.10.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

7.11 License

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.
```

(continues on next page)

(continued from previous page)

```
    "Source" form shall mean the preferred form for making modifications,
    including but not limited to software source code, documentation
    source, and configuration files.

    "Object" form shall mean any form resulting from mechanical
    transformation or translation of a Source form, including but
    not limited to compiled object code, generated documentation,
    and conversions to other media types.

    "Work" shall mean the work of authorship, whether in Source or
    Object form, made available under the License, as indicated by a
    copyright notice that is included in or attached to the work
    (an example is provided in the Appendix below).

    "Derivative Works" shall mean any work, whether in Source or Object
    form, that is based on (or derived from) the Work and for which the
    editorial revisions, annotations, elaborations, or other modifications
    represent, as a whole, an original work of authorship. For the purposes
    of this License, Derivative Works shall not include works that remain
    separable from, or merely link (or bind by name) to the interfaces of,
    the Work and Derivative Works thereof.

    "Contribution" shall mean any work of authorship, including
    the original version of the Work and any modifications or additions
    to that Work or Derivative Works thereof, that is intentionally
    submitted to Licensor for inclusion in the Work by the copyright owner
    or by an individual or Legal Entity authorized to submit on behalf of
    the copyright owner. For the purposes of this definition, "submitted"
    means any form of electronic, verbal, or written communication sent
    to the Licensor or its representatives, including but not limited to
    communication on electronic mailing lists, source code control systems,
    and issue tracking systems that are managed by, or on behalf of, the
    Licensor for the purpose of discussing and improving the Work, but
    excluding communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as "Not a Contribution."

    "Contributor" shall mean Licensor and any individual or Legal Entity
    on behalf of whom a Contribution has been received by Licensor and
    subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    copyright license to reproduce, prepare Derivative Works of,
    publicly display, publicly perform, sublicense, and distribute the
    Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    (except as stated in this section) patent license to make, have made,
```

(continues on next page)

(continued from previous page)

```
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.
```

(continues on next page)

(continued from previous page)

```
   5. Submission of Contributions. Unless You explicitly state otherwise,
      any Contribution intentionally submitted for inclusion in the Work
      by You to the Licensor shall be under the terms and conditions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or modify
      the terms of any separate license agreement you may have executed
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied, including, without limitation, any warranties or conditions
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
      PARTICULAR PURPOSE. You are solely responsible for determining the
      appropriateness of using or redistributing the Work and assume any
      risks associated with Your exercise of permissions under this License.

   8. Limitation of Liability. In no event and under no legal theory,
      whether in tort (including negligence), contract, or otherwise,
      unless required by applicable law (such as deliberate and grossly
      negligent acts) or agreed to in writing, shall any Contributor be
      liable to You for damages, including any direct, indirect, special,
      incidental, or consequential damages of any character arising as a
      result of this License or out of the use or inability to use the
      Work (including but not limited to damages for loss of goodwill,
      work stoppage, computer failure or malfunction, or any and all
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9. Accepting Warranty or Additional Liability. While redistributing
      the Work or Derivative Works thereof, You may choose to offer,
      and charge a fee for, acceptance of support, warranty, indemnity,
      or other liability obligations and/or rights consistent with this
      License. However, in accepting such obligations, You may act only
      on Your own behalf and on Your sole responsibility, not on behalf
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by reason
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the following
```

(continues on next page)

(continued from previous page)

```
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

INDEX

I

K

L

N

O

R

S

T

U

V