

OpenZGY/Python Public API (ALPHA)

Generated by Doxygen 1.8.17

1 Main Page	1
2 Example	3
3 Namespace Index	5
3.1 Packages	5
4 Hierarchical Index	7
4.1 Class Hierarchy	7
5 Class Index	9
5.1 Class List	9
6 Namespace Documentation	11
6.1 openzgy Namespace Reference	11
6.1.1 Detailed Description	11
6.2 openzgy.api Namespace Reference	11
6.2.1 Detailed Description	12
6.2.2 Function Documentation	13
6.2.2.1 ZgyCompressFactory()	13
6.2.2.2 ZgyKnownCompressors()	13
6.2.2.3 ZgyKnownDecompressors()	13
6.3 openzgy.exception Namespace Reference	13
6.3.1 Detailed Description	14
6.4 openzgy.iterator Namespace Reference	14
6.4.1 Detailed Description	14
6.4.2 Function Documentation	15
6.4.2.1 readall()	15
7 Class Documentation	17
7.1 openzgy.api.ProgressWithDots Class Reference	17
7.1.1 Detailed Description	17
7.2 openzgy.api.SampleDataType Class Reference	18
7.2.1 Detailed Description	18
7.3 openzgy.api.UnitDimension Class Reference	18
7.3.1 Detailed Description	19
7.4 openzgy.exception.ZgyAborted Class Reference	19
7.4.1 Detailed Description	19
7.5 openzgy.exception.ZgyCorruptedFile Class Reference	20
7.5.1 Detailed Description	20
7.6 openzgy.exception.ZgyEndOfFile Class Reference	20
7.6.1 Detailed Description	21
7.7 openzgy.exception.ZgyError Class Reference	21
7.7.1 Detailed Description	21

7.8 openzgy.exception.ZgyFormatError Class Reference	22
7.8.1 Detailed Description	22
7.9 openzgy.exception.ZgyInternalError Class Reference	22
7.9.1 Detailed Description	23
7.10 openzgy.api.ZgyMeta Class Reference	23
7.10.1 Detailed Description	24
7.10.2 Constructor & Destructor Documentation	24
7.10.2.1 __init__()	24
7.10.3 Member Function Documentation	24
7.10.3.1 annotcorners()	24
7.10.3.2 annotinc()	25
7.10.3.3 annotstart()	25
7.10.3.4 brickcount()	25
7.10.3.5 bricksizes()	25
7.10.3.6 corners()	25
7.10.3.7 datarange()	26
7.10.3.8 datatype()	26
7.10.3.9 histogram()	26
7.10.3.10 hunitdim()	27
7.10.3.11 hunitfactor()	27
7.10.3.12 hunitname()	27
7.10.3.13 indexcorners()	27
7.10.3.14 meta()	28
7.10.3.15 nlods()	28
7.10.3.16 numthreads()	28
7.10.3.17 raw_datarange()	28
7.10.3.18 size()	29
7.10.3.19 statistics()	29
7.10.3.20 zinc()	29
7.10.3.21 zstart()	29
7.10.3.22 zunitdim()	29
7.10.3.23 zunitfactor()	30
7.10.3.24 zunitname()	30
7.11 openzgy.api.ZgyMetaAndTools Class Reference	30
7.11.1 Detailed Description	31
7.11.2 Member Function Documentation	31
7.11.2.1 annotToIndex()	31
7.11.2.2 annotToWorld()	31
7.11.2.3 indexToAnnot()	31
7.11.2.4 indexToWorld()	32
7.11.2.5 transform()	32
7.11.2.6 worldToAnnot()	32

7.11.2.7 worldToIndex()	32
7.12 openzgy.exception.ZgyMissingFeature Class Reference	33
7.12.1 Detailed Description	33
7.13 openzgy.api.ZgyReader Class Reference	33
7.13.1 Detailed Description	34
7.13.2 Member Function Documentation	34
7.13.2.1 read()	34
7.13.2.2 readconst()	34
7.14 openzgy.exception.ZgySegmentIsClosed Class Reference	35
7.14.1 Detailed Description	36
7.15 openzgy.exception.ZgyUserError Class Reference	36
7.15.1 Detailed Description	36
7.16 openzgy.api.ZgyUtils Class Reference	36
7.16.1 Detailed Description	37
7.16.2 Constructor & Destructor Documentation	37
7.16.2.1 __init__()	37
7.16.3 Member Function Documentation	37
7.16.3.1 delete()	38
7.17 openzgy.api.ZgyWriter Class Reference	38
7.17.1 Detailed Description	39
7.17.2 Constructor & Destructor Documentation	39
7.17.2.1 __init__()	39
7.17.3 Member Function Documentation	40
7.17.3.1 close()	41
7.17.3.2 finalize()	41
7.17.3.3 write()	41
7.17.3.4 writeconst()	42
Index	43

Chapter 1

Main Page

The OpenZGY Python API allows read/write access to files stored in the ZGY format. The main part of the API is here:

- [ZgyReader](#) and its [ZgyMeta](#) base class.
- [ZgyWriter](#) also extending [ZgyMeta](#).
- [ZgyUtils](#) for anything not read or write.
- [exception.ZgyError](#) you might want to catch.
- [ProgressWithDots](#) example of progress reporting.
- [Example](#) Example application.

If you are reading this document from `doxygen/pure/apidoc.pdf` in the source tree then please see `doxygen/README.md` for an explanation of why the documentation produced by the build might be better.

Chapter 2

Example

```
#!/usr/bin/env python3
"""
Implement a simple copy command that exercises the new code.
"""
import numpy as np
import os, sys
from ..api import ZgyReader, ZgyWriter, ProgressWithDots, SampleDataType
from ..test.utils import SDCredentials
from ..iterator import readall
def suggest_range(value, dt):
    """Special case handling for inconsistent all-constant files."""
    dt_lo, dt_hi = {SampleDataType.int8: (-128, +127),
                    SampleDataType.int16: (-32768, +32767)}[dt]
    if value == 0:
        return (dt_lo / dt_hi, 1)
    elif value > 0:
        return (0, value * (1 - dt_hi / dt_lo))
    else:
        return (value * (1 - dt_lo / dt_hi), 0)
def copy_open_file(r, w, progress):
    """Simple example of manually iterating over files."""
    def _roundup(x, step): return ((x + step - 1) // step) * step
    blocksize = (r.bricksize[0], r.bricksize[1],
                 _roundup(r.size[2], r.bricksize[2]))
    total = (((r.size[0] + blocksize[0] - 1) // blocksize[0]) *
             ((r.size[1] + blocksize[1] - 1) // blocksize[1]))
    done = 0
    data = np.zeros(blocksize, dtype=np.float32)
    for ii in range(0, r.size[0], blocksize[0]):
        for jj in range(0, r.size[1], blocksize[1]):
            r.read((ii, jj, 0), data)
            w.write((ii, jj, 0), data)
            done += 1
            if progress: progress(done, total)
def copy_open_v2(r, w, progress):
    """Fewer lines of code but more going on behind the scenes."""
    for datastart, datasize, data in readall(r, progress=progress):
        w.write(datastart, data)
def copy(srcfilename, dstfilename):
    with ZgyReader(srcfilename, iocontext = SDCredentials()) as r:
        if r.datatype in (SampleDataType.int8, SampleDataType.int16):
            if r.raw_datarange[0] == r.raw_datarange[1]:
                datarange = suggest_range(r.raw_datarange[0], r.datatype)
                with ZgyWriter(dstfilename, templatename=srcfilename,
                               datarange = datarange,
                               iocontext = SDCredentials()) as w:
                    w.writeconst((0,0,0), r.raw_datarange[0], w.size, False)
                    w.finalize(progress=ProgressWithDots())
            return
        with ZgyWriter(dstfilename, templatename=srcfilename,
                       iocontext = SDCredentials()) as w:
            copy_open_file(r, w, progress=ProgressWithDots())
            w.finalize(progress=ProgressWithDots())
if __name__ == "__main__":
    np.seterr(all='raise')
    copy(sys.argv[1], sys.argv[2])
# Copyright 2017-2020, Schlumberger
#
# 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.
```

Chapter 3

Namespace Index

3.1 Packages

Here are the packages with brief descriptions (if available):

openzgy	The top level only has package members	11
openzgy.api	User visible apis are here	11
openzgy.exception	Exceptions that may be raised by OpenZGY	13
openzgy.iterator	Efficient iterator for reading an entire ZGY file	14

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Exception	
openzgy.exception.ZgyError	21
openzgy.exception.ZgyAborted	19
openzgy.exception.ZgyCorruptedFile	20
openzgy.exception.ZgyEndOfFile	20
openzgy.exception.ZgyFormatError	22
openzgy.exception.ZgyInternalError	22
openzgy.exception.ZgyMissingFeature	33
openzgy.exception.ZgySegmentIsClosed	35
openzgy.exception.ZgyUserError	36
openzgy.api.ProgressWithDots	17
openzgy.api.ZgyMeta	23
openzgy.api.ZgyMetaAndTools	30
openzgy.api.ZgyReader	33
openzgy.api.ZgyWriter	38
openzgy.api.ZgyUtils	36
Enum	
openzgy.api.SampleDataType	18
openzgy.api.UnitDimension	18

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

openzgy.api.ProgressWithDots	
Simple progress bar	17
openzgy.api.SampleDataType	
Sample data type used in the public API	18
openzgy.api.UnitDimension	
Horizontal or vertical dimension as used in the public API	18
openzgy.exception.ZgyAborted	
User aborted the operation	19
openzgy.exception.ZgyCorruptedFile	
The ZGY file became corrupted while writing to it	20
openzgy.exception.ZgyEndOfFile	
Trying to read past EOF	20
openzgy.exception.ZgyError	
Base class for all exceptions thrown by OpenZGY	21
openzgy.exception.ZgyFormatError	
Corrupted or unsupported ZGY file	22
openzgy.exception.ZgyInternalError	
Exception that might be caused by a bug in OpenZGY	22
openzgy.api.ZgyMeta	
Base class shared betewwn ZgyReader and ZgyWriter	23
openzgy.api.ZgyMetaAndTools	
Base class shared betewwn ZgyReader and ZgyWriter	30
openzgy.exception.ZgyMissingFeature	
Missing feature	33
openzgy.api.ZgyReader	
Main entry point for reading ZGY files	33
openzgy.exception.ZgySegmentIsClosed	
Exception used internally to request a retry	35
openzgy.exception.ZgyUserError	
Exception that might be caused by the calling application	36
openzgy.api.ZgyUtils	
Operations other than read and write	36
openzgy.api.ZgyWriter	
Main API for creating ZGY files	38

Chapter 6

Namespace Documentation

6.1 openzgy Namespace Reference

The top level only has package members.

Namespaces

- [api](#)

User visible apis are here.

- [exception](#)

Exceptions that may be raised by OpenZGY.

- [iterator](#)

Efficient iterator for reading an entire ZGY file.

6.1.1 Detailed Description

The top level only has package members.

6.2 openzgy.api Namespace Reference

User visible apis are here.

Classes

- class [ProgressWithDots](#)
Simple progress bar.
- class [SampleDataType](#)
Sample data type used in the public API.
- class [UnitDimension](#)
Horizontal or vertical dimension as used in the public API.
- class [ZgyMeta](#)
Base class shared betewwn [ZgyReader](#) and [ZgyWriter](#).
- class [ZgyMetaAndTools](#)
Base class shared betewwn [ZgyReader](#) and [ZgyWriter](#).
- class [ZgyReader](#)
Main entry point for reading ZGY files.
- class [ZgyUtils](#)
Operations other than read and write.
- class [ZgyWriter](#)
Main API for creating ZGY files.

Functions

- def [ZgyCompressFactory](#) (name, *args, **kwargs)
- def [ZgyKnownCompressors](#) ()
- def [ZgyKnownDecompressors](#) ()

6.2.1 Detailed Description

User visible apis are here.

This file contains the public OpenZGY API.

The API is modeled roughly after the API exposed by the Python wrapper around the existing C++ ZGY-Public API. This is probably just as good a starting point than anything else. And it makes testing simpler for those tests that compare the old and new behavior.

```
api.ZgyMeta:

    * Has a private reference to a impl.meta.ZgyInternalMeta instance.
    * Contains a large number of properties exposing meta data,
      most of which will present information from the ZgyInternalMeta
      in a way that is simpler to use and doesn't depend on the
      file version.
    * End users will access methods from this class. Most likely
      via the derived ZgyReader and ZgyWriter classes. The end users
      might not care that there is a separate base class.

api.ZgyMetaAndTools(ZgyMeta):

    * Add coordinate conversion routines to a ZgyMeta instance.

api.ZgyReader(ZgyMetaAndTools): # + read, readconst, close
api.ZgyWriter(ZgyMetaAndTools): # + write, writeconst, finalize, close

    * Has a private reference to a impl.meta.ZgyInternalBulk instance.
    * Add bulk read and write functionality, forwarding the requests
      to the internal bulk instance.
    * These classes with their own and inherited methods and properties
      comprise the public OpenZGY API.
    * Currently the ZgyWriter does not expose the bulk read() function
      but it does allow accessing all the metadata. Allowing read()
      might be added later if it appears to be useful.
    * In practice this just means to let ZgyWriter inherit ZgyReader.
```

6.2.2 Function Documentation

6.2.2.1 ZgyCompressFactory()

```
def openzgy.api.ZgyCompressFactory (
    name,
    * args,
    ** kwargs )
```

Look up a compression algorithm by name and instantiate a compressor, passing the required compression parameters. Using this approach reduces the coupling between client code and the compressor.

6.2.2.2 ZgyKnownCompressors()

```
def openzgy.api.ZgyKnownCompressors ( )
```

Return the names of all compressors known to the system. This is primarily for logging, but might in principle be used in a GUI to present a list of compressors to choose from. The problem with that is how to handle the argument list.

6.2.2.3 ZgyKnownDecompressors()

```
def openzgy.api.ZgyKnownDecompressors ( )
```

Return the names of all compressors known to the system. This is primarily for logging.

6.3 openzgy.exception Namespace Reference

Exceptions that may be raised by OpenZGY.

Classes

- class [ZgyAborted](#)
User aborted the operation.
- class [ZgyCorruptedFile](#)
The ZGY file became corrupted while writing to it.
- class [ZgyEndOfFile](#)
Trying to read past EOF.
- class [ZgyError](#)
Base class for all exceptions thrown by OpenZGY.
- class [ZgyFormatError](#)
Corrupted or unsupported ZGY file.
- class [ZgyInternalError](#)
Exception that might be caused by a bug in OpenZGY.
- class [ZgyMissingFeature](#)
Missing feature.
- class [ZgySegmentIsClosed](#)
Exception used internally to request a retry.
- class [ZgyUserError](#)
Exception that might be caused by the calling application.

6.3.1 Detailed Description

Exceptions that may be raised by OpenZGY.

Defines exceptions that may be raised by OpenZGY.

These classes are both visible to the OpenZGY public API and referenced directly from the implementation classes. I apologize for the broken encapsulation. Re-mapping the exceptions in the API layer didn't seem worth the trouble.

6.4 openzgy.iterator Namespace Reference

Efficient iterator for reading an entire ZGY file.

Functions

- def [readall](#) (r, *blocksize=None, chunksize=None, maxbytes=128 * 1024 * 1024, dtype=None, cropsiz=None, cropoffset=None, lod=0, sizeonly=False, progress=None)

6.4.1 Detailed Description

Efficient iterator for reading an entire ZGY file.

Efficient iterator for reading an entire ZGY file.

Naming conventions used in this file for parameters:

```
surveysize: (i,j,k)-- Size of entire survey without padding.
bricksize: (i,j,k) -- How data is stored. Usually (64, 64, 64).
blocksize: (i,j,k) -- How much to read (at most) in one call.
chunksize: (i,j,k) -- How much to return (at most) at once.
maxbytes: int64    -- Max bytes to read at once, defaults to 1 GB
dtype: np.dtype    -- Sample data type to return (float or storage).
readfn: callable   -- To be called when reading a block.
progress: callable -- Invoked after each read from disk.
```

6.4.2 Function Documentation

6.4.2.1 readall()

```
def openzgy.iterator.readall (
    r,
    * blocksize = None,
    chunksize = None,
    maxbytes = 128*1024*1024,
    dtype = None,
    cropsize = None,
    cropoffset = None,
    lod = 0,
    sizeonly = False,
    progress = None )
```

Convenience function to iterate over an entire cube, trying to use an efficient block size on read and optionally returning a smaller chunk size to the caller. The returned data buffer belongs to this iterator and may be overwritten on the next call. Two different iterators do not share their buffers.

Note: If blocksize is a multiple of chunksize and chunksize is a multiple of bricksizes then the following holds: if a chunk smaller than the requested chunksize is received then this can only be due to reading at the end or bottom of the survey. The caller might rely on this. E.g. when computing low resolution bricks and writing them out in a way that avoids a read/modify/write.

CAVEAT: This function might be overkill in some cases.

parameters:

```
r: ZgyReader          -- The open ZGY file
blocksize: (i,j,k)    -- How much to read (at most) in one call.
                        If omitted, use a reasonable default.
                        Dims set to 0 mean "as much as possible".
chunksize: (i,j,k)    -- How much to return (at most) at once.
                        Defaults to blocksize, i.e. return the
                        same blocks that are read. Will never
                        return more than blocksize, so if set to
                        more than 64 you probably want to set an
                        explicit blocksize as well.
maxbytes: int64       -- Max bytes to read at once, defaults to 128 MB
                        Ignored if an explicit blocksize is specified.
dtype: np.dtype       -- Data type of returned buffer. Defaults to
                        what is in the file. Use np.float32 to
                        convert and scale (if needed) to float.
cropsize: (i,j,k)    -- Only read this amount of data.
                        Note, no sanity check of this parameter.
cropoffset: (i,j,k)   -- Start reading at this point.
                        Note, no sanity check of this parameter.
lod: int              -- Pass >0 for decimated data.
sizeonly: bool        -- If false, return sizes but not data.
progress: callable    -- Invoked after each read from disk.
```


Chapter 7

Class Documentation

7.1 openzgy.api.ProgressWithDots Class Reference

Simple progress bar.

Public Member Functions

- `def __init__(self, length=51, outfile=sys.stderr)`
- `def __call__(self, done, total)`

7.1.1 Detailed Description

Simple progress bar.

Progress bar that writes dots (51 by default) to standard output. This can be user as-is for simple command line apps, or you can use the source code as an example on how to write your own.

The default of 51 dots will print one dot at startup and then one additional dot for each 2% work done.

If you are using this to write to the cloud a file that is smaller than ~10 GB then the progress bar will probably move in larger jumps. Because writing to a cloud back-end uses very large buffers. Most cloud back-ends cannot report progress inside a "write block".

When passing a progress reporter to a function, make sure you do not pass the class itself. You need to create an instance of it.

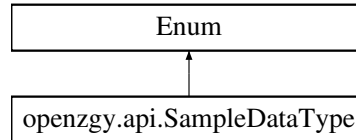
The documentation for this class was generated from the following file:

- `openzgy/api.py`

7.2 openzgy.api.SampleDataType Class Reference

Sample data type used in the public API.

Inheritance diagram for openzgy.api.SampleDataType:



Static Public Attributes

- int **unknown** = 1000
- int **int8** = 1001
- int **int16** = 1002
- int **float** = 1003

7.2.1 Detailed Description

Sample data type used in the public API.

Sample data type used in the public API.
Corresponds to `RawDataType` used in the ZGY file format.

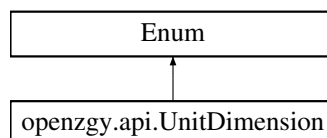
The documentation for this class was generated from the following file:

- `openzgy/api.py`

7.3 openzgy.api.UnitDimension Class Reference

Horizontal or vertical dimension as used in the public API.

Inheritance diagram for openzgy.api.UnitDimension:



Static Public Attributes

- int **unknown** = 2000
- int **time** = 2001
- int **length** = 2002
- int **arcangle** = 2003

7.3.1 Detailed Description

Horizontal or vertical dimension as used in the public API.

Horizontal or vertical dimension as used in the public API.

Horizontal dimension may be length or arc angle, although most applications only support length. Vertical dimension may be time or length. Vertical length is of course the same as depth. Arguably there should have been separate enums for horizontal and vertical dimension since the allowed values differ.

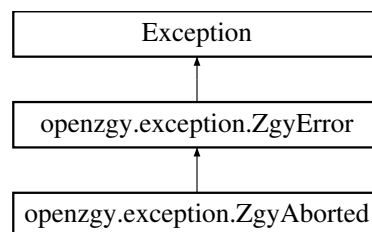
The documentation for this class was generated from the following file:

- openzgy/api.py

7.4 openzgy.exception.ZgyAborted Class Reference

User aborted the operation.

Inheritance diagram for openzgy.exception.ZgyAborted:



7.4.1 Detailed Description

User aborted the operation.

User aborted the operation.

If the user supplied a progress callback and this callback returned false then the operation in progress will and by throwing this exception. Which means that this is not an error; it is a consequence of the abort.

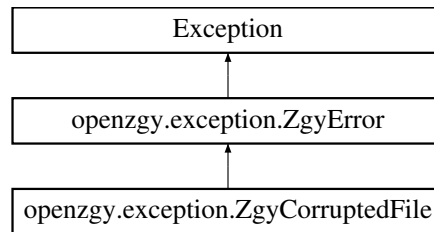
The documentation for this class was generated from the following file:

- openzgy/exception.py

7.5 openzgy.exception.ZgyCorruptedFile Class Reference

The ZGY file became corrupted while writing to it.

Inheritance diagram for openzgy.exception.ZgyCorruptedFile:



7.5.1 Detailed Description

The ZGY file became corrupted while writing to it.

The ZGY file became corrupted while writing to it.

No further writes are allowed on this file because a previous write raised an exception and we don't know the file's state. Subsequent writes will also throw this exception.

The safe approach is to assume that the error caused the file to become corrupted. It is recommended that the application closes and deletes the file.

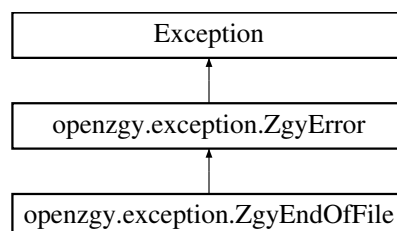
The documentation for this class was generated from the following file:

- openzgy/exception.py

7.6 openzgy.exception.ZgyEndOfFile Class Reference

Trying to read past EOF.

Inheritance diagram for openzgy.exception.ZgyEndOfFile:



7.6.1 Detailed Description

Trying to read past EOF.

```
Trying to read past EOF.
```

This is always considered an error, and is often due to a corrupted ZGY file. So this error should probably be treated as a `ZgyFormatError`.

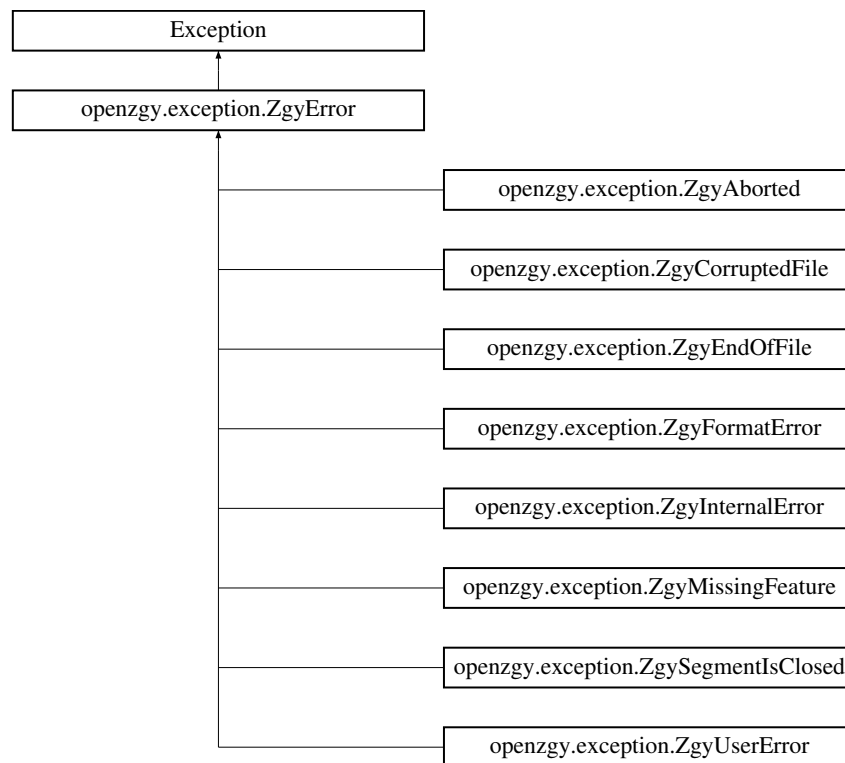
The documentation for this class was generated from the following file:

- `openzgy/exception.py`

7.7 openzgy.exception.ZgyError Class Reference

Base class for all exceptions thrown by OpenZGY.

Inheritance diagram for `openzgy.exception.ZgyError`:



7.7.1 Detailed Description

Base class for all exceptions thrown by OpenZGY.

```
Base class for all exceptions thrown by %OpenZGY.
```

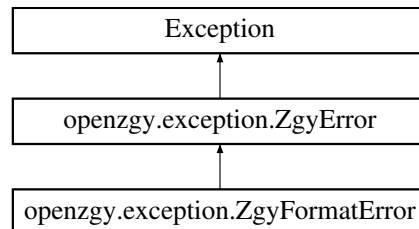
The documentation for this class was generated from the following file:

- `openzgy/exception.py`

7.8 openzgy.exception.ZgyFormatError Class Reference

Corrupted or unsupported ZGY file.

Inheritance diagram for openzgy.exception.ZgyFormatError:



7.8.1 Detailed Description

Corrupted or unsupported ZGY file.

Corrupted or unsupported ZGY file.

In some cases a corrupted file might lead to a `ZgyInternalError` or `ZgyEndOfFile` being thrown instead of this one. Because it isn't always easy to figure out the root cause.

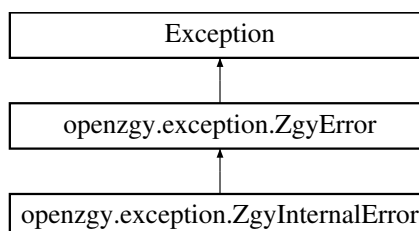
The documentation for this class was generated from the following file:

- openzgy/exception.py

7.9 openzgy.exception.ZgyInternalError Class Reference

Exception that might be caused by a bug in OpenZGY.

Inheritance diagram for openzgy.exception.ZgyInternalError:



7.9.1 Detailed Description

Exception that might be caused by a bug in OpenZGY.

Exception that might be caused by a bug in %OpenZGY.

Determining whether a problem is the fault of the calling application or the %OpenZGY library itself can be guesswork. Application code might choose to treat ZgyUserError and ZgyInternalError the same way.

A corrupt file might also be reported as ZgyInternalError instead of the more appropriate ZgyFormatError.

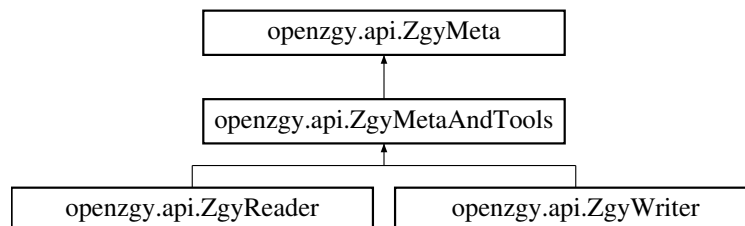
The documentation for this class was generated from the following file:

- openzgy/exception.py

7.10 openzgy.api.ZgyMeta Class Reference

Base class shared betewwn [ZgyReader](#) and [ZgyWriter](#).

Inheritance diagram for openzgy.api.ZgyMeta:



Public Member Functions

- def [__init__](#) (self, [meta](#))
- def [size](#) (self)
- def [datatype](#) (self)
- def [datarange](#) (self)
- def [raw_datarange](#) (self)
- def [zunitdim](#) (self)
- def [hunitdim](#) (self)
- def [zunitname](#) (self)
- def [hunitname](#) (self)
- def [zunitfactor](#) (self)
- def [hunitfactor](#) (self)
- def [zstart](#) (self)
- def [zinc](#) (self)
- def [annotstart](#) (self)
- def [annotinc](#) (self)
- def [corners](#) (self)
- def [indexcorners](#) (self)
- def [annotcorners](#) (self)

- def [bricksizes](#) (self)
- def [brickcount](#) (self)
- def [nlods](#) (self)
- def [meta](#) (self)
- def [numthreads](#) (self)
- def **numthreads** (self, x)
- def **dump** (self, file=None)
- def [statistics](#) (self)
- def [histogram](#) (self)

7.10.1 Detailed Description

Base class shared betewwn [ZgyReader](#) and [ZgyWriter](#).

Base class shared betewwn [ZgyReader](#) and [ZgyWriter](#).

7.10.2 Constructor & Destructor Documentation

7.10.2.1 `__init__()`

```
def openzgy.api.ZgyMeta.__init__ (
    self,
    meta )
```

Create an instance, providing the [ZgyInternalMeta](#) to use.

7.10.3 Member Function Documentation

7.10.3.1 `annotcorners()`

```
def openzgy.api.ZgyMeta.annotcorners (
    self )
```

Redundant with `Start, Inc, Size`.
Annotation coordinates of each of the 4 corners, ordered as `HCorners`.

7.10.3.2 annotinc()

```
def openzgy.api.ZgyMeta.annotinc (
    self )
```

Inline and crossline number increments between adjacent sections of the cube.

7.10.3.3 annotstart()

```
def openzgy.api.ZgyMeta.annotstart (
    self )
```

First inline and crossline numbers.

7.10.3.4 brickcount()

```
def openzgy.api.ZgyMeta.brickcount (
    self )
```

Number of bricks (including empties) ordered by [lod][dimension].

7.10.3.5 bricksizes()

```
def openzgy.api.ZgyMeta.bricksizes (
    self )
```

Size of a brick. Should always be (64, 64, 64).

7.10.3.6 corners()

```
def openzgy.api.ZgyMeta.corners (
    self )
```

World XY coordinates of each of the 4 corners.
The same coordinates in ordinal numbers are
(0, 0), (Size[0]-1, 0), (0, Size[1]-1), (Size[0]-1, Size[0]-1))

7.10.3.7 datarange()

```
def openzgy.api.ZgyMeta.datarange (
    self )
```

For integral data this is the lowest and highest sample value than can be represented in storage. The lowest storage value (e.g. -128 for SignedInt8 data) will be returned as DataMinMax[0] when read as float. Similarly the highest storage value e.g. +127 will be returned as DataMinMax[1]. When integer data is read as the "native" integral type then no automatic scaling is applied. Note that in this case the actual range of the samples on file might be smaller (for int8, not all of the range -128..+127 might be used) but it cannot be larger.

For floating point data these numbers are supposed to be the actual value range of the samples on file. It is a good idea to enforce this here, as the values stored by older writers cannot be trusted. Note: Also enforced on write in impl.meta.InfoHeaderV2.calculate_write. TODO-Worry: In some float32 datasets the bulk data might have ridiculously large spikes which will be included in the statistical range but not in the codingrange. So, codingrange is actually the one that is correct. Also, can we have a situation where stats are not filled in while the codingrange is set? I am not sure this is currently handled.

7.10.3.8 datatype()

```
def openzgy.api.ZgyMeta.datatype (
    self )
```

Sample data type.
The ZGY-Public API uses enums: "int8", "int16", "float".
In some cases these are also passed as strings.
The old Python wrapper for ZGY-Public is stringly typed.
Instead of returning a real enum it returns the name.

7.10.3.9 histogram()

```
def openzgy.api.ZgyMeta.histogram (
    self )
```

Return the statistics stored in the file header as a named tuple.
NOTE, I might want to change this to another type if there is a need to implement the same method in the ZGY-Public wrapper, as it might be trickier to define a namedtuple there.

7.10.3.10 hunitdim()

```
def openzgy.api.ZgyMeta.hunitdim (
    self )
```

Dimension in the horizontal direction. Should always be "length".
The original specification called for supporting "arcangle" as well,
i.e. coordinates in degrees instead of a projection. But most
application code that use ZGY will not support this.
The old Python wrapper for ZGY-Public is stringly typed.
Instead of returning a real enum it returns the name.

7.10.3.11 hunitfactor()

```
def openzgy.api.ZgyMeta.hunitfactor (
    self )
```

Factor to multiply stored horizontal values with to get SI units.

7.10.3.12 hunitname()

```
def openzgy.api.ZgyMeta.hunitname (
    self )
```

Unit in the horizontal direction. E.g. "m" or "ft".

7.10.3.13 indexcorners()

```
def openzgy.api.ZgyMeta.indexcorners (
    self )
```

Redundant with Size.
Ordinal coordinates of each of the 4 corners, ordered as "corners".

7.10.3.14 meta()

```
def openzgy.api.ZgyMeta.meta (
    self )
```

A dictionary of all the meta information, which can later be passed as `**kwargs` to the `ZgyWriter` constructor. and "indexcorners", "annotcorners", "brickcount", "nlods" are all derived properties that will never be settable. "numthreads" is a property of the implementation, not the file.

7.10.3.15 nlods()

```
def openzgy.api.ZgyMeta.nlods (
    self )
```

Number of level-of-detail layers, including lod 0 a.k.a. full resolution.

7.10.3.16 numthreads()

```
def openzgy.api.ZgyMeta.numthreads (
    self )
```

How many threads to use when reading. Currently ignored.

7.10.3.17 raw_datarange()

```
def openzgy.api.ZgyMeta.raw_datarange (
    self )
```

As datarange, but the actual values read from the file before they might have been changed to try to fix a bad file. Only use this property if you want to handle such files differently than the library does.

7.10.3.18 size()

```
def openzgy.api.ZgyMeta.size (
    self )
```

Number of inlines, crosslines, and samples in that order.

7.10.3.19 statistics()

```
def openzgy.api.ZgyMeta.statistics (
    self )
```

Return the statistics stored in the file header as a named tuple.
NOTE, I might want to change this to another type if there is a need to implement the same method in the ZGY-Public wrapper, as it might be trickier to define a namedtuple there.

7.10.3.20 zinc()

```
def openzgy.api.ZgyMeta.zinc (
    self )
```

Sample interval, given in the vertical unit.

7.10.3.21 zstart()

```
def openzgy.api.ZgyMeta.zstart (
    self )
```

Distance from surface/MSL to first sample, given in the vertical unit.

7.10.3.22 zunitdim()

```
def openzgy.api.ZgyMeta.zunitdim (
    self )
```

Dimension in the vertical direction. "time" or "length".
"time" might on file be "SeismicTWT" or "SeismicOWT".
The old Python wrapper for ZGY-Public is stringly typed.
Instead of returning a real enum it returns the name.

7.10.3.23 zunitfactor()

```
def openzgy.api.ZgyMeta.zunitfactor (
    self )
```

Factor to multiply stored vertical values with to get SI units.
E.g. 0.001 for ms, 1.0 for m or 0.3048 for ft.

7.10.3.24 zunitname()

```
def openzgy.api.ZgyMeta.zunitname (
    self )
```

Unit in the horizontal direction. E.g. "ms", "m", or "ft".
Note that Petrel might ignore this settings and instead
prompt the user to state what the unit should be.

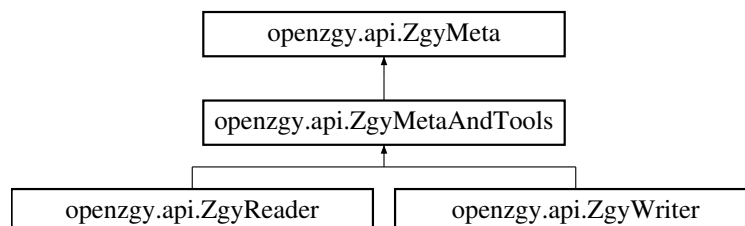
The documentation for this class was generated from the following file:

- openzgy/api.py

7.11 openzgy.api.ZgyMetaAndTools Class Reference

Base class shared betewwn [ZgyReader](#) and [ZgyWriter](#).

Inheritance diagram for openzgy.api.ZgyMetaAndTools:



Public Member Functions

- def [annotToIndex](#) (self, point)
- def [annotToWorld](#) (self, point)
- def [indexToAnnot](#) (self, point)
- def [indexToWorld](#) (self, point)
- def [worldToAnnot](#) (self, point)
- def [worldToIndex](#) (self, point)

Static Public Member Functions

- def [transform](#) (A, B, data)
- def [transform1](#) (A, B, point)

7.11.1 Detailed Description

Base class shared betewwn [ZgyReader](#) and [ZgyWriter](#).

Base class shared betewwn [ZgyReader](#) and [ZgyWriter](#).
Adds coordinate conversion tools.

7.11.2 Member Function Documentation

7.11.2.1 [annotToIndex\(\)](#)

```
def openzgy.api.ZgyMetaAndTools.annotToIndex (  
    self,  
    point )
```

Convert inline, crossline to ordinal

7.11.2.2 [annotToWorld\(\)](#)

```
def openzgy.api.ZgyMetaAndTools.annotToWorld (  
    self,  
    point )
```

Convert inline, crossline to world X,Y

7.11.2.3 [indexToAnnot\(\)](#)

```
def openzgy.api.ZgyMetaAndTools.indexToAnnot (  
    self,  
    point )
```

Convert ordinal to inline, crossline

7.11.2.4 indexToWorld()

```
def openzgy.api.ZgyMetaAndTools.indexToWorld (
    self,
    point )
```

Convert ordinal to world X,Y

7.11.2.5 transform()

```
def openzgy.api.ZgyMetaAndTools.transform (
    A,
    B,
    data ) [static]
```

Linear transformation of an array of double-precision coordinates.
The coordinate systems to convert between are defined by
three arbitrary points in the source system and the target.
Arguments: ((ax0,ay0), (ax1,ay1), (ax2,ay2)),
 ((bx0,by0), (bx1,by1), (bx2,by2)),
 data
where data is a 2d array of size (length, 2)

7.11.2.6 worldToAnnot()

```
def openzgy.api.ZgyMetaAndTools.worldToAnnot (
    self,
    point )
```

Convert world X,Y to inline, crossline

7.11.2.7 worldToIndex()

```
def openzgy.api.ZgyMetaAndTools.worldToIndex (
    self,
    point )
```

Convert world X,Y to ordinal

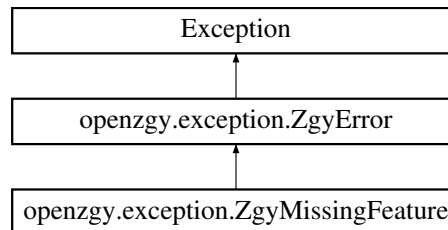
The documentation for this class was generated from the following file:

- openzgy/api.py

7.12 openzgy.exception.ZgyMissingFeature Class Reference

Missing feature.

Inheritance diagram for openzgy.exception.ZgyMissingFeature:



7.12.1 Detailed Description

Missing feature.

Missing feature.

Raised if some optional plug-in (e.g. some cloud back end or a compressor) was loaded or explicitly requested, so we know about it, but the plug-in is not operational for some reason.

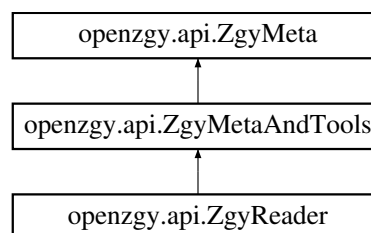
The documentation for this class was generated from the following file:

- openzgy/exception.py

7.13 openzgy.api.ZgyReader Class Reference

Main entry point for reading ZGY files.

Inheritance diagram for openzgy.api.ZgyReader:



Public Member Functions

- def **__init__** (self, filename, *_update=False, iocontext=None)
- def **__enter__** (self)
- def **__exit__** (self, type, value, traceback)
- def **read** (self, start, data, *lod=0, verbose=None)

Read bulk data into a caller specified buffer.
- def **readconst** (self, start, size, *lod=0, as_float=True, verbose=None)

Get hint about all constant region.
- def **close** (self)

Additional Inherited Members

7.13.1 Detailed Description

Main entry point for reading ZGY files.

Main entry point for reading ZGY files.

Obtain a concrete instance by calling the constructor.
You can then use the instance to read both meta data and bulk data.
It is recommended to explicitly close the file when done with it.

7.13.2 Member Function Documentation

7.13.2.1 read()

```
def openzgy.api.ZgyReader.read (
    self,
    start,
    data,
    * lod = 0,
    verbose = None )
```

Read bulk data into a caller specified buffer.

Read an arbitraty region of bulk data into a caller specified buffer.

The buffer's type must be float, short, or char. Any file may be read as float. If the buffer is of type short or char then the file must be of precisely that type.

Arguments: (i0,j0,k0), buffer, lod=0

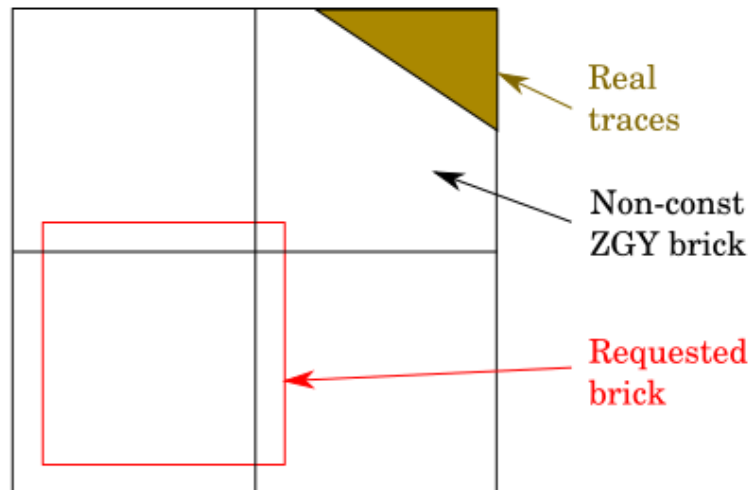
7.13.2.2 readconst()

```
def openzgy.api.ZgyReader.readconst (
    self,
    start,
    size,
    * lod = 0,
    as_float = True,
    verbose = None )
```

Get hint about all constant region.

readconst() will fail here because it only checks with the granularity of one ZGY brick.

readconst() will also fail if a brick was written non-const then later overwritten with const data.



Get hint about all constant region.

Check to see if the specified region is known to have all samples set to the same value. Returns that value, or None if it isn't.

The function only makes inexpensive checks so it might return None even if the region was in fact constant. It will not make the opposite mistake. This method is only intended as a hint to improve performance.

For int8 and int16 files the caller may specify whether to scale the values or not.

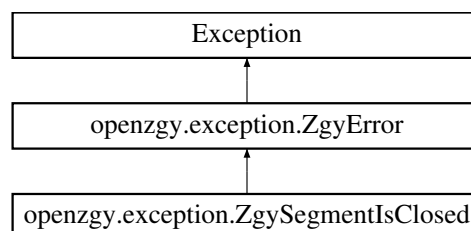
The documentation for this class was generated from the following file:

- openzgy/api.py

7.14 openzgy.exception.ZgySegmentIsClosed Class Reference

Exception used internally to request a retry.

Inheritance diagram for openzgy.exception.ZgySegmentIsClosed:



7.14.1 Detailed Description

Exception used internally to request a retry.

Exception used internally to request a retry.
A write to the cloud failed because the region that was attempted written had already been flushed. And the cloud back-end does not allow writing it again. The calling code, still inside the OpenZGY library, should be able to catch and recover from this problem.

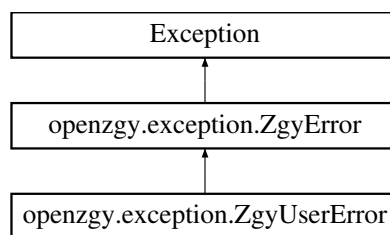
The documentation for this class was generated from the following file:

- `openzgy/exception.py`

7.15 `openzgy.exception.ZgyUserError` Class Reference

Exception that might be caused by the calling application.

Inheritance diagram for `openzgy.exception.ZgyUserError`:



7.15.1 Detailed Description

Exception that might be caused by the calling application.

Exception that might be caused by the calling application.

Determining whether a problem is the fault of the calling application or the %OpenZGY library itself can be guesswork. Application code might choose to treat `ZgyUserError` and `ZgyInternalError` the same way.

The documentation for this class was generated from the following file:

- `openzgy/exception.py`

7.16 `openzgy.api.ZgyUtils` Class Reference

Operations other than read and write.

Public Member Functions

- def `__init__` (self, iocontext=None)
Create a new concrete instance of [ZgyUtils](#).
- def `delete` (self, filename)
Delete a file.

7.16.1 Detailed Description

Operations other than read and write.

Operations other than read and write.

Any operations that don't fit into ZgyReader or ZgyWriter go here. Such as deleting a file. Or any other operation that does not need the file to be open first.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 `__init__()`

```
def openzgy.api.ZgyUtils.__init__ (
    self,
    iocontext = None )
```

Create a new concrete instance of [ZgyUtils](#).

Create a new concrete instance of ZgyUtils.

The reason you need to supply a file name or a file name prefix is that you need to provide enough information to identify the back-end that this instance will be bound to. So if you have registered a back-end named "xx", both "xx://some/bogus/file.zgy" and just "xx://" will produce an instance that works for your XX backend,

For performance reasons you should consider caching one ZgyUtils instance for each back end you will be using. Instead of just creating a new one each time you want to invoke a method. Just remember that most operations need an instance created with the same prefix.

7.16.3 Member Function Documentation

7.16.3.1 delete()

```
def openzgy.api.ZgyUtils.delete (
    self,
    filename )
```

Delete a file.

Works both for local and cloud files.

Delete a file. Works both for local and cloud files.
Note that the instance must be of the correct (local or cloud) type.

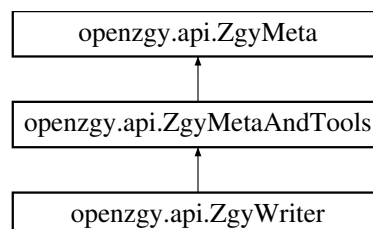
The documentation for this class was generated from the following file:

- openzgy/api.py

7.17 openzgy.api.ZgyWriter Class Reference

Main API for creating ZGY files.

Inheritance diagram for openzgy.api.ZgyWriter:



Public Member Functions

- def `__init__` (self, filename, *iocontext=None, compressor=None, lodcompressor=None, **kwargs)
- def `__enter__` (self)
- def `__exit__` (self, type, value, traceback)
- def `write` (self, start, data, *verbose=None)
Write an arbitrary region.
- def `writeconst` (self, start, value, size, is_storage, *verbose=None)
Write a single value to a region of the file.
- def `finalize` (self, *decimation=None, progress=None, force=False, verbose=None)
- def `close` (self)

Additional Inherited Members

7.17.1 Detailed Description

Main API for creating ZGY files.

Main API for creating ZGY files.

Obtain a concrete instance by calling the constructor.
All meta data is specified in the call to `open()`, so meta data will appear to be read only. You can use the instance to write bulk data. The file becomes read only once the instance is closed.

It is recommended to call `finalize()` and `close()` after all bulk has been written. But if you forget then this will be done when the writer goes out of scope, provided of course that you used a "with" block.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 `__init__()`

```
def openzgy.api.ZgyWriter.__init__ (
    self,
    filename,
    * ioccontext = None,
    compressor = None,
    lodcompressor = None,
    ** kwargs )
```

Create a new ZGY file.

Optionally pass `templatename = otherfile` to create a new file similar to `otherfile`. Explicit keyword arguments override information from `otherfile`.

Optionally pass `templatename = filename` to erase all data blocks from `filename` but keep the metadata. New data blocks can then be written to the file. Petrel/BASE might need this feature, due to the way it writes new files. They tend to get opened several times to add meta information. Caveat: Behind the scenes the file is simply deleted and re-created. This is slightly less efficient than opening the file for read/write.

`templatename`: string
Optionally create a file similar to this one.
TODO-Low: In the future might also accept a `ZgyReader` instance.
This is convenient if a file is being copied, so as to not needing to open it twice.

`filename`: string
The local or cloud file to create.

`size`: (int, int, int)
Number of inlines, crosslines, samples.

`bricksize`: (int, int, int)
Size of a single brick. Defaults to (64, 64, 64).

Please use the default unless you really know what you are doing. In any case, each size needs to be a power of 2.

`datatype: SampleDataType`
Specify int8, int16, or float.

`datarange = (float, float)`
Used only if `datatype` is integral, to convert from storage to actual sample values. The lowest possible storage value, i.e. -128 or -32768, maps to `datarange[0]` and the highest possible storage value maps to `datarange[1]`.

`zunitdim: UnitDimension.time, length, or unknown.`
`zunitname: string`
`zunitfactor: float`
Describe how to convert between storage units and SI units in the vertical direction. Petrel ignores these settings and prompts the user.

`hunitdim: UnitDimension.length, arcangle, or unknown.`
`hunitname: string`
`hunitfactor: float`
Describe how to convert between storage units and SI units in the horizontal direction. Most applications cannot handle arcangle. Petrel ignores these settings and prompts the user.

`zstart: float`
The time or depth corresponding to the shallowest sample.

`zinc: float`
The vertical time or depth distance between neighboring samples.

`annotstart: (float, float)`
The inline and crossline numbers corresponding to the ordinal position (0, 0) i.e. the first sample on the file.

`annotinc: (float, float)`
The inline / crossline step between neighboring samples.
The samples at ordinal (1, 1) will have annotation
`annotstart + annotinc`.

`corners: (float, float)[4]`
World coordinates of each corner, order as
First inline / first crossline,
last inline / first crossline,
first inline / last crossline,
last inline / last crossline.

`compressor, lodcompressor: callable`
If set, attempt to compress each block with this callable. Typically this should be a lambda or a class, because it needs to capture the compression parameters.
Example:

```
compressor = ZgyCompressFactory("ZFP", snr = 30)
```

If different compression parameters are desired for full- and low resolution bricks then `lodcompressor` can be provided as well. It defaults to `compressor`. Using two different instances, even if the parameters match, may also cause statistics to be reported separately for fullres and lowres.
TODO-Low: Future: passing `zfp_compressor = snr` is equivalent to `compressor = ZgyCompressFactory("ZFP", snr = snr)`. Unlike the `compressor` keyword this also works in the wrapper.

7.17.3 Member Function Documentation

7.17.3.1 close()

```
def openzgy.api.ZgyWriter.close (
    self )
```

Close the currently open file.
Failure to close the file will corrupt it.

7.17.3.2 finalize()

```
def openzgy.api.ZgyWriter.finalize (
    self,
    * decimation = None,
    progress = None,
    force = False,
    verbose = None )
```

Generate low resolution data, statistics, and histogram.
This will be called automatically from close(), but in that case it is not possible to request a progress callback.

If the processing raises an exception the data is still marked as clean. Called can force a retry by passing force=True.

Arguments:

- decimation: Optionally override the decimation algorithms by passing an array of impl.lodalgo.DecimationType with one entry for each level of detail. If the array is too short then the last entry is used for subsequent levels.
TODO-Low: The expected enum type is technically internal and ought to have been mapped to an enum api.XXX.
- progress: Function(done, total) called to report progress. If it returns False the computation is aborted.
Will be called at least one, even if there is no work.
- force: If true, generate the low resolution data even if it appears to not be needed. Use with caution.
Especially if writing to the cloud, where data should only be written once.
- verbose: optional function to print diagnostic information.

7.17.3.3 write()

```
def openzgy.api.ZgyWriter.write (
    self,
    start,
    data,
    * verbose = None )
```

Write an arbitrary region.

Write bulk data. Type must be np.float32, np.int16, or np.int8. np.float32 may be written to any file and will be converted if needed before storing. Writing an integral type implies that values are to be written without conversion. In that case the type of the buffer must match exactly the file's storage type. You cannot write int8 data to an int16 file or vice versa.

Arguments:

- start: tuple(i0,j0,k0) where to start writing.
- data: np.ndarray of np.int8, np.int16, or np.float32

7.17.3.4 writeconst()

```
def openzgy.api.ZgyWriter.writeconst (
    self,
    start,
    value,
    size,
    is_storage,
    * verbose = None )
```

Write a single value to a region of the file.

Write a single value to a region of the file.

This is equivalent to creating a constant-value array with `np.full()` and write that. But this version might be considerably faster.

If `is_storage` is false and the input value cannot be converted to storage values due to being outside range after conversion then the normal rules (use closest valid value) apply. If `is_storage` is True then an error is raised if the supplied value cannot be represented.

Arguments:

- `start:` tuple(i0,j0,k0) where to start writing.
- `size:` tuple(ni,nj,nk) size of region to write.
- `value:` Scalar to be written.
- `is_storge:` False if the value shall be converted to storage
True if it is already storage and should be written
unchanged. Ignored if the storage type is float.

The documentation for this class was generated from the following file:

- `openzgy/api.py`

Index

- `__init__`
 - [openzgy.api.ZgyMeta, 24](#)
 - [openzgy.api.ZgyUtils, 37](#)
 - [openzgy.api.ZgyWriter, 39](#)
- `annotcorners`
 - [openzgy.api.ZgyMeta, 24](#)
- `annotinc`
 - [openzgy.api.ZgyMeta, 24](#)
- `annotstart`
 - [openzgy.api.ZgyMeta, 25](#)
- `annotToIndex`
 - [openzgy.api.ZgyMetaAndTools, 31](#)
- `annotToWorld`
 - [openzgy.api.ZgyMetaAndTools, 31](#)
- `brickcount`
 - [openzgy.api.ZgyMeta, 25](#)
- `bricksize`
 - [openzgy.api.ZgyMeta, 25](#)
- `close`
 - [openzgy.api.ZgyWriter, 40](#)
- `corners`
 - [openzgy.api.ZgyMeta, 25](#)
- `datarange`
 - [openzgy.api.ZgyMeta, 25](#)
- `datatype`
 - [openzgy.api.ZgyMeta, 26](#)
- `delete`
 - [openzgy.api.ZgyUtils, 37](#)
- `finalize`
 - [openzgy.api.ZgyWriter, 41](#)
- `histogram`
 - [openzgy.api.ZgyMeta, 26](#)
- `hunitdim`
 - [openzgy.api.ZgyMeta, 26](#)
- `hunitfactor`
 - [openzgy.api.ZgyMeta, 27](#)
- `hunitname`
 - [openzgy.api.ZgyMeta, 27](#)
- `indexcorners`
 - [openzgy.api.ZgyMeta, 27](#)
- `indexToAnnot`
 - [openzgy.api.ZgyMetaAndTools, 31](#)
- `indexToWorld`
 - [openzgy.api.ZgyMetaAndTools, 31](#)

- `meta`
 - [openzgy.api.ZgyMeta, 27](#)
- `nlods`
 - [openzgy.api.ZgyMeta, 28](#)
- `numthreads`
 - [openzgy.api.ZgyMeta, 28](#)
- `openzgy, 11`
- `openzgy.api, 11`
 - [ZgyCompressFactory, 13](#)
 - [ZgyKnownCompressors, 13](#)
 - [ZgyKnownDecompressors, 13](#)
- `openzgy.api.ProgressWithDots, 17`
- `openzgy.api.SampleDataType, 18`
- `openzgy.api.UnitDimension, 18`
- `openzgy.api.ZgyMeta, 23`
 - `__init__, 24`
 - `annotcorners, 24`
 - `annotinc, 24`
 - `annotstart, 25`
 - `brickcount, 25`
 - `bricksize, 25`
 - `corners, 25`
 - `datarange, 25`
 - `datatype, 26`
 - `histogram, 26`
 - `hunitdim, 26`
 - `hunitfactor, 27`
 - `hunitname, 27`
 - `indexcorners, 27`
 - `meta, 27`
 - `nlods, 28`
 - `numthreads, 28`
 - `raw_datarange, 28`
 - `size, 28`
 - `statistics, 29`
 - `zinc, 29`
 - `zstart, 29`
 - `zunitdim, 29`
 - `zunitfactor, 29`
 - `zunitname, 30`
- `openzgy.api.ZgyMetaAndTools, 30`
 - [annotToIndex, 31](#)
 - [annotToWorld, 31](#)
 - [indexToAnnot, 31](#)
 - [indexToWorld, 31](#)
 - [transform, 32](#)
 - [worldToAnnot, 32](#)
 - [worldToIndex, 32](#)

- openzgy.api.ZgyReader, 33
 - read, 34
 - readconst, 34
- openzgy.api.ZgyUtils, 36
 - __init__, 37
 - delete, 37
- openzgy.api.ZgyWriter, 38
 - __init__, 39
 - close, 40
 - finalize, 41
 - write, 41
 - writeconst, 41
- openzgy.exception, 13
- openzgy.exception.ZgyAborted, 19
- openzgy.exception.ZgyCorruptedFile, 20
- openzgy.exception.ZgyEndOfFile, 20
- openzgy.exception.ZgyError, 21
- openzgy.exception.ZgyFormatError, 22
- openzgy.exception.ZgyInternalError, 22
- openzgy.exception.ZgyMissingFeature, 33
- openzgy.exception.ZgySegmentIsClosed, 35
- openzgy.exception.ZgyUserError, 36
- openzgy.iterator, 14
 - readall, 15
- raw_datarange
 - openzgy.api.ZgyMeta, 28
- read
 - openzgy.api.ZgyReader, 34
- readall
 - openzgy.iterator, 15
- readconst
 - openzgy.api.ZgyReader, 34
- size
 - openzgy.api.ZgyMeta, 28
- statistics
 - openzgy.api.ZgyMeta, 29
- transform
 - openzgy.api.ZgyMetaAndTools, 32
- worldToAnnot
 - openzgy.api.ZgyMetaAndTools, 32
- worldToIndex
 - openzgy.api.ZgyMetaAndTools, 32
- write
 - openzgy.api.ZgyWriter, 41
- writeconst
 - openzgy.api.ZgyWriter, 41
- ZgyCompressFactory
 - openzgy.api, 13
- ZgyKnownCompressors
 - openzgy.api, 13
- ZgyKnownDecompressors
 - openzgy.api, 13
- zinc
 - openzgy.api.ZgyMeta, 29
- zstart
 - openzgy.api.ZgyMeta, 29
- zunitdim
 - openzgy.api.ZgyMeta, 29
- zunitfactor
 - openzgy.api.ZgyMeta, 29
- zunitname
 - openzgy.api.ZgyMeta, 30