

# Alpha version. Expect changes.

OpenZGY/Python API and Internals (ALPHA)

Generated by Doxygen 1.8.17

Alpha version. Expect changes.

# Alpha version. Expect changes.

i

<b>1 Main Page</b>	<b>1</b>
<b>2 Example</b>	<b>3</b>
<b>3 Namespace Index</b>	<b>5</b>
3.1 Packages	5
<b>4 Hierarchical Index</b>	<b>7</b>
4.1 Class Hierarchy	7
<b>5 Class Index</b>	<b>9</b>
5.1 Class List	9
<b>6 Namespace Documentation</b>	<b>11</b>
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.impl Namespace Reference	14
6.4.1 Detailed Description	15
6.5 openzgy.impl.bulk Namespace Reference	15
6.5.1 Detailed Description	15
6.6 openzgy.impl.enum Namespace Reference	15
6.6.1 Detailed Description	16
6.7 openzgy.impl.file Namespace Reference	16
6.7.1 Detailed Description	17
6.7.2 Function Documentation	17
6.7.2.1 FileFactory()	17
6.8 openzgy.impl.lodalgo Namespace Reference	17
6.8.1 Detailed Description	18
6.8.2 Function Documentation	18
6.8.2.1 decimate()	18
6.8.2.2 decimate8()	18
6.9 openzgy.impl.meta Namespace Reference	19
6.9.1 Detailed Description	19
6.10 openzgy.impl.stats Namespace Reference	20
6.10.1 Detailed Description	20
6.11 openzgy.impl.transform Namespace Reference	21

# Alpha version. Expect changes.

ii

6.11.1 Detailed Description	21
6.11.2 Function Documentation	22
6.11.2.1 acpToOcp()	22
6.11.2.2 generalTransform()	23
6.12 openzgy.iterator Namespace Reference	23
6.12.1 Detailed Description	23
6.12.2 Function Documentation	24
6.12.2.1 readall()	24
6.13 openzgy.zgypublic Namespace Reference	24
6.13.1 Detailed Description	25
<b>7 Class Documentation</b>	<b>27</b>
7.1 openzgy.api._internal Class Reference	27
7.1.1 Detailed Description	27
7.2 openzgy.impl.meta.AlphaLUPV1 Class Reference	28
7.3 openzgy.impl.meta.AlphaLUPV2 Class Reference	28
7.4 openzgy.impl.meta.AlphaLUPV3 Class Reference	28
7.5 openzgy.impl.meta.BrickLUPV1 Class Reference	29
7.6 openzgy.impl.meta.BrickLUPV2 Class Reference	29
7.7 openzgy.impl.meta.BrickLUPV3 Class Reference	30
7.8 openzgy.impl.enum.BrickStatus Class Reference	30
7.8.1 Detailed Description	30
7.9 openzgy.impl.compress.CompressFactoryImpl Class Reference	31
7.9.1 Detailed Description	31
7.9.2 Member Function Documentation	31
7.9.2.1 decompress()	31
7.9.2.2 knownCompressors()	31
7.9.2.3 knownDecompressors()	32
7.9.2.4 registerCompressor()	32
7.9.2.5 registerDecompressor()	33
7.10 openzgy.impl.compress.CompressPlugin Class Reference	33
7.10.1 Detailed Description	34
7.10.2 Constructor & Destructor Documentation	34
7.10.2.1 __init__()	34
7.10.3 Member Function Documentation	34
7.10.3.1 __call__()	35
7.10.3.2 compress()	35
7.10.3.3 decompress()	35
7.10.3.4 dump()	36
7.11 openzgy.impl.compress.CompressStats Class Reference	37
7.12 openzgy.impl.file.Config Class Reference	37
7.13 openzgy.impl.lodalgo.DecimationType Class Reference	37

7.13.1 Detailed Description	38
7.14 openzgy.impl.bulk.ErrorsWillCorruptFile Class Reference	38
7.14.1 Detailed Description	39
7.15 openzgy.impl.meta.ErrorsWillCorruptFile Class Reference	39
7.15.1 Detailed Description	39
7.16 openzgy.impl.file.FileADT Class Reference	39
7.16.1 Constructor & Destructor Documentation	40
7.16.1.1 __init__()	40
7.16.2 Member Function Documentation	40
7.16.2.1 xx_close()	40
7.16.2.2 xx_read()	41
7.16.2.3 xx_readv()	41
7.16.2.4 xx_write()	42
7.17 openzgy.impl.file.FileConfig Class Reference	42
7.17.1 Constructor & Destructor Documentation	42
7.17.1.1 __init__()	43
7.18 openzgy.impl.meta.FileHeader Class Reference	43
7.18.1 Detailed Description	43
7.19 openzgy.impl.genlod.GenLodBase Class Reference	43
7.19.1 Detailed Description	44
7.20 openzgy.impl.genlod.GenLodC Class Reference	44
7.20.1 Detailed Description	45
7.21 openzgy.impl.genlod.GenLodImpl Class Reference	45
7.21.1 Detailed Description	45
7.21.2 Member Function Documentation	45
7.21.2.1 __call__()	46
7.22 openzgy.impl.meta.HeaderBase Class Reference	46
7.22.1 Detailed Description	46
7.22.2 Member Function Documentation	46
7.22.2.1 checkformats()	47
7.22.2.2 dump()	47
7.22.2.3 headersize()	47
7.22.2.4 pack()	47
7.22.2.5 read()	48
7.22.2.6 unpack()	48
7.23 openzgy.impl.meta.HistHeaderV1 Class Reference	48
7.23.1 Constructor & Destructor Documentation	48
7.23.1.1 __init__()	49
7.24 openzgy.impl.meta.HistHeaderV2 Class Reference	49
7.24.1 Constructor & Destructor Documentation	49
7.24.1.1 __init__()	49
7.25 openzgy.impl.meta.HistHeaderV3 Class Reference	50

# Alpha version. Expect changes.

iv

7.26 openzgy.impl.histogram.HistogramData Class Reference	50
7.26.1 Member Function Documentation	50
7.26.1.1 binvalue()	50
7.26.1.2 np_range()	51
7.26.1.3 vv_range()	51
7.27 openzgy.impl.meta.InfoHeaderV1 Class Reference	51
7.27.1 Constructor & Destructor Documentation	51
7.27.1.1 __init__()	52
7.28 openzgy.impl.meta.InfoHeaderV2 Class Reference	52
7.28.1 Constructor & Destructor Documentation	52
7.28.1.1 __init__()	52
7.28.2 Member Function Documentation	53
7.28.2.1 calculate_write()	53
7.29 openzgy.impl.meta.InfoHeaderV3 Class Reference	53
7.30 openzgy.impl.file.LocalFile Class Reference	53
7.30.1 Constructor & Destructor Documentation	54
7.30.1.1 __init__()	54
7.30.2 Member Function Documentation	54
7.30.2.1 xx_close()	54
7.31 openzgy.impl.file.LocalFileLinux Class Reference	55
7.31.1 Member Function Documentation	55
7.31.1.1 xx_read()	55
7.31.1.2 xx_readv()	56
7.31.1.3 xx_write()	56
7.32 openzgy.impl.file.LocalFileOther Class Reference	57
7.32.1 Member Function Documentation	57
7.32.1.1 xx_read()	57
7.32.1.2 xx_readv()	58
7.32.1.3 xx_write()	58
7.33 openzgy.impl.meta.LookupTable Class Reference	59
7.33.1 Detailed Description	59
7.33.2 Constructor & Destructor Documentation	59
7.33.2.1 __init__()	60
7.34 openzgy.impl.meta.OffsetHeaderV1 Class Reference	60
7.34.1 Constructor & Destructor Documentation	60
7.34.1.1 __init__()	60
7.34.2 Member Function Documentation	60
7.34.2.1 calculate()	61
7.35 openzgy.impl.meta.OffsetHeaderV2 Class Reference	61
7.35.1 Constructor & Destructor Documentation	61
7.35.1.1 __init__()	61
7.35.2 Member Function Documentation	62

# Alpha version. Expect changes.

v

7.35.2.1 calculate()	62
7.36 openzgy.impl.meta.OffsetHeaderV3 Class Reference	62
7.37 openzgy.api.ProgressWithDots Class Reference	62
7.37.1 Detailed Description	63
7.38 openzgy.impl.enum.RawCoordType Class Reference	63
7.38.1 Detailed Description	64
7.39 openzgy.impl.enum.RawDataType Class Reference	64
7.39.1 Detailed Description	64
7.40 openzgy.impl.enum.RawGridDefinition Class Reference	65
7.40.1 Detailed Description	65
7.41 openzgy.impl.enum.RawHorizontalDimension Class Reference	65
7.41.1 Detailed Description	66
7.42 openzgy.impl.enum.RawVerticalDimension Class Reference	66
7.42.1 Detailed Description	66
7.43 openzgy.api.SampleDataType Class Reference	66
7.43.1 Detailed Description	67
7.44 openzgy.impl.bulk.ScalarBuffer Class Reference	67
7.44.1 Detailed Description	67
7.45 openzgy.impl.file.SDConfig Class Reference	68
7.45.1 Constructor & Destructor Documentation	68
7.45.1.1 __init__()	68
7.45.2 Member Function Documentation	70
7.45.2.1 extra()	70
7.46 openzgy.impl.file.SeismicStoreFile Class Reference	71
7.46.1 Detailed Description	71
7.46.2 Constructor & Destructor Documentation	71
7.46.2.1 __init__()	72
7.46.3 Member Function Documentation	72
7.46.3.1 xx_close()	72
7.46.3.2 xx_read()	72
7.46.3.3 xx_readv()	73
7.46.3.4 xx_write()	73
7.47 openzgy.impl.file.SeismicStoreFileDelayedWrite Class Reference	73
7.47.1 Detailed Description	74
7.47.2 Constructor & Destructor Documentation	74
7.47.2.1 __init__()	75
7.47.3 Member Function Documentation	75
7.47.3.1 xx_close()	75
7.47.3.2 xx_eof()	75
7.47.3.3 xx_read()	76
7.47.3.4 xx_write()	76
7.48 openzgy.impl.stats.StatisticData Class Reference	76

# Alpha version. Expect changes.

7.48.1 Detailed Description	77
7.48.2 Member Function Documentation	77
7.48.2.1 scale()	77
7.49 openzgy.impl.meta.StringListV1 Class Reference	77
7.49.1 Detailed Description	78
7.50 openzgy.impl.meta.StringListV2 Class Reference	78
7.50.1 Detailed Description	78
7.50.2 Constructor & Destructor Documentation	78
7.50.2.1 __init__()	78
7.50.3 Member Function Documentation	79
7.50.3.1 size()	79
7.51 openzgy.impl.meta.StringListV3 Class Reference	79
7.52 openzgy.api.UnitDimension Class Reference	79
7.52.1 Detailed Description	80
7.53 openzgy.impl.enum.UpdateMode Class Reference	80
7.53.1 Detailed Description	81
7.54 openzgy.impl.file.UsageHint Class Reference	81
7.55 openzgy.impl.zfp_compress.ZfpCompressPlugin Class Reference	82
7.55.1 Detailed Description	82
7.55.2 Member Function Documentation	82
7.55.2.1 __call__()	82
7.55.2.2 decompress()	83
7.56 openzgy.exception.ZgyAborted Class Reference	84
7.56.1 Detailed Description	84
7.57 openzgy.exception.ZgyCorruptedFile Class Reference	84
7.57.1 Detailed Description	85
7.58 openzgy.exception.ZgyEndOfFile Class Reference	85
7.58.1 Detailed Description	85
7.59 openzgy.exception.ZgyError Class Reference	86
7.59.1 Detailed Description	86
7.60 openzgy.exception.ZgyFormatError Class Reference	86
7.60.1 Detailed Description	87
7.61 openzgy.impl.bulk.ZgyInternalBulk Class Reference	87
7.61.1 Detailed Description	87
7.61.2 Member Function Documentation	87
7.61.2.1 readConstantValue()	87
7.61.2.2 readToExistingBuffer()	88
7.62 openzgy.exception.ZgyInternalError Class Reference	88
7.62.1 Detailed Description	88
7.63 openzgy.impl.meta.ZgyInternalMeta Class Reference	89
7.63.1 Detailed Description	89
7.64 openzgy.api.ZgyMeta Class Reference	89



7.64.1 Detailed Description	90
7.64.2 Constructor & Destructor Documentation	90
7.64.2.1 <code>__init__()</code>	90
7.64.3 Member Function Documentation	90
7.64.3.1 <code>annotcorners()</code>	90
7.64.3.2 <code>annotinc()</code>	91
7.64.3.3 <code>annotstart()</code>	91
7.64.3.4 <code>brickcount()</code>	91
7.64.3.5 <code>bricksize()</code>	91
7.64.3.6 <code>corners()</code>	91
7.64.3.7 <code>datarange()</code>	92
7.64.3.8 <code>datatype()</code>	92
7.64.3.9 <code>histogram()</code>	92
7.64.3.10 <code>hunitdim()</code>	93
7.64.3.11 <code>hunitfactor()</code>	93
7.64.3.12 <code>hunitname()</code>	93
7.64.3.13 <code>indexcorners()</code>	93
7.64.3.14 <code>meta()</code>	94
7.64.3.15 <code>nlods()</code>	94
7.64.3.16 <code>numthreads()</code>	94
7.64.3.17 <code>size()</code>	94
7.64.3.18 <code>statistics()</code>	94
7.64.3.19 <code>zinc()</code>	95
7.64.3.20 <code>zstart()</code>	95
7.64.3.21 <code>zunitdim()</code>	95
7.64.3.22 <code>zunitfactor()</code>	95
7.64.3.23 <code>zunitname()</code>	95
7.65 <code>openzgy.api.ZgyMetaAndTools</code> Class Reference	96
7.65.1 Detailed Description	96
7.65.2 Member Function Documentation	96
7.65.2.1 <code>annotToIndex()</code>	96
7.65.2.2 <code>annotToWorld()</code>	97
7.65.2.3 <code>indexToAnnot()</code>	97
7.65.2.4 <code>indexToWorld()</code>	97
7.65.2.5 <code>transform()</code>	97
7.65.2.6 <code>worldToAnnot()</code>	98
7.65.2.7 <code>worldToIndex()</code>	98
7.66 <code>openzgy.exception.ZgyMissingFeature</code> Class Reference	98
7.66.1 Detailed Description	98
7.67 <code>openzgy.api.ZgyReader</code> Class Reference	99
7.67.1 Detailed Description	99
7.67.2 Member Function Documentation	99

# Alpha version. Expect changes.

viii

7.67.2.1 read()	100
7.67.2.2 readconst()	100
7.68 openzgy.zgypublic.ZgyReader Class Reference	100
7.68.1 Member Function Documentation	101
7.68.1.1 bricksizes()	101
7.68.1.2 meta()	101
7.69 openzgy.exception.ZgySegmentIsClosed Class Reference	102
7.69.1 Detailed Description	102
7.70 openzgy.exception.ZgyUserError Class Reference	102
7.70.1 Detailed Description	102
7.71 openzgy.api.ZgyUtils Class Reference	103
7.71.1 Detailed Description	103
7.71.2 Constructor & Destructor Documentation	103
7.71.2.1 __init__()	103
7.71.3 Member Function Documentation	103
7.71.3.1 delete()	104
7.72 openzgy.zgypublic.ZgyWriter Class Reference	104
7.72.1 Member Function Documentation	104
7.72.1.1 bricksizes()	105
7.72.1.2 meta()	105
7.73 openzgy.api.ZgyWriter Class Reference	105
7.73.1 Detailed Description	106
7.73.2 Constructor & Destructor Documentation	106
7.73.2.1 __init__()	106
7.73.3 Member Function Documentation	107
7.73.3.1 close()	107
7.73.3.2 errorflag() [1/2]	108
7.73.3.3 errorflag() [2/2]	108
7.73.3.4 finalize()	108
7.73.3.5 write()	109
7.73.3.6 writeconst()	109
<b>Index</b>	<b>111</b>

# Alpha version. Expect changes.

## 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 viewing the full Doxygen documentation then this covers both the API and most of the implementation. So if you look at the list of classes and methods this might seem a bit daunting. All you really need to use the API should be in the above list. Excluding trivial structs that will be cross referenced as needed. So you don't need to go looking for them. Of course, if you want to work on OpenZGY itself then you probably need everything.

See also the following related pages:

- [physicalformat](#)
- [implementation](#)
- [lowres](#)
- [migration](#)



# Alpha version. Expect changes.

## 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
from ..test.utils import SDCredentials
from ..iterator import readall
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:
        with ZgyWriter(dstfilename, templatenamename=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.
```

# Alpha version. Expect changes.

# Alpha version. Expect changes.

## Chapter 3

### Namespace Index

#### 3.1 Packages

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

<a href="#">openzgy</a>	The top level only has package members . . . . .	11
<a href="#">openzgy.api</a>	User visible apis are here . . . . .	11
<a href="#">openzgy.exception</a>	Exceptions that may be raised by OpenZGY . . . . .	13
<a href="#">openzgy.impl</a>	All implementation details are in this namespace . . . . .	14
<a href="#">openzgy.impl.bulk</a>	Reading and writing bulk data . . . . .	15
<a href="#">openzgy.impl.enum</a>	Enums not visible to the public API . . . . .	15
<a href="#">openzgy.impl.file</a>	Low level I/O, abstract layer . . . . .	16
<a href="#">openzgy.impl.lodalgo</a>	Decimation algorithms to output low resolution bricks . . . . .	17
<a href="#">openzgy.impl.meta</a>	Meta data read/write . . . . .	19
<a href="#">openzgy.impl.stats</a>	Code to compute statistics . . . . .	20
<a href="#">openzgy.impl.transform</a>	Deal with spatial location of the cube . . . . .	21
<a href="#">openzgy.iterator</a>	Efficient iterator for reading an entire ZGY file . . . . .	23
<a href="#">openzgy.zgypublic</a>	Consolidate old and new Python API . . . . .	24





# Alpha version. Expect changes.

## Chapter 4

### Hierarchical Index

#### 4.1 Class Hierarchy

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

openzgy.api._internal . . . . .	27
openzgy.impl.compress.CompressFactoryImpl . . . . .	31
openzgy.impl.compress.CompressPlugin . . . . .	33
openzgy.impl.zfp_compress.ZfpCompressPlugin . . . . .	82
openzgy.impl.compress.CompressStats . . . . .	37
openzgy.impl.file.Config . . . . .	37
openzgy.impl.file.FileConfig . . . . .	42
openzgy.impl.file.SDConfig . . . . .	68
openzgy.impl.bulk.ErrorsWillCorruptFile . . . . .	38
openzgy.impl.meta.ErrorsWillCorruptFile . . . . .	39
Exception	
openzgy.exception.ZgyError . . . . .	86
openzgy.exception.ZgyAborted . . . . .	84
openzgy.exception.ZgyCorruptedFile . . . . .	84
openzgy.exception.ZgyEndOfFile . . . . .	85
openzgy.exception.ZgyFormatError . . . . .	86
openzgy.exception.ZgyInternalError . . . . .	88
openzgy.exception.ZgyMissingFeature . . . . .	98
openzgy.exception.ZgySegmentIsClosed . . . . .	102
openzgy.exception.ZgyUserError . . . . .	102
openzgy.impl.file.FileADT . . . . .	39
openzgy.impl.file.LocalFile . . . . .	53
openzgy.impl.file.LocalFileLinux . . . . .	55
openzgy.impl.file.LocalFileOther . . . . .	57
openzgy.impl.file.SeismicStoreFile . . . . .	71
openzgy.impl.file.SeismicStoreFileDelayedWrite . . . . .	73
openzgy.impl.genlod.GenLodBase . . . . .	43
openzgy.impl.genlod.GenLodImpl . . . . .	45
openzgy.impl.genlod.GenLodC . . . . .	44
openzgy.impl.meta.HeaderBase . . . . .	46
openzgy.impl.meta.FileHeader . . . . .	43
openzgy.impl.meta.HistHeaderV1 . . . . .	48
openzgy.impl.meta.HistHeaderV2 . . . . .	49
openzgy.impl.meta.HistHeaderV3 . . . . .	50

openzgy.impl.meta.InfoHeaderV1 . . . . .	51
openzgy.impl.meta.InfoHeaderV2 . . . . .	52
openzgy.impl.meta.InfoHeaderV3 . . . . .	53
openzgy.impl.histogram.HistogramData . . . . .	50
openzgy.impl.meta.LookupTable . . . . .	59
openzgy.impl.meta.AlphaLUPV1 . . . . .	28
openzgy.impl.meta.AlphaLUPV2 . . . . .	28
openzgy.impl.meta.AlphaLUPV3 . . . . .	28
openzgy.impl.meta.BrickLUPV1 . . . . .	29
openzgy.impl.meta.BrickLUPV2 . . . . .	29
openzgy.impl.meta.BrickLUPV3 . . . . .	30
openzgy.impl.meta.OffsetHeaderV1 . . . . .	60
openzgy.impl.meta.OffsetHeaderV2 . . . . .	61
openzgy.impl.meta.OffsetHeaderV3 . . . . .	62
openzgy.api.ProgressWithDots . . . . .	62
openzgy.impl.bulk.ScalarBuffer . . . . .	67
openzgy.impl.stats.StatisticData . . . . .	76
openzgy.impl.meta.StringListV1 . . . . .	77
openzgy.impl.meta.StringListV2 . . . . .	78
openzgy.impl.meta.StringListV3 . . . . .	79
openzgy.impl.bulk.ZgyInternalBulk . . . . .	87
openzgy.impl.meta.ZgyInternalMeta . . . . .	89
openzgy.api.ZgyMeta . . . . .	89
openzgy.api.ZgyMetaAndTools . . . . .	96
openzgy.api.ZgyReader . . . . .	99
openzgy.api.ZgyWriter . . . . .	105
ZgyReader	
openzgy.zgypublic.ZgyReader . . . . .	100
openzgy.api.ZgyUtils . . . . .	103
ZgyWriter	
openzgy.zgypublic.ZgyWriter . . . . .	104
Enum	
openzgy.api.SampleDataType . . . . .	66
openzgy.api.UnitDimension . . . . .	79
openzgy.impl.enum.BrickStatus . . . . .	30
openzgy.impl.enum.RawCoordType . . . . .	63
openzgy.impl.enum.RawDataType . . . . .	64
openzgy.impl.enum.RawGridDefinition . . . . .	65
openzgy.impl.enum.RawHorizontalDimension . . . . .	65
openzgy.impl.enum.RawVerticalDimension . . . . .	66
openzgy.impl.enum.UpdateMode . . . . .	80
openzgy.impl.file.UsageHint . . . . .	81
openzgy.impl.lodalgo.DecimationType . . . . .	37

# Alpha version. Expect changes.

## Chapter 5

## Class Index

### 5.1 Class List

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

<a href="#">openzgy.api._internal</a>	
Explicit control of imported symbols	27
<a href="#">openzgy.impl.meta.AlphaLUPV1</a>	28
<a href="#">openzgy.impl.meta.AlphaLUPV2</a>	28
<a href="#">openzgy.impl.meta.AlphaLUPV3</a>	28
<a href="#">openzgy.impl.meta.BrickLUPV1</a>	29
<a href="#">openzgy.impl.meta.BrickLUPV2</a>	29
<a href="#">openzgy.impl.meta.BrickLUPV3</a>	30
<a href="#">openzgy.impl.enum.BrickStatus</a>	30
<a href="#">openzgy.impl.compress.CompressFactoryImpl</a>	31
<a href="#">openzgy.impl.compress.CompressPlugin</a>	33
<a href="#">openzgy.impl.compress.CompressStats</a>	37
<a href="#">openzgy.impl.file.Config</a>	37
<a href="#">openzgy.impl.lodalgo.DecimationType</a>	
Possible algorithms to generate LOD bricks	37
<a href="#">openzgy.impl.bulk.ErrorsWillCorruptFile</a>	38
<a href="#">openzgy.impl.meta.ErrorsWillCorruptFile</a>	39
<a href="#">openzgy.impl.file.FileADT</a>	39
<a href="#">openzgy.impl.file.FileConfig</a>	42
<a href="#">openzgy.impl.meta.FileHeader</a>	43
<a href="#">openzgy.impl.genlod.GenLodBase</a>	43
<a href="#">openzgy.impl.genlod.GenLodC</a>	44
<a href="#">openzgy.impl.genlod.GenLodImpl</a>	45
<a href="#">openzgy.impl.meta.HeaderBase</a>	46
<a href="#">openzgy.impl.meta.HistHeaderV1</a>	48
<a href="#">openzgy.impl.meta.HistHeaderV2</a>	49
<a href="#">openzgy.impl.meta.HistHeaderV3</a>	50
<a href="#">openzgy.impl.histogram.HistogramData</a>	50
<a href="#">openzgy.impl.meta.InfoHeaderV1</a>	51
<a href="#">openzgy.impl.meta.InfoHeaderV2</a>	52
<a href="#">openzgy.impl.meta.InfoHeaderV3</a>	53
<a href="#">openzgy.impl.file.LocalFile</a>	53
<a href="#">openzgy.impl.file.LocalFileLinux</a>	55
<a href="#">openzgy.impl.file.LocalFileOther</a>	57
<a href="#">openzgy.impl.meta.LookupTable</a>	59

<a href="#">openzgy.impl.meta.OffsetHeaderV1</a>	60
<a href="#">openzgy.impl.meta.OffsetHeaderV2</a>	61
<a href="#">openzgy.impl.meta.OffsetHeaderV3</a>	62
<a href="#">openzgy.api.ProgressWithDots</a>	
Simple progress bar	62
<a href="#">openzgy.impl.enum.RawCoordType</a>	63
<a href="#">openzgy.impl.enum.RawDataType</a>	64
<a href="#">openzgy.impl.enum.RawGridDefinition</a>	65
<a href="#">openzgy.impl.enum.RawHorizontalDimension</a>	65
<a href="#">openzgy.impl.enum.RawVerticalDimension</a>	66
<a href="#">openzgy.api.SampleDataType</a>	
Sample data type used in the public API	66
<a href="#">openzgy.impl.bulk.ScalarBuffer</a>	67
<a href="#">openzgy.impl.file.SDConfig</a>	68
<a href="#">openzgy.impl.file.SeismicStoreFile</a>	71
<a href="#">openzgy.impl.file.SeismicStoreFileDelayedWrite</a>	73
<a href="#">openzgy.impl.stats.StatisticData</a>	76
<a href="#">openzgy.impl.meta.StringListV1</a>	77
<a href="#">openzgy.impl.meta.StringListV2</a>	78
<a href="#">openzgy.impl.meta.StringListV3</a>	79
<a href="#">openzgy.api.UnitDimension</a>	
Horizontal or vertical dimension as used in the public API	79
<a href="#">openzgy.impl.enum.UpdateMode</a>	80
<a href="#">openzgy.impl.file.UsageHint</a>	81
<a href="#">openzgy.impl.zfp_compress.ZfpCompressPlugin</a>	82
<a href="#">openzgy.exception.ZgyAborted</a>	
User aborted the operation	84
<a href="#">openzgy.exception.ZgyCorruptedFile</a>	
The ZGY file became corrupted while writing to it	84
<a href="#">openzgy.exception.ZgyEndOfFile</a>	
Trying to read past EOF	85
<a href="#">openzgy.exception.ZgyError</a>	
Base class for all exceptions thrown by OpenZGY	86
<a href="#">openzgy.exception.ZgyFormatError</a>	
Corrupted or unsupported ZGY file	86
<a href="#">openzgy.impl.bulk.ZgyInternalBulk</a>	87
<a href="#">openzgy.exception.ZgyInternalError</a>	
Exception that might be caused by a bug in OpenZGY	88
<a href="#">openzgy.impl.meta.ZgyInternalMeta</a>	89
<a href="#">openzgy.api.ZgyMeta</a>	
Base class shared betewwn <a href="#">ZgyReader</a> and <a href="#">ZgyWriter</a>	89
<a href="#">openzgy.api.ZgyMetaAndTools</a>	
Base class shared betewwn <a href="#">ZgyReader</a> and <a href="#">ZgyWriter</a>	96
<a href="#">openzgy.exception.ZgyMissingFeature</a>	
Missing feature	98
<a href="#">openzgy.api.ZgyReader</a>	
Main entry point for reading ZGY files	99
<a href="#">openzgy.zgypublic.ZgyReader</a>	100
<a href="#">openzgy.exception.ZgySegmentIsClosed</a>	
Exception used internally to request a retry	102
<a href="#">openzgy.exception.ZgyUserError</a>	
Exception that might be caused by the calling application	102
<a href="#">openzgy.api.ZgyUtils</a>	
Operations other than read and write	103
<a href="#">openzgy.zgypublic.ZgyWriter</a>	104
<a href="#">openzgy.api.ZgyWriter</a>	
Main API for creating ZGY files	105

# Alpha version. Expect changes.

## 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.*
- [impl](#)  
*All implementation details are in this namespace.*
- [iterator](#)  
*Efficient iterator for reading an entire ZGY file.*
- [zgypublic](#)  
*Consolidate old and new Python API.*

#### 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 [\\_internal](#)  
*Explicit control of imported symbols.*
- 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.impl Namespace Reference

All implementation details are in this namespace.

## Namespaces

- [bulk](#)  
*Reading and writing bulk data.*
- [enum](#)  
*Enums not visible to the public API.*
- [file](#)  
*Low level I/O, abstract layer.*
- [lodalgo](#)  
*Decimation algorithms to output low resolution bricks.*
- [meta](#)  
*Meta data read/write.*
- [stats](#)  
*Code to compute statistics.*
- [transform](#)  
*Deal with spatial location of the cube.*



### 6.4.1 Detailed Description

All implementation details are in this namespace.

## 6.5 openzgy.impl.bulk Namespace Reference

Reading and writing bulk data.

### Classes

- class [ErrorsWillCorruptFile](#)
- class [ScalarBuffer](#)
- class [ZgyInternalBulk](#)

### Functions

- def **dprint** (\*args, \*\*kwargs)

### 6.5.1 Detailed Description

Reading and writing bulk data.

This file contains a number of classes and free functions dealing with bulk data access. There is a similar file `impl.meta` that deals with metadata. Nothing in this file should be directly visible to users of the public API.

`impl.bulk.ZgyInternalBulk:`

- \* Internal methods to read and write bulk.
- \* Needs access to much of the meta data information.  
Currently this is handled by giving this class references to those file headers it needs, e.g. `InfoHeader`, `BrickLUP`, etc.  
TODO-Low: Refactor, would it be cleaner to instead provide a reference to the `ZgyInternalMeta` instance? This gives access to more metadata than the class actually needs. But would provide better isolation nevertheless.
- \* Refactoring notes: There are a couple of sections in this class that ought to have been separated out. As it is now the class does too much.
  - Write support
  - Lod generation
  - Histogram generation

## 6.6 openzgy.impl.enum Namespace Reference

Enums not visible to the public API.

## Classes

- class [BrickStatus](#)
- class [RawCoordType](#)
- class [RawDataType](#)
- class [RawGridDefinition](#)
- class [RawHorizontalDimension](#)
- class [RawVerticalDimension](#)
- class [UpdateMode](#)

### 6.6.1 Detailed Description

Enums not visible to the public API.

```
Enums not visible to the public API.
```

```
Some of the enums are used to describe parts of the file format,  
giving a symbolic name to an integer stored in the file.  
That kind of information should definitely be hidden from clients.
```

## 6.7 openzgy.impl.file Namespace Reference

Low level I/O, abstract layer.

## Classes

- class [Config](#)
- class [FileADT](#)
- class [FileConfig](#)
- class [LocalFile](#)
- class [LocalFileLinux](#)
- class [LocalFileOther](#)
- class [SDConfig](#)
- class [SeismicStoreFile](#)
- class [SeismicStoreFileDelayedWrite](#)
- class [UsageHint](#)

## Functions

- def [FileFactory](#) (filename, mode, iocontext)

## Variables

- **sd** = None

### 6.7.1 Detailed Description

Low level I/O, abstract layer.

Low level I/O, abstract layer.

This file contains classes that do low level I/O either to on-prem data using the regular read and write methods of the OS or to seismic store using SDAPI. Other back ends can easily be added.

Some work is done to consolidate read requests and to buffer write requests in the cloud back end. This is needed to get acceptable performance. This explains why this file is quite large.

```
impl.file.Config:
    impl.file.FileConfig(Config):
    impl.file.SDConfig(Config):

    * Details such as user credentials etc. established when the
      file is open. Specific to the backend type.
    * Note that there is currently no way to pass a configuration
      object along with every read and write request. This might
      have been useful for a server type application but would
      require the config parameter to ripple across at least 50
      existing methods. I doubt this would be worth the trouble.

impl.file.FileADT:
    impl.file.LocalFile(FileADT):
        impl.file.LocalFileOther(LocalFile):
        impl.file.LocalFileLinux(LocalFile):
    impl.file.SeismicStoreFile(FileADT):
    impl.file.SeismicStoreFileDelayedWrite(FileADT):

    * Higher level code should only access the polymorphic FileADT
      base class and the impl.file.FileFactory that creates an
      instance of the desired type.
```

### 6.7.2 Function Documentation

#### 6.7.2.1 FileFactory()

```
def openzgy.impl.file.FileFactory (
    filename,
    mode,
    iocontext )
```

Return a FileADT instance able to read and/or write to the named file.  
In the future the function might return different types of instances  
e.g. if filename refers to something on the cloud.

## 6.8 openzgy.impl.lodalgo Namespace Reference

Decimation algorithms to output low resolution bricks.

## Classes

- class [DecimationType](#)  
*Possible algorithms to generate LOD bricks.*

## Functions

- def [decimate](#) (brick, algo, \*\*kwargs)
- def [decimate8](#) (bricks, algo, \*\*kwargs)

### 6.8.1 Detailed Description

Decimation algorithms to output low resolution bricks.

Decimation algorithms to output low resolution bricks.

### 6.8.2 Function Documentation

#### 6.8.2.1 [decimate\(\)](#)

```
def openzgy.impl.lodalgo.decimate (
    brick,
    algo,
    ** kwargs )
```

Decimate a single input brick to produce one smaller output brick.

#### 6.8.2.2 [decimate8\(\)](#)

```
def openzgy.impl.lodalgo.decimate8 (
    bricks,
    algo,
    ** kwargs )
```

Decimate 8 equal-sized input bricks to produce one output.

Currently not used in production code. Only in the unit test.  
The genlod module prefers to split up and paste its data itself.  
This function should perhaps be moved to the unit test module.

Most of the decimation algorithms operate on a local 2x2x2 region for each output sample. So it makes no difference whether we decimate each input brick and then combine them, or the other way around. Combining last *might* use less memory if lazy evaluation works. Combining first might help the algorithm perform better (LowPass) or faster (WeightedAverage?)

## 6.9 openzgy.impl.meta Namespace Reference

Meta data read/write.

### Classes

- class [AlphaLUPV1](#)
- class [AlphaLUPV2](#)
- class [AlphaLUPV3](#)
- class [BrickLUPV1](#)
- class [BrickLUPV2](#)
- class [BrickLUPV3](#)
- class [ErrorsWillCorruptFile](#)
- class [FileHeader](#)
- class [HeaderBase](#)
- class [HistHeaderV1](#)
- class [HistHeaderV2](#)
- class [HistHeaderV3](#)
- class [InfoHeaderV1](#)
- class [InfoHeaderV2](#)
- class [InfoHeaderV3](#)
- class [LookupTable](#)
- class [OffsetHeaderV1](#)
- class [OffsetHeaderV2](#)
- class [OffsetHeaderV3](#)
- class [StringListV1](#)
- class [StringListV2](#)
- class [StringListV3](#)
- class [ZgyInternalMeta](#)

### Functions

- def **OffsetHeaderFactory** (version)
- def **InfoHeaderFactory** (version)
- def **StringListFactory** (version)
- def **HistHeaderFactory** (version)
- def **AlphaLUPFactory** (version)
- def **BrickLUPFactory** (version)
- def **checkAllFormats** (\*verbose=False, file=None)

### 6.9.1 Detailed Description

Meta data read/write.

Meta data read/write

This file contains a number of classes and free functions dealing with meta data access, i.e. everything in the ZGY file except the actual bulk data. Nothing in this file should be directly visible to users of the public API.

impl.meta.ZgyInternalMeta:

- \* Holds a collection of header instances, one of each type.
- \* A function to populate all the instances, mainly by invoking read() in each header instance, in the right order.  
With the current format there are several chicken and egg problems.
- \* A function to populate all the header instances when creating a new file.
- \* A function to flush all headers instances to file after writing.
- \* The user visible ZgyReader and ZgyWriter classes should contain an instance of this class, using composition not inheritance because there are no methods or data in this class that the user is allowed to use directly.

FileHeader  
OffsetHeader{V1,V2,V3}  
InfoHeader{V1,V2,V3}  
StringList{V1,V2,V3}  
HistHeader{V1,V2,V3}  
AlphaLUP{V1,V2,V3}  
BrickLUP{V1,V2,V3}:

- \* All these represent a particular version of one of the headers that make up a complete ZGY file. Each header knows about the physical layout on the file. In C++ the class should be a POD with an exact match to what is found on file. If additional data members are needed they should be introduced in a derived class. In Python there are no POD types so specifying the physical layout is a bit more involved. And any derived data might as well be included in the same class.

OffsetHeaderFactory, InfoHeaderFactory, etc.

- \* Free functions to create an instance of this type, with the numerical version number passed in.

HeaderBase, LookupTable

- \* These are "convenience" base classes for sharing code; they are not intended to provide polymorphic behavior.

## 6.10 openzgy.impl.stats Namespace Reference

Code to compute statistics.

### Classes

- class [StatisticData](#)

### 6.10.1 Detailed Description

Code to compute statistics.

Code to compute statistics.

impl.stats.StatisticData:

- \* Accumulate statistics: count, sum, sum of squares, and value range.

## 6.11 openzgy.impl.transform Namespace Reference

Deal with spatial location of the cube.

### Functions

- def [generalTransform](#) (ax0, ay0, ax1, ay1, ax2, ay2, bx0, by0, bx1, by1, bx2, by2, data)
- def [acpToOcp](#) (orig, inc, size, il, xl, wx, wy)

### 6.11.1 Detailed Description

Deal with spatial location of the cube.

Deal with spatial location of the cube.

The grid definition in gpiline, gpxline, gpx, and gpy defines an affine transform from inline, crossline to world X, Y. Older ZGY files might use several ways to specify the transform.

Since an affine transform is used it is possible to define a coordinate system where the X and Y axes are not precisely perpendicular to each other. It is up to the application reading the files to decide whether to accept non-orthogonal coordinate systems. When writing files you should think long and hard about whether you really need to allow the user to create a non orthogonal ZGY file. If you do, you might regret it after a few decades of supporting that feature in all your applications that read ZGY.

"FourPoint" is what new ZGY files should use on write. This provides the coordinates of the four corners of the survey both in ordinal (i, j), annotation (inline, crossline), and world (X, Y) coordinates. The order of these points is:

```
First inline / first crossline,  
last inline / first crossline,  
first inline / last crossline,  
last inline / last crossline.
```

The API will always return the lattice as "FourPoint", even if the ZGY file uses a different way of specifying geometry.

The ordinal cornere are not stored. They are implied to be

```
(          0,          0)  
(size[0] - 1,          0)  
(          0, size[1] - 1)  
(size[0] - 1, size[1] - 1)
```

The annotation corners are required to be stored, even though this is redundant with "orig" and "inc":

```
(orig[0],          orig[1])  
(orig[0]+inc[0]*(size[0]-1), orig[1])  
(orig[0],          orig[1]+inc[1]*(size[1]-1))  
(orig[0]+inc[0]*(size[0]-1), orig[1]+inc[1]*(size[1]-1))
```

The world X, Y coordinates are required to be stored for all four corners, in spite of the last corner being redundant. As long as the transform is required to be affine, three points are sufficient.

"ThreePoint" is more general. Instead of storing the four corners, three arbitrary points are specified with both inline, crossline and world X, Y coordinates. The last point in the arrays is unused. As

long as the points are not colinear or duplicated, this is enough to define the affine transform.

The "Parametric" type is not used. The original intent was that with "Parametric", the "gazim" field would contain Inline, crossline azimuths (a.k.a. "base angle" and "side angle", respectively) as 360-degree compass direction (0 = N, 90 = E, 180 = S, 270 = W). The "gbinsz" field would contain Inline, crossline bin size as METERS always (i.e. NOT dependent on hdim or hunit). To my knowledge, no ZGY files have ever been written using these fields. So, readers should treat both "Unknown" and "Parametric" as errors.

New files should always be written using "FourPoint" to assure that all ZGY readers handle them correctly. When reading, always assume "ThreePoint" on the file but convert to "FourPoint" before providing presenting the information to the user.

This means the reader should trust that the first three points in (gpiline, gpiline) line up with the first three points in (gpx, gpy). The reader should not trust that those points are in fact the corners. And the reader should ignore the last point completely. ZGY files in the wild might have been written explicitly in "ThreePoint" mode, or they might accidentally have been written as "FourPoint" with the corners in the wrong order. Reading in "ThreePoint" mode handles all those cases.

## 6.11.2 Function Documentation

### 6.11.2.1 acpToOcp()

```
def openzgy.impl.transform.acpToOcp (
    orig,
    inc,
    size,
    il,
    xl,
    wx,
    wy )
```

Convert 3 arbitrary control points containing annotation- and world coords into 4 ordered corner points according to the Petrel Ordered Corner Points (OCP) definition, which corresponds to these bulk data indices:

```
(      0,      0)
(size[0] - 1,      0)
(      0, size[1] - 1)
(size[0] - 1, size[1] - 1)
```

See PetrelOrientationHandling

This is used to convert from a ThreePoint to a FourPoint definition. If the definition is already FourPoint then calling this function with the 3 first points should return the same result.

See OrderedCornerPoints.cpp.

TODO-Test, since this Python code was not ported from there (I used lattice.py in interpretation-sandbox instead), some extra testing is needed to verify the code.



### 6.11.2.2 generalTransform()

```
def openzgy.impl.transform.generalTransform (
    ax0,
    ay0,
    ax1,
    ay1,
    ax2,
    ay2,
    bx0,
    by0,
    bx1,
    by1,
    bx2,
    by2,
    data )
```

Coordinate conversion based on 3 arbitrary control points in both the source and the target system, i.e. no need to calculate the transformation matrix first.

This is my favorite code snippet for coordinate conversion. Almost no ambiguity about what it does and how to use it.

The data array is converted in place. The code tries to ensure that the output is floating point even though the input might not be. This is to prevent surprises if converting the result to a numpy type (where types are more strict) and forgetting to give an explicit type.

See iltf2d.cpp.cpp.

TODO-Test, since this Python code was not ported from there (I used lattice.py in interpretation-sandbox instead), some extra testing is needed to verify the code.

## 6.12 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, cropsizes=None, cropoffset=None, lod=0, sizeonly=False, progress=None)

### 6.12.1 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.
```

## 6.12.2 Function Documentation

### 6.12.2.1 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 bricksize 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.

## 6.13 openzgy.zgypublic Namespace Reference

Consolidate old and new Python API.

### Classes

- class [ZgyReader](#)
- class [ZgyWriter](#)

#### 6.13.1 Detailed Description

Consolidate old and new Python API.

Consolidate old and new Python API.

This file is for testing only, when comparing the old and new apis.  
The Python wrapper for the old ZGY-Public API is not quite identical  
to the new OpenZGY api. This file wraps the old API to become more  
similar to the new.

Application code should NOT use this package when migrating. Instead,  
please modify the code to fit the slightly changed API. The documentation  
for this package might help to show which changes are required.



# Alpha version. Expect changes.

## Chapter 7

## Class Documentation

### 7.1 openzgy.api.\_internal Class Reference

Explicit control of imported symbols.

#### Static Public Attributes

- `enum`
- `ZgyInternalMeta`
- `ZgyInternalBulk`
- `ScalarBuffer`
- `generalTransform`
- `FileFactory`
- `StatisticData`
- `DecimationType`
- `CompressFactoryImpl`
- `ZfpCompressPlugin`
- `GenLodC`

#### 7.1.1 Detailed Description

Explicit control of imported symbols.

This class is only used to rename some internal code.

I want to be explicit about which classes I need from `impl.*` but at the same time I don't want to pollute the user-visible `api` namespace with names not even starting with an underscore that the user has no business accessing. I am not sure whether my way of achieving this is considered a kludge.

What I really want is to say something like:

```
from .impl.genlod import GenLodC as _internal.GenLodC
```

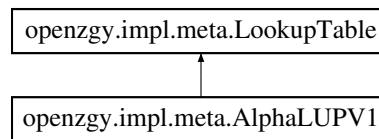
but apparently this is not allowed.

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

- `openzgy/api.py`

## 7.2 openzgy.impl.meta.AlphaLUPV1 Class Reference

Inheritance diagram for openzgy.impl.meta.AlphaLUPV1:



### Public Member Functions

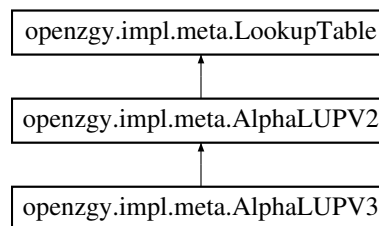
- `def __init__(self, buf, lupsizes)`

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

- `openzgy/impl/meta.py`

## 7.3 openzgy.impl.meta.AlphaLUPV2 Class Reference

Inheritance diagram for openzgy.impl.meta.AlphaLUPV2:



### Public Member Functions

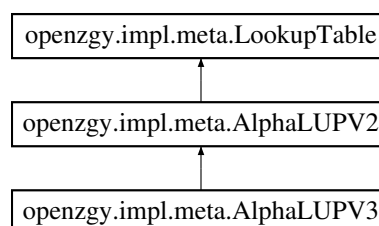
- `def __init__(self, buf, lupsizes)`

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

- `openzgy/impl/meta.py`

## 7.4 openzgy.impl.meta.AlphaLUPV3 Class Reference

Inheritance diagram for openzgy.impl.meta.AlphaLUPV3:



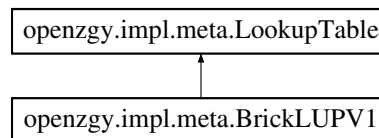
### Additional Inherited Members

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

- openzgy/impl/meta.py

## 7.5 openzgy.impl.meta.BrickLUPV1 Class Reference

Inheritance diagram for openzgy.impl.meta.BrickLUPV1:



### Public Member Functions

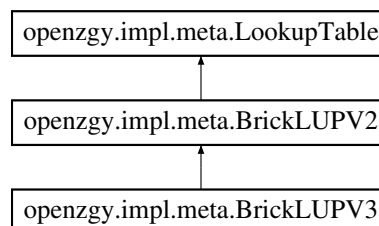
- `def __init__(self, buf, lupsized)`

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

- openzgy/impl/meta.py

## 7.6 openzgy.impl.meta.BrickLUPV2 Class Reference

Inheritance diagram for openzgy.impl.meta.BrickLUPV2:



### Public Member Functions

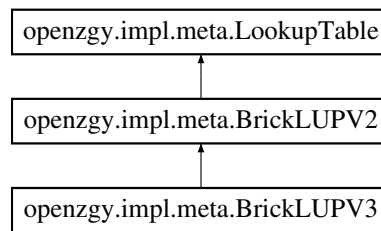
- `def __init__(self, buf, lupsized)`

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

- openzgy/impl/meta.py

## 7.7 openzgy.impl.meta.BrickLUPV3 Class Reference

Inheritance diagram for openzgy.impl.meta.BrickLUPV3:



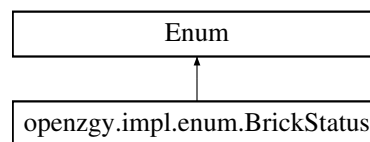
### Additional Inherited Members

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

- openzgy/impl/meta.py

## 7.8 openzgy.impl.enum.BrickStatus Class Reference

Inheritance diagram for openzgy.impl.enum.BrickStatus:



### Static Public Attributes

- int **Missing** = 0
- int **Constant** = 1
- int **Normal** = 2
- int **Compressed** = 3

### 7.8.1 Detailed Description

Brick status as used in the internal API only.

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

- openzgy/impl/enum.py



## 7.9 openzgy.impl.compress.CompressFactoryImpl Class Reference

### Public Member Functions

- def [knownCompressors](#) ()
- def [knownDecompressors](#) ()

### Static Public Member Functions

- def [registerCompressor](#) (name, fn)
- def [registerDecompressor](#) (name, fn)
- def **factory** (name, \*args, \*\*kwargs)
- def [decompress](#) (cdata, status, shape, file\_dtype, user\_dtype)

### 7.9.1 Detailed Description

Registry of known compress and decompress algorithms.  
Those two are completely separate but we might as well  
handle both in the same class.

### 7.9.2 Member Function Documentation

#### 7.9.2.1 [decompress\(\)](#)

```
def openzgy.impl.compress.CompressFactoryImpl.decompress (
    cdata,
    status,
    shape,
    file_dtype,
    user_dtype ) [static]
```

Loop over all registered decompressors and try to find one that  
can handle this particular brick. Raises an error if none found.  
See [CompressPlugin.decompress\(\)](#) for parameter descriptions.

#### 7.9.2.2 [knownCompressors\(\)](#)

```
def openzgy.impl.compress.CompressFactoryImpl.knownCompressors ( )
```

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.

### 7.9.2.3 knownDecompressors()

```
def openzgy.impl.compress.CompressFactoryImpl.knownDecompressors ( )
```

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

### 7.9.2.4 registerCompressor()

```
def openzgy.impl.compress.CompressFactoryImpl.registerCompressor (
    name,
    fn ) [static]
```

Register a factory function that will be called to create  
a function that in turn can be used to compress a data block.  
Pass `fn = None` if you for some reason need to remove a registration.

The registered function can have any signature; the signature  
needs to include whatever parameters the actual compressor wants.

The function that is created by the factory must have the signature:

```
raw:          bytes.
brickstatus:  impl.enum.BrickStatus,
bricksize:    tuple(int,int,int),
file_dtype:   np.dtype,
result_dtype: np.dtype
```

The function's return value:

```
np.ndarray with rank 3, shape bricksize, and dtype result_dtype.
```

Example usage of the factory:

```
old: with ZgyWriter(snr=snr)
new: with ZgyWriter(compressor = ZgyCompressFactory("ZFP", snr=30),
```

The example shows that this is a bit more inconvenient for the end  
user. But it allows for using different compression plug-ins with  
arbitrary parameters.

Note that user code doesn't need to use `ZgyCompressFactory()` and  
its list of known compressors. Instead a compression function  
can be provided directly. But most likely the factory method will  
be simpler to maintain.

`fn()` is allowed to return `None`, which will have the same effect  
as if the user did not specify any compression. E.g. there might be  
a convention that `snr<0` means store uncompressed. Allowing the  
factory to return `None` in that case means the example above would  
still work. Otherwise the `snr<0` test would be included in the  
client code. Which is messy.

### 7.9.2.5 registerDecompressor()

```
def openzgy.impl.compress.CompressFactoryImpl.registerDecompressor (
    name,
    fn ) [static]
```

Register a factory function that is able to decompress one or more types of compressed data. The registered functions will be called in reverse order of registration until one of them indicates that it has decompressed the data. You cannot remove a registration but you can effectively disable it by registering another one that recognizes the same input data. The supplied name is only for information.

The function that is created by the factory must have the signature:

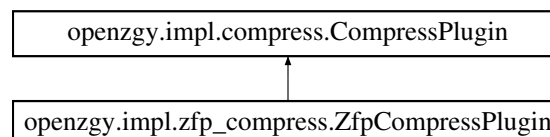
```
raw:          bytes.
brickstatus:  impl.enum.BrickStatus,
bricksize:    tuple(int,int,int),
file_dtype:   np.dtype,
result_dtype: np.dtype
```

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

- openzgy/impl/compress.py

## 7.10 openzgy.impl.compress.CompressPlugin Class Reference

Inheritance diagram for openzgy.impl.compress.CompressPlugin:



### Public Member Functions

- def `__init__` (self, \*args, \*\*kwargs)
- def `__call__` (self, data)
- def `dump` (\*args, \*\*kwargs)

### Static Public Member Functions

- def `compress` (data, \*args, \*\*kwargs)
- def `decompress` (cdata, status, shape, file\_dtype, user\_dtype)

## 7.10.1 Detailed Description

Base class for OpenZGY compression plug-ins.

If anybody wants to add additional compression algorithms it is recommended but not required to use this base class. See `CompressFactoryImpl.register{Compressor,Decompressor}` for how to use plain functors (C++) or callables (Python) instead.

This class performs triple duty as it handles both compression and decompression static methods (need not have been together) and an instance of the class can be used a compressor functor if a lambda is too limiting. To invoke the methods:

```
MyCompressPlugin.factory(...) (data)
MyCompressPlugin.compress(data, ...) (NOT recommended)
MyCompressPlugin.decompress(cdata,status,shape,file_dtype,user_dtype)
```

The following will also work but should only be used for very simple compressors that have no parameters. In the first case `MyCompressPlugin` won't have the option to return `None` for certain parameters, and in the second case handling a variable argument list becomes trickier.

To register this class:

```
CompressFactoryImpl.registerCompressor("My",MyCompressPlugin.factory)
CompressFactoryImpl.registerDecompressor("My",MyCompressPlugin.decompress)
```

To use the compression part from client code:

```
compressor = ZgyCompressFactory("My", ...)
```

## 7.10.2 Constructor & Destructor Documentation

### 7.10.2.1 `__init__()`

```
def openzgy.impl.compress.CompressPlugin.__init__ (
    self,
    * args,
    ** kwargs )
```

Create an instance that remembers the arguments it was created with. When the instance is called as a function it will invoke `compress()` with those arguments. So you can use either of the following:

```
compressor = CompressPlugin.compress # no arguments
compressor = CompressPlugin(...)
compressor = lambda x: CompressPlugin.compress(x, ...)
```

Derived classes don't need to redefine `__init__` and `__call__`. But they might want to in order to get argument checking. The `__init__` in the base class accepts any arguments so an error won't be caught until the first time the compressor is invoked.

## 7.10.3 Member Function Documentation

### 7.10.3.1 `__call__()`

```
def openzgy.impl.compress.CompressPlugin.__call__ (
    self,
    data )
```

Invoke the compressor with arguments passed by the constructor.

Reimplemented in [openzgy.impl.zfp\\_compress.ZfpCompressPlugin](#).

### 7.10.3.2 `compress()`

```
def openzgy.impl.compress.CompressPlugin.compress (
    data,
    * args,
    ** kwargs ) [static]
```

This is an abstract method.

Compress a 3d or (TODO-Low 2d) numpy array, returning a bytes-like result. If called with a single "data" argument the compression will be done with default parameters and no extended logging.

Additional arguments are specific to the compression type.

The function can be used directly as the compression hook. But you probably want a lambda expression or a real instance of this class instead, to be able to specify parameters.

The compression algorithm is used is assumed to handle big / little endian conversion itself. TODO-Worry this is not quite true for ZFP. See the documentation. A special compilation flag is needed on big endian machines. Also I suspect the optional hedaer (which this code uses) might need byte swapping.

### 7.10.3.3 `decompress()`

```
def openzgy.impl.compress.CompressPlugin.decompress (
    cdata,
    status,
    shape,
    file_dtype,
    user_dtype ) [static]
```

This is an abstract method.

Decompress bytes or similar into a `numpy.ndarray`.

Arguments:

```
cdata      -- bytes or bytes-like compressed data,
              possibly with trailing garbage.
status     -- Currently always BrickStatus.Compressed,
              in the future the status might be used to
              distinguish between different compression
              algorithms instead of relying on magic numbers.
shape      -- Rank and size of the result in case this is
              not encoded by the compression algorithm.
file_dtype -- Original value type before compression,
              in case the decompressor cannot figure it out.
              This will exactly match the dtype of the
              data buffer passed to the compressor.
user_dtype -- Required value type of returned array.
```

Passing an uncompressed brick to this function is an error.  
We don't have enough context to handle uncompressed bricks  
that might require byteswapping and fix for legacy quirks.  
Also cannot handle constant bricks, missing bricks, etc.

The reason `user_dtype` is needed is to avoid additional  
quantization noise when the user requests integer compressed data  
to be read as float. the decompressor might need to convert  
float data to int, only to have it converted back to float later.

Current assumptions made of all candidate algorithms:

- The compressed data stream may have trailing garbage;  
this will be silently ignored by the decompressor.
- The compressed data stream will never be longer than  
the uncompressed data. This needs to be enforced by  
the compressor. The compressor is allowed to give up  
and tell the caller to not compress this brick.
- The reason for the two assumptions above is an  
implementation detail; the reported size of a  
compressed brick is not completely reliable.  
This might change in the next version
- The compressed data stream must start with a magic  
number so the decompressor can figure out whether  
this is the correct algorithm to use.

If the assumptions cannot be met, the compressor / decompressor  
for this particular type could be modified to add an extra header  
with the compressed size and a magic number. Or we might add a  
(size, algorithm number) header to every compressed block to  
relieve the specific compressor / decompressor from worrying  
about this. Or the brick status could be used to encode which  
algorithm was used, picked up from the MSB of the lup entry.  
Which would also require the compressor to return both the  
actual compressed data and the code to identify the decompressor.  
That is the main reason we are also passed the "status" arg.  
Caveat: If adding an extra header, keep in mind that this header  
must be included when checking that the compressed stream is not  
too big.

Reimplemented in [openzgy.impl.zfp\\_compress.ZfpCompressPlugin](#).

### 7.10.3.4 dump()

```
def openzgy.impl.compress.CompressPlugin.dump (
    * args,
    ** kwargs )
```

Output statistics to standard output, if possible.

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

- openzgy/impl/compress.py

## 7.11 openzgy.impl.compress.CompressStats Class Reference

### Public Member Functions

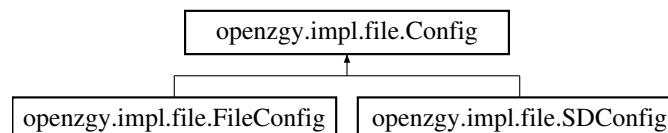
- def **\_\_init\_\_** (self, details)
- def **add\_data** (self, idata, csize, ddata, \*ctime=None, dtime=None, msg=None)
- def **add** (self, signal, noise, snr, isize, csize, \*ctime=None, dtime=None, msg=None)
- def **snr** (self)
- def **dump** (self, msg=None, \*outfile=None, text=True, csv=False)
- def **empty** (self)

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

- openzgy/impl/compress.py

## 7.12 openzgy.impl.file.Config Class Reference

Inheritance diagram for openzgy.impl.file.Config:



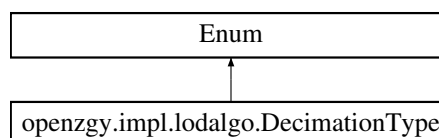
The documentation for this class was generated from the following file:

- openzgy/impl/file.py

## 7.13 openzgy.impl.lodalgo.DecimationType Class Reference

Possible algorithms to generate LOD bricks.

Inheritance diagram for openzgy.impl.lodalgo.DecimationType:



## Static Public Attributes

- int **LowPass** = 0,
- int **WeightedAverage** = 1,
- int **Average** = 2,
- int **Median** = 3,
- int **Minimum** = 4,
- int **Maximum** = 5,
- int **MinMax** = 6,
- int **Decimate** = 7,
- int **DecimateSkipNaN** = 8,
- int **DecimateRandom** = 9,
- int **AllZero** = 10,
- int **WhiteNoise** = 11,
- int **MostFrequent** = 12,
- int **MostFrequentNon0** = 13,
- int **AverageNon0** = 14,

### 7.13.1 Detailed Description

Possible algorithms to generate LOD bricks.

```
Possible algorithms to generate LOD bricks.
We might trim this list later to what is actually in use.
The "classic" ZGY only uses the first two.
```

```
CAVEAT: Many of these might be expensive to port and/or not
possible to implement efficiently in Python.
```

```
TODO-Low: Avoid exposing this enum to the public API.
```

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

- `openzgy/impl/lodalgo.py`

## 7.14 openzgy.impl.bulk.ErrorsWillCorruptFile Class Reference

### Public Member Functions

- `def __init__(self, parent)`
- `def __enter__(self)`
- `def __exit__(self, type, value, traceback)`



### 7.14.1 Detailed Description

Duplicated between impl.bulk and impl.meta. Maybe fix sometime. Technically the two differ because they set different flags. In ZgyInternalBulk and ZgyInternalMeta respectively. But that distinction isn't easy to see in Python.

Start a critical section where any exception means that the owner class should be permanently flagged with `_is_bad = True`. Typically this is used to prevent secondary errors after a write failure that has most likely corrupted the entire file. The exception itself will not be caught.

The `_is_bad` flag normally means that any further attempts to access this class, at least for writing, will raise a `ZgyCorruptedFile` exception. Regardless of what the exception was that caused the flag to be set.

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

- `openzgy/impl/bulk.py`

## 7.15 openzgy.impl.meta.ErrorsWillCorruptFile Class Reference

### Public Member Functions

- `def __init__(self, parent)`
- `def __enter__(self)`
- `def __exit__(self, type, value, traceback)`

### 7.15.1 Detailed Description

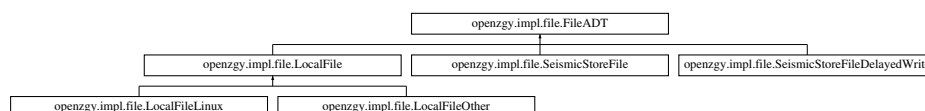
Duplicated between impl.bulk and impl.meta. Maybe fix sometime.

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

- `openzgy/impl/meta.py`

## 7.16 openzgy.impl.file.FileADT Class Reference

Inheritance diagram for `openzgy.impl.file.FileADT`:



## Public Member Functions

- `def __init__` (self, filename, mode, iocontext)
- `def __enter__` (self)
- `def __exit__` (self, type, value, traceback)
- `def xx_close` (self)
- `def xx_read` (self, offset, size, \*usagehint=UsageHint.Unknown)
- `def xx_write` (self, data, offset, \*usagehint=UsageHint.Unknown)
- `def xx_readv` (self, requests, \*parallel\_ok=False, immutable\_ok=False, transient\_ok=False, usagehint=UsageHint.Unknown)
- `def threadsafe` (self)
- `def xx_iscloud` (self)

## 7.16.1 Constructor & Destructor Documentation

### 7.16.1.1 \_\_init\_\_()

```
def openzgy.impl.file.FileADT.__init__ (
    self,
    filename,
    mode,
    iocontext )
```

Open a file in the specified mode, which must be "rb" or "w+b".  
Caller should use a "with" block to ensure the file gets closed.  
The iocontext is an optional data structure that the user may  
specify when a reader is created. It might be used to hold  
user credentials etc. needed to access the low level file.  
TODO-Low: support "r+b" (update) at some point in the future.

Reimplemented in [openzgy.impl.file.SeismicStoreFileDelayedWrite](#), [openzgy.impl.file.SeismicStoreFile](#), and [openzgy.impl.file.LocalFile](#).

## 7.16.2 Member Function Documentation

### 7.16.2.1 xx\_close()

```
def openzgy.impl.file.FileADT.xx_close (
    self )
```

Close a previously opened file.  
No action if the file is already closed.

Reimplemented in [openzgy.impl.file.SeismicStoreFileDelayedWrite](#), [openzgy.impl.file.SeismicStoreFile](#), and [openzgy.impl.file.LocalFile](#).

### 7.16.2.2 `xx_read()`

```
def openzgy.impl.file.FileADT.xx_read (
    self,
    offset,
    size,
    * usagehint = UsageHint.Unknown )
```

Read binary data from the file. Both size and offset are mandatory. I.e. caller is not allowed to read "the entire file", and not allowed to "read from where I left off the last time". The actual reading will be done in a derived class. The base class only validates the arguments.

Reimplemented in [openzgy.impl.file.SeismicStoreFileDelayedWrite](#), [openzgy.impl.file.LocalFileLinux](#), [openzgy.impl.file.LocalFileOther](#) and [openzgy.impl.file.SeismicStoreFile](#).

### 7.16.2.3 `xx_readv()`

```
def openzgy.impl.file.FileADT.xx_readv (
    self,
    requests,
    * parallel_ok = False,
    immutable_ok = False,
    transient_ok = False,
    usagehint = UsageHint.Unknown )
```

Read binary data from multiple regions in the file. Each part of the request specifies offset, size, and a delivery functor which will be invoked to pass back the returned bulk.

Arguments:

- `parallel_ok`: If true then the delivery functor might be called simultaneously from multiple worker threads. The function itself will block until all the data has been read or an error occurs.
- `immutable_ok`: If true the caller promises that the delivery functor will not try to modify the data buffer. Pass False e.g. if the functor may need to byteswap the data it has read from file.
- `transient_ok`: If true the caller promises that the delivery functor will not keep a reference to the data buffer after the functor returns.

The delivery functor is called as  
`fn(data)`

FUTURE: a new argument `partial_ok` may be set to True if it is ok to call the delivery functor with less data than requested, and to keep calling it until all data has been delivered. The signature of the delivery functor gets changed to `fn(data, offset, size)`. Offset is the absolute file offset. I.e. not relative to the requested offset. Passing `partial_ok=True` might elide some buffer copies if the caller is doing something simple (such as reading an uncompressed brick) where partial copies are possible, and the backend is in the cloud, and a longer lived cache is being maintained, and the cache block size is smaller than the requested size. That is a lot of ifs. There was some code to handle `partial_ok` but it has been removed. Get it from the git history if you really want it.

Reimplemented in [openzgy.impl.file.SeismicStoreFile](#), [openzgy.impl.file.LocalFileLinux](#), and [openzgy.impl.file.LocalFileOther](#).

## 7.16.2.4 xx\_write()

```
def openzgy.impl.file.FileADT.xx_write (
    self,
    data,
    offset,
    * usagehint = UsageHint.Unknown )
```

Write binary data to the file. Offset is mandatory. I.e. caller is not allowed to "write to where I left off the last time". The actual writing will be done in a derived class. The base class only validates the arguments.

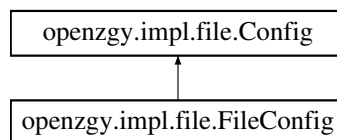
Reimplemented in [openzgy.impl.file.SeismicStoreFileDelayedWrite](#), [openzgy.impl.file.SeismicStoreFile](#), [openzgy.impl.file.LocalFileLinux](#) and [openzgy.impl.file.LocalFileOther](#).

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

- [openzgy/impl/file.py](#)

## 7.17 openzgy.impl.file.FileConfig Class Reference

Inheritance diagram for openzgy.impl.file.FileConfig:



### Public Member Functions

- def [\\_\\_init\\_\\_](#) (self, context)
- def **dump** (self)

### Public Attributes

- **maxsize**
- **maxhole**
- **aligned**
- **segsz**
- **threads**

### 7.17.1 Constructor & Destructor Documentation

### 7.17.1.1 `__init__()`

```
def openzgy.impl.file.FileConfig.__init__ (
    self,
    context )
```

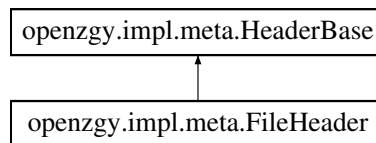
Currently the on-prem file reader has no user settable configuration.

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

- `openzgy/impl/file.py`

## 7.18 openzgy.impl.meta.FileHeader Class Reference

Inheritance diagram for `openzgy.impl.meta.FileHeader`:



### Public Member Functions

- `def __init__(self, buf=None)`

### 7.18.1 Detailed Description

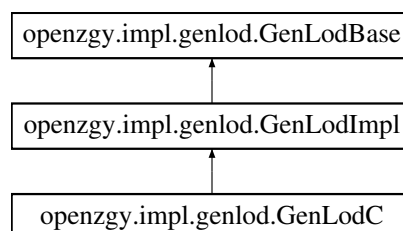
Unpack a byte buffer into a new instance of this header.  
Caller is responsible for all I/O, so we don't need an `iocontext`.

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

- `openzgy/impl/meta.py`

## 7.19 openzgy.impl.genlod.GenLodBase Class Reference

Inheritance diagram for `openzgy.impl.genlod.GenLodBase`:



## Public Member Functions

- `def __init__(self, size, *bricksize=(64, 64, 64), dtype=np.float32, range_hint=None, nlods=None, decimation=None, histogram=None, defaultvalue=None, progress=None, verbose=None)`

### 7.19.1 Detailed Description

Abstract class for generating low resolution bricks, histogram, and statistics. At this level only define virtual methods for I/O. The implementation can be used as-is when mocking the class. The optional `nlods` parameter is only used as a consistency check.

Note that the `WeightedAverage` algorithm requires a histogram to work. If no histogram was provided then the current contents of the accumulated histogram will be used. This is unfortunate and might cause brick artifacts. Especially in the first few bricks that are generated. With a non-recursive algorithm (plan C) and with only `lod2` and above uses weighted average then this is unproblematic. Because in that case we will be done with the histogram once we need it. TODO-Low consider doing an initial pass with a statistical sampling of the `lod0` data, only for use with weighted average. There will be a minor issue with some values appearing to have zero frequency, but this should not cause any trouble. (assume "1").

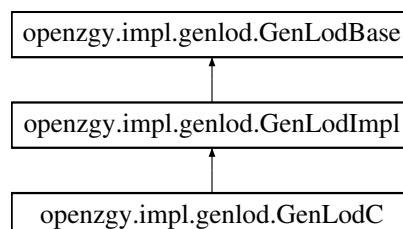
Note that the `WeightedAverage` and `AverageNon0` algorithms expect a `defaultvalue` to use when all inputs are `inf/nan` or (for `AverageNon0`) zero. Only relevant for integral types, to ensure that the default is whatever will produce the value closest to 0 after conversion. And integral data can neither be `inf` nor `nan`, so this is a pretty academic issue. For `AverageNon0` that algorithm is currently not used. So it isn't even clear what the desired behavior should be.

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

- `openzgy/impl/genlod.py`

## 7.20 openzgy.impl.genlod.GenLodC Class Reference

Inheritance diagram for `openzgy.impl.genlod.GenLodC`:



## Public Member Functions

- `def __init__(self, accessor, *compressor=None, decimation=None, progress=None, verbose=None)`

### 7.20.1 Detailed Description

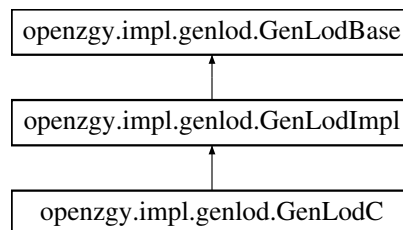
Generate and store low resolution bricks, histogram, and statistics. See doc/lowres.html for details. I/O is done via ZgyInternalBulk. Use this class as part as finalize(). The implementation uses plan C, which means the full resolution data will be read from the ZGY file. To implement plan D, make a derived class that redefines `_read()` to query the client for the required full resolution data. `_read()` must then also call `_write()` to store the data it just received.

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

- openzgy/impl/genlod.py

## 7.21 openzgy.impl.genlod.GenLodImpl Class Reference

Inheritance diagram for openzgy.impl.genlod.GenLodImpl:



### Public Member Functions

- def `__init__`(self, \*args, \*\*kwargs)
- def `__call__`(self)

### 7.21.1 Detailed Description

Abstract class for generating low resolution bricks, histogram, and statistics. The inherited methods for I/O are still stubs. See doc/lowres.html for details. This class implements plan C or D which is good for compressed data and acceptable for uncompressed. The ordering of low resolution bricks in the file will not be optimal. For optimal ordering but working only for uncompressed data consider implementing plan B in addition to the plan C already implemented. The implementation can be used as-is in a unit test with mocked I/O.

### 7.21.2 Member Function Documentation

## 7.21.2.1 `__call__()`

```
def openzgy.impl.genlod.GenLodImpl.__call__ (
    self )
```

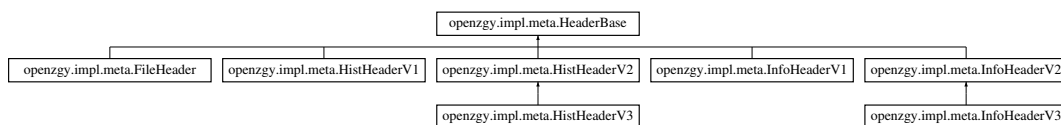
Generate and store statistics, histogram, and all low resolution bricks. Works for plans C and D. If we also need an implementation of plan B then this method would need to iterate over all bricks and lods, and `_accumulate` would not make any recursive calls.

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

- `openzgy/impl/genlod.py`

## 7.22 `openzgy.impl.meta.HeaderBase` Class Reference

Inheritance diagram for `openzgy.impl.meta.HeaderBase`:



### Public Member Functions

- `def headersize (cls)`
- `def checkformats (cls, verbose=False, *file=None)`
- `def pack (self)`
- `def unpack (self, buf=None)`
- `def read (cls, f, offset)`
- `def dump (self, prefix="", file=None)`

### 7.22.1 Detailed Description

Convenience base class for implementing classes that map 1:1 to a specific header with a specific version that exists in a ZGY file. The constructor should unpack a supplied byte buffer into a new instance of this header. Caller is responsible for all I/O, so no methods in this class need an `iocontext` except `read()`. The various headers don't need to inherit this base class if they don't want to.

### 7.22.2 Member Function Documentation



### 7.22.2.1 checkformats()

```
def openzgy.impl.meta.HeaderBase.checkformats (
    cls,
    verbose = False,
    * file = None )
```

Helper to compare the python definition of the header layout with the C++ version. Also check that the same attribute isn't listed twice.

### 7.22.2.2 dump()

```
def openzgy.impl.meta.HeaderBase.dump (
    self,
    prefix = "",
    file = None )
```

Print the entire contents of the object, including derived fields.

### 7.22.2.3 headersize()

```
def openzgy.impl.meta.HeaderBase.headersize (
    cls )
```

Return the size this header has on disk.

### 7.22.2.4 pack()

```
def openzgy.impl.meta.HeaderBase.pack (
    self )
```

Convert the contents of this class to a byte array suitable for storing in the ZGY file.

## 7.22.2.5 read()

```
def openzgy.impl.meta.HeaderBase.read (
    cls,
    f,
    offset )
```

Read the header from disk and parse it, returning a new instance.

## 7.22.2.6 unpack()

```
def openzgy.impl.meta.HeaderBase.unpack (
    self,
    buf = None )
```

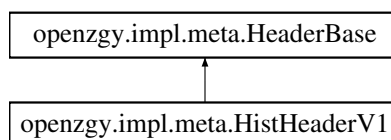
Convert a byte array as read from the ZGY file into a Python object. Normally a call to `unpack()` will immediately be followed by a call to `calculate()` to fill in any derived information, convert enums, etc. If `buf` is `None`, unpacking is done on an all zero buffer. This ensures that all data fields are present in the object. Simplifying things if the application is creating an instance from scratch.

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

- `openzgy/impl/meta.py`

## 7.23 openzgy.impl.meta.HistHeaderV1 Class Reference

Inheritance diagram for `openzgy.impl.meta.HistHeaderV1`:



### Public Member Functions

- `def \_\_init\_\_ (self, buf=None)`

### 7.23.1 Constructor & Destructor Documentation

### 7.23.1.1 `__init__()`

```
def openzgy.impl.meta.HistHeaderV1.__init__ (
    self,
    buf = None )
```

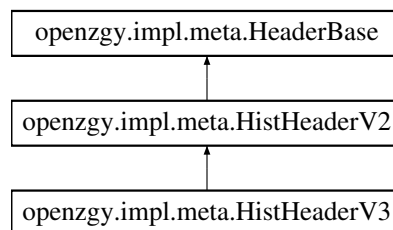
Unpack a byte buffer into a new instance of this header.  
Caller is responsible for all I/O, so we don't need an iocontext.

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

- openzgy/impl/meta.py

## 7.24 openzgy.impl.meta.HistHeaderV2 Class Reference

Inheritance diagram for openzgy.impl.meta.HistHeaderV2:



### Public Member Functions

- def `__init__` (self, buf=None)

### 7.24.1 Constructor & Destructor Documentation

#### 7.24.1.1 `__init__()`

```
def openzgy.impl.meta.HistHeaderV2.__init__ (
    self,
    buf = None )
```

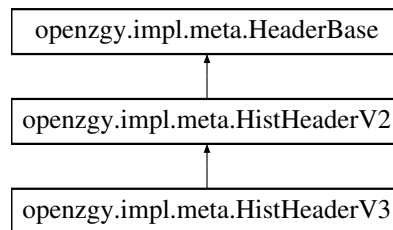
Unpack a byte buffer into a new instance of this header.  
Caller is responsible for all I/O, so we don't need an iocontext.

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

- openzgy/impl/meta.py

## 7.25 openzgy.impl.meta.HistHeaderV3 Class Reference

Inheritance diagram for openzgy.impl.meta.HistHeaderV3:



### Additional Inherited Members

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

- openzgy/impl/meta.py

## 7.26 openzgy.impl.histogram.HistogramData Class Reference

### Public Member Functions

- def **\_\_init\_\_** (self, range\_hint=None, dtype=np.float32)
- def **add** (self, data, factor=1)
- def **scale** (self, a, b)
- def **resize** (self, newsize)
- def **bins** (self)
- def **vv\_range** (self)
- def **np\_range** (self)
- def **binvalue** (self, bin\_number)

### 7.26.1 Member Function Documentation

#### 7.26.1.1 binvalue()

```
def openzgy.impl.histogram.HistogramData.binvalue (
    self,
    bin_number )
```

Convert a single bin number to the center value of this bin.  
Note that in ZGY this will refer to storage values, so you  
may need to explicitly convert the result.

### 7.26.1.2 np\_range()

```
def openzgy.impl.histogram.HistogramData.np_range (
    self )
```

Histogram range, numpy and salmon style, with numbers representing the edges of the first and last bin.

### 7.26.1.3 vv\_range()

```
def openzgy.impl.histogram.HistogramData.vv_range (
    self )
```

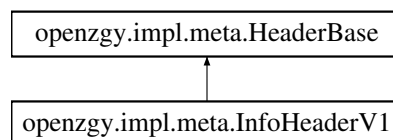
Histogram range, voxelvision and zgy style, with numbers representing the center value of the first and last bin.

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

- openzgy/impl/histogram.py

## 7.27 openzgy.impl.meta.InfoHeaderV1 Class Reference

Inheritance diagram for openzgy.impl.meta.InfoHeaderV1:



### Public Member Functions

- def [\\_\\_init\\_\\_](#) (self, buf=None)
- def **load** (self, buf)
- def **calculate** (self, sl=None, hh=None)

### 7.27.1 Constructor & Destructor Documentation

## 7.27.1.1 `__init__()`

```
def openzgy.impl.meta.InfoHeaderV1.__init__ (
    self,
    buf = None )
```

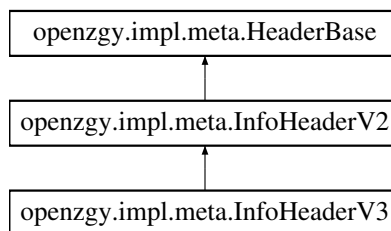
Unpack a byte buffer into a new instance of this header.  
Caller is responsible for all I/O, so we don't need an iocontext.

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

- `openzgy/impl/meta.py`

## 7.28 `openzgy.impl.meta.InfoHeaderV2` Class Reference

Inheritance diagram for `openzgy.impl.meta.InfoHeaderV2`:



### Public Member Functions

- `def \_\_init\_\_ (self, buf=None)`
- `def calculate (self, sl=None, hh=None)`
- `def calculate\_write (self)`

### 7.28.1 Constructor & Destructor Documentation

#### 7.28.1.1 `__init__()`

```
def openzgy.impl.meta.InfoHeaderV2.__init__ (
    self,
    buf = None )
```

Unpack a byte buffer into a new instance of this header.  
Caller is responsible for all I/O, so we don't need an iocontext.

### 7.28.2 Member Function Documentation

#### 7.28.2.1 calculate\_write()

```
def openzgy.impl.meta.InfoHeaderV2.calculate_write (
    self )
```

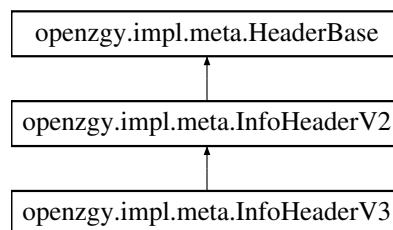
Call this when important parts of ths struct has changed,  
Note that on write it is we that compute slbufsize that gets copied  
to offsetheader and used when the string list is written.  
Not the other way around.

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

- openzgy/impl/meta.py

## 7.29 openzgy.impl.meta.InfoHeaderV3 Class Reference

Inheritance diagram for openzgy.impl.meta.InfoHeaderV3:



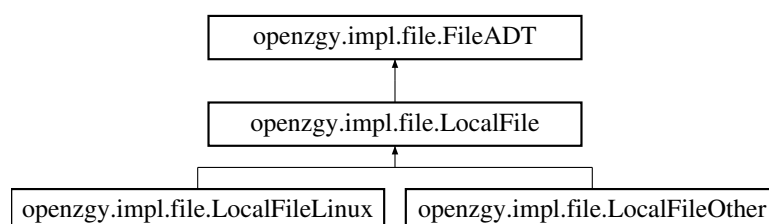
### Additional Inherited Members

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

- openzgy/impl/meta.py

## 7.30 openzgy.impl.file.LocalFile Class Reference

Inheritance diagram for openzgy.impl.file.LocalFile:



## Public Member Functions

- def `__init__` (self, filename, mode, iocontext)
- def `xx_eof` (self)
- def `xx_close` (self)

### 7.30.1 Constructor & Destructor Documentation

#### 7.30.1.1 `__init__()`

```
def openzgy.impl.file.LocalFile.__init__ (
    self,
    filename,
    mode,
    iocontext )
```

Open a file in the specified mode, which must be "rb" or "w+b".  
Caller should use a "with" block to ensure the file gets closed.  
The iocontext is an optional data structure that the user may  
specify when a reader is created. It might be used to hold  
user credentials etc. needed to access the low level file.  
TODO-Low: support "r+b" (update) at some point in the future.

Reimplemented from [openzgy.impl.file.FileADT](#).

### 7.30.2 Member Function Documentation

#### 7.30.2.1 `xx_close()`

```
def openzgy.impl.file.LocalFile.xx_close (
    self )
```

Close a previously opened file.  
No action if the file is already closed.

Reimplemented from [openzgy.impl.file.FileADT](#).

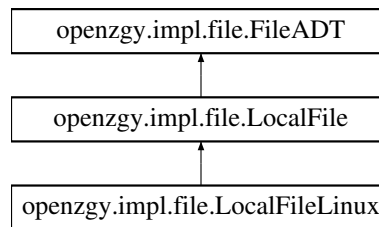
The documentation for this class was generated from the following file:

- `openzgy/impl/file.py`



### 7.31 openzgy.impl.file.LocalFileLinux Class Reference

Inheritance diagram for openzgy.impl.file.LocalFileLinux:



#### Public Member Functions

- def [xx\\_read](#) (self, offset, size, \*usagehint=UsageHint.Unknown)
- def [xx\\_readv](#) (self, requests, \*parallel\_ok=False, immutable\_ok=False, transient\_ok=False, usagehint=UsageHint.Unknown)
- def [xx\\_write](#) (self, data, offset, \*usagehint=UsageHint.Unknown)
- def [threadsafe](#) (self)

#### 7.31.1 Member Function Documentation

##### 7.31.1.1 [xx\\_read\(\)](#)

```
def openzgy.impl.file.LocalFileLinux.xx_read (
    self,
    offset,
    size,
    * usagehint = UsageHint.Unknown )
```

Read binary data from the file. Both size and offset are mandatory. I.e. caller is not allowed to read "the entire file", and not allowed to "read from where I left off the last time". The actual reading will be done in a derived class. The base class only validates the arguments.

Reimplemented from [openzgy.impl.file.FileADT](#).

## 7.31.1.2 xx\_readv()

```
def openzgy.impl.file.LocalFileLinux.xx_readv (
    self,
    requests,
    * parallel_ok = False,
    immutable_ok = False,
    transient_ok = False,
    usagehint = UsageHint.Unknown )
```

Read binary data from multiple regions in the file. Each part of the request specifies offset, size, and a delivery functor which will be invoked to pass back the returned bulk.

Arguments:

`parallel_ok`: If true then the delivery functor might be called simultaneously from multiple worker threads. The function itself will block until all the data has been read or an error occurs.

`immutable_ok`: If true the caller promises that the delivery functor will not try to modify the data buffer. Pass False e.g. if the functor may need to byteswap the data it has read from file.

`transient_ok`: If true the caller promises that the delivery functor will not keep a reference to the data buffer after the functor returns.

The delivery functor is called as  
`fn(data)`

FUTURE: a new argument `partial_ok` may be set to True if it is ok to call the delivery functor with less data than requested, and to keep calling it until all data has been delivered. The signature of the delivery functor gets changed to `fn(data, offset, size)`. Offset is the absolute file offset. I.e. not relative to the requested offset. Passing `partial_ok=True` might elide some buffer copies if the caller is doing something simple (such as reading an uncompressed brick) where partial copies are possible, and the backend is in the cloud, and a longer lived cache is being maintained, and the cache block size is smaller than the requested size. That is a lot of ifs. There was some code to handle `partial_ok` but it has been removed. Get it from the git history if you really want it.

Reimplemented from [openzgy.impl.file.FileADT](#).

## 7.31.1.3 xx\_write()

```
def openzgy.impl.file.LocalFileLinux.xx_write (
    self,
    data,
    offset,
    * usagehint = UsageHint.Unknown )
```

Write binary data to the file. Offset is mandatory. I.e. caller is not allowed to "write to where I left off the last time". The actual writing will be done in a derived class. The base class only validates the arguments.

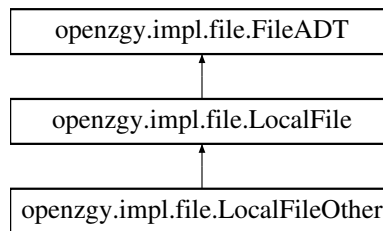
Reimplemented from [openzgy.impl.file.FileADT](#).

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

- `openzgy/impl/file.py`

### 7.32 openzgy.impl.file.LocalFileOther Class Reference

Inheritance diagram for openzgy.impl.file.LocalFileOther:



#### Public Member Functions

- def [xx\\_read](#) (self, offset, size, \*usagehint=UsageHint.Unknown)
- def [xx\\_readv](#) (self, requests, \*parallel\_ok=False, immutable\_ok=False, transient\_ok=False, usagehint=UsageHint.Unknown)
- def [xx\\_write](#) (self, data, offset, \*usagehint=UsageHint.Unknown)
- def [threadsafe](#) (self)

#### 7.32.1 Member Function Documentation

##### 7.32.1.1 [xx\\_read\(\)](#)

```
def openzgy.impl.file.LocalFileOther.xx_read (
    self,
    offset,
    size,
    * usagehint = UsageHint.Unknown )
```

Read binary data from the file. Both size and offset are mandatory. I.e. caller is not allowed to read "the entire file", and not allowed to "read from where I left off the last time". The actual reading will be done in a derived class. The base class only validates the arguments.

Reimplemented from [openzgy.impl.file.FileADT](#).

## 7.32.1.2 xx\_readv()

```
def openzgy.impl.file.LocalFileOther.xx_readv (
    self,
    requests,
    * parallel_ok = False,
    immutable_ok = False,
    transient_ok = False,
    usagehint = UsageHint.Unknown )
```

Read binary data from multiple regions in the file. Each part of the request specifies offset, size, and a delivery functor which will be invoked to pass back the returned bulk.

Arguments:

`parallel_ok`: If true then the delivery functor might be called simultaneously from multiple worker threads. The function itself will block until all the data has been read or an error occurs.

`immutable_ok`: If true the caller promises that the delivery functor will not try to modify the data buffer. Pass False e.g. if the functor may need to byteswap the data it has read from file.

`transient_ok`: If true the caller promises that the delivery functor will not keep a reference to the data buffer after the functor returns.

The delivery functor is called as

```
fn(data)
```

FUTURE: a new argument `partial_ok` may be set to True if it is ok to call the delivery functor with less data than requested, and to keep calling it until all data has been delivered. The signature of the delivery functor gets changed to `fn(data, offset, size)`. Offset is the absolute file offset. I.e. not relative to the requested offset. Passing `partial_ok=True` might elide some buffer copies if the caller is doing something simple (such as reading an uncompressed brick) where partial copies are possible, and the backend is in the cloud, and a longer lived cache is being maintained, and the cache block size is smaller than the requested size. That is a lot of ifs. There was some code to handle `partial_ok` but it has been removed. Get it from the git history if you really want it.

Reimplemented from [openzgy.impl.file.FileADT](#).

## 7.32.1.3 xx\_write()

```
def openzgy.impl.file.LocalFileOther.xx_write (
    self,
    data,
    offset,
    * usagehint = UsageHint.Unknown )
```

Write binary data to the file. Offset is mandatory. I.e. caller is not allowed to "write to where I left off the last time". The actual writing will be done in a derived class. The base class only validates the arguments.

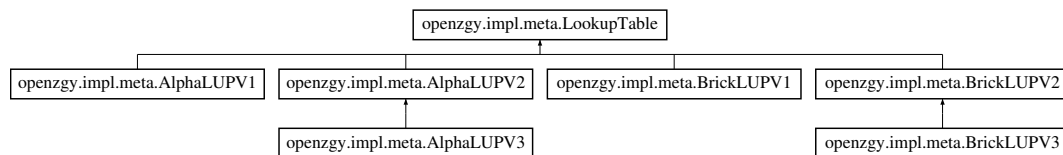
Reimplemented from [openzgy.impl.file.FileADT](#).

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

- [openzgy/impl/file.py](#)

### 7.33 openzgy.impl.meta.LookupTable Class Reference

Inheritance diagram for openzgy.impl.meta.LookupTable:



#### Public Member Functions

- `def \_\_init\_\_ (self, buf, lupsizes, mustflip)`
- `def pack (self)`
- `def read (cls, f, offset, lupsizes)`
- `def dump (self, prefix="", prefix0="", *file=None)`

#### 7.33.1 Detailed Description

Both the Alpha lookup table and the Brick lookup table hold a 64-bit file offset for each tile or brick in the file. Alpha tiles are bitmaps used to flag dead traces, and only have (i,j) coordinates. Bricks contain the actual samples and are indexed with (i, j, k).

The size of the lookup tables depend on the survey size. The first entry in the lookup table is for the brick or tile (always just one) holding the lowest resolution. This is immediately followed by one or more entries for the bricks or tiles at level of detail N-1, and so on until the entries for lod 0. Within one lod level the first entry is for the lowest numbered i,j,[k]. For subsequent entries the i numbers vary fastest and the k (or j in the alpha case) varies slowest. Note that this is somewhat non intuitive as it is the opposite of the ordering of samples within a tile.

In version 1 of the lookup tables the file offsets are stored in a somewhat quirky manner. The high 32 bits and the low 32 bits are both stored as little-endian integers, but the high part is stored first. So it is part big-endian, part little-endian.

An offset of 0 means the corresponding brick or tile does not exist. An offset of 1 means the brick or tile contains all zeros and does not take any space on the file. An offset with the most significant bit set also means the brick or tile has a constant value. In this case the actual value is encoded in the least significant 8/16/32 bits (depending on valuetype) of the stored offset. Offsets 0x8000000000000000 and 0x0000000000000001 are equivalent. Actually, v2 and later uses the first form while v1 used the second. For robustness both forms should be accepted regardless of version.

#### 7.33.2 Constructor & Destructor Documentation

## 7.33.2.1 `__init__()`

```
def openzgy.impl.meta.LookupTable.__init__ (
    self,
    buf,
    lupsiz,
    mustflip )
```

Unpack a byte buffer into a new instance of this header.  
Caller is responsible for all I/O, so we don't need an iocontext.

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

- openzgy/impl/meta.py

## 7.34 `openzgy.impl.meta.OffsetHeaderV1` Class Reference

### Public Member Functions

- def `__init__` (self)
- def `size` (self)
- def `load` (self, buf)
- def `calculate` (self, ih=None)
- def `read` (cls, f)
- def `dump` (self, \*file=None)

### 7.34.1 Constructor & Destructor Documentation

#### 7.34.1.1 `__init__()`

```
def openzgy.impl.meta.OffsetHeaderV1.__init__ (
    self )
```

Unpack a byte buffer into a new instance of this header.  
Caller is responsible for all I/O, so we don't need an iocontext.

#### 7.34.2 Member Function Documentation

### 7.34.2.1 calculate()

```
def openzgy.impl.meta.OffsetHeaderV1.calculate (
    self,
    ih = None )
```

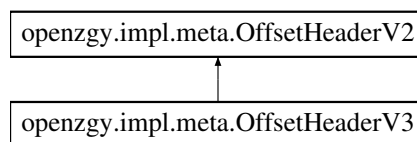
Offsets are stored explicitly, so this only needs to set size.

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

- openzgy/impl/meta.py

## 7.35 openzgy.impl.meta.OffsetHeaderV2 Class Reference

Inheritance diagram for openzgy.impl.meta.OffsetHeaderV2:



### Public Member Functions

- def `__init__` (self, buf=None)
- def **pack** (self)
- def **size** (self)
- def **read** (cls, f)
- def **dump** (self, \*file=None)
- def `calculate` (self, ih=None)

### 7.35.1 Constructor & Destructor Documentation

#### 7.35.1.1 `__init__`()

```
def openzgy.impl.meta.OffsetHeaderV2.__init__ (
    self,
    buf = None )
```

Unpack a byte buffer into a new instance of this header.  
Caller is responsible for all I/O, so we don't need an iocontext.

## 7.35.2 Member Function Documentation

### 7.35.2.1 calculate()

```
def openzgy.impl.meta.OffsetHeaderV2.calculate (
    self,
    ih = None )
```

Calculate offsets and sizes for the various headers and tables. Some information requires the InfoHeader to be already known. If it isn't we will just calculate as much as we can.

In general the size of a header as written to file might be larger than the size that the header expects to unpack. This allows adding more data fields at the end of the header. Older readers will just unpack the fields they know about.

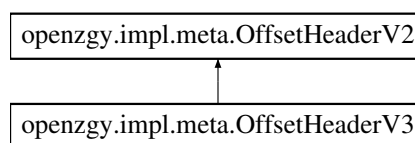
For ZGY V2 and V3 this is moot, as all the offsets are implicit with all the headers written sequentially. So the size needs to match exactly or the headers following this will be corrupt.

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

- openzgy/impl/meta.py

## 7.36 openzgy.impl.meta.OffsetHeaderV3 Class Reference

Inheritance diagram for openzgy.impl.meta.OffsetHeaderV3:



### Additional Inherited Members

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

- openzgy/impl/meta.py

## 7.37 openzgy.api.ProgressWithDots Class Reference

Simple progress bar.



### Public Member Functions

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

### 7.37.1 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".

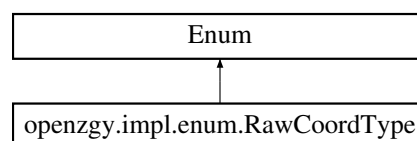
When passing a progress reporter to a function, make sure you do not pass the class itself. You need to create an instance of it.

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

- `openzgy/api.py`

## 7.38 openzgy.impl.enum.RawCoordType Class Reference

Inheritance diagram for `openzgy.impl.enum.RawCoordType`:



### Static Public Attributes

- `int Unknown = 0`
- `int Meters = 1`
- `int Feet = 2`
- `int ArcSec = 3`
- `int ArcDeg = 4`
- `int ArcDegMinSec = 5`

## 7.38.1 Detailed Description

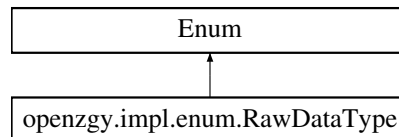
Coordinate type codes as stored in V1 files only.  
The values are stored on the file, so the numbers must not be changed.  
Source: BrickedFileVersion.cpp.  
There is no corresponding enum in the API layer.

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

- openzgy/impl/enum.py

## 7.39 openzgy.impl.enum.RawDataType Class Reference

Inheritance diagram for openzgy.impl.enum.RawDataType:



### Static Public Attributes

- int **SignedInt8** = 0
- int **UnsignedInt8** = 1
- int **SignedInt16** = 2
- int **UnsignedInt16** = 3
- int **SignedInt32** = 4
- int **UnsignedInt32** = 5
- int **Float32** = 6
- int **IbmFloat32** = 7

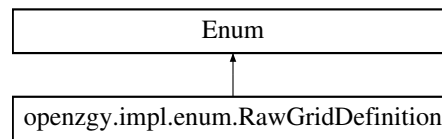
## 7.39.1 Detailed Description

Sample data type as stored on the file.  
In the public API this maps to SampleDataType.  
Source: BrickedFileVersion.cpp, MetaDataValue.h  
This enum is used for all versions.  
Note that the existing public ZGY library  
only recognizes SignedInt8, SignedInt16, and Float32.

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

- openzgy/impl/enum.py

Inheritance diagram for openzgy.impl.enum.RawGridDefinition:



### Static Public Attributes

- int **Unknown** = 0
- int **Parametric** = 1
- int **ThreePoint** = 2
- int **FourPoint** = 3

#### 7.40.1 Detailed Description

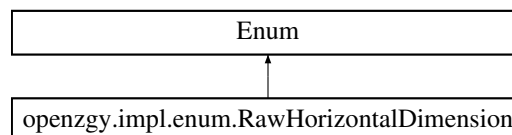
Method used to define the geometry. Only `FourPoint` is allowed for write, and only `ThreePoint` (treated as `FourPoint`) and `FourPoint` supported on read. The values are stored in the file, so the numbers must not be changed. There is no corresponding enum in the API layer.

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

- `openzgy/impl/enum.py`

## 7.41 openzgy.impl.enum.RawHorizontalDimension Class Reference

Inheritance diagram for openzgy.impl.enum.RawHorizontalDimension:



### Static Public Attributes

- int **Unknown** = 0
- int **Length** = 1
- int **ArcAngle** = 2

## 7.41.1 Detailed Description

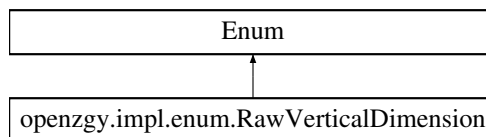
Horizontal dimension as seen in V2 files and later.  
In the public API this maps to `UnitDimension`.  
The values are stored in the file, so the numbers must not be changed.  
Source: `PoststackSeis3dInfo.h`, `MetaDataValue.h`, `ReaderImp::getMetaData`

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

- `openzgy/impl/enum.py`

## 7.42 openzgy.impl.enum.RawVerticalDimension Class Reference

Inheritance diagram for `openzgy.impl.enum.RawVerticalDimension`:



### Static Public Attributes

- `int Unknown = 0`
- `int Depth = 1`
- `int SeismicTWT = 2`
- `int SeismicOWT = 3`

## 7.42.1 Detailed Description

Vertical dimension as seen in V2 files and later.  
In the public API this maps to `UnitDimension`.  
The values are stored in the file, so the numbers must not be changed.  
Source: `PoststackSeis3dInfo.h`, `MetaDataValue.h`, `ReaderImp::getMetaData`

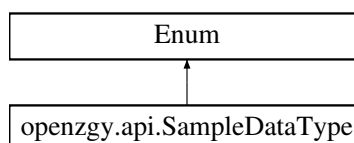
The documentation for this class was generated from the following file:

- `openzgy/impl/enum.py`

## 7.43 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

#### 7.43.1 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

## 7.44 openzgy.impl.bulk.ScalarBuffer Class Reference

### Public Member Functions

- def **\_\_init\_\_** (self, shape, value, dtype)
- def **shape** (self)
- def **dtype** (self)
- def **value** (self)
- def **itemsiz**e (self)
- def **\_\_len\_\_** (self)
- def **\_\_str\_\_** (self)
- def **\_\_repr\_\_** (self)
- def **inflate** (self)

#### 7.44.1 Detailed Description

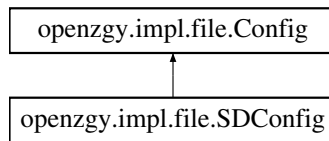
Represents an ndarray where all elements have the same value.  
This is basically a scalar but additionally has a `.shape` attribute telling how large an array it represents.

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

- openzgy/impl/bulk.py

## 7.45 openzgy.impl.file.SDConfig Class Reference

Inheritance diagram for openzgy.impl.file.SDConfig:



### Public Member Functions

- def `__init__` (self, context)
- def `extra` (self)
- def `dump` (self)

### Public Attributes

- `sdurl`
- `sdapikey`
- `sdtoken`
- `maxsize`
- `maxhole`
- `aligned`
- `segsize`
- `threads`
- `legaltag`
- `writeid`
- `seismicmeta`

### 7.45.1 Constructor & Destructor Documentation

#### 7.45.1.1 `__init__()`

```
def openzgy.impl.file.SDConfig.__init__ (
    self,
    context )
```

Process an iocontext for seismic store, doing consistency checks and applying fallbacks from environment variables and hard coded defaults.

The context itself should be a dict or equivalent. The code also supports the older style using a class instance with attributes. That feature is deprecated and will be removed. It violates the principle of least surprise.

A fully specified iocontext has the following attributes:

# Alpha version. Expect changes.

`sdurl: string`  
Where to contact the seismic store service.  
Defaults to `$OPENZGY_SDURL`.

`sdapikey: string`  
Authorization for application to access the seismic store API.  
Defaults to `$OPENZGY_SDAPIKEY`.

`sdtoken: string`  
User credentials. Set to `$OPENZGY_TOKEN` if not found, beware that this might not be secure. The code will no longer use the token last saved by `sdcfg` as a fallback. If this is desired you must specify `"FILE:carbon.slbapp.com"` as the token. Caveat: The `sdcfg` token is not refreshed so it might time out after an hour. Run `"sdtutil auth idtoken > /dev/null"` to refresh.

`maxsize: int` specified in MB between 0 and 1024.  
Zero is taken to mean do not consolidate.  
Tell the reader to try to consolidate neighboring bricks when reading from seismic store. This is usually possible when the application requests full traces or at least traces longer than 64 samples. Setting `maxsize` limits this consolidation to the specified size. The assumption is that for really large blocks the per-block overhead becomes insignificant compared to the transfer time.

Consolidating requests has higher priority than using multiple threads. So, capping `maxsize` might allow more data to be read in parallel.

Note that currently the spitting isn't really smart. With a 64 MB limit and 65 contiguous 1 MB buffers it might end up reading 64+1 MB instead of e.g. 32+33 MB.

Note that the low level reader should not assume that requests are capped at this size. They might be larger e.g. when reading the header information.

Defaults to `$OPENZGY_MAXSIZE_MB` if not specified, or 2 MB.

`maxhole: int` specified in MB between 0 and 1024.  
This applies when consolidate neighboring bricks when reading from seismic store. Setting `maxhole > 0` tells the reader that it is ok to also consolidate requests that are almost neighbors, with a gap up to and including `maxhole`. The data read from the gap will be discarded unless picked up by some (not yet implemented) cache.

For cloud access with high bandwidth (cloud-to-cloud) this should be at least 2 MB because smaller blocks will take just as long to read. For low bandwidth cloud access (cloud-to-on-prem) it should be less. If a fancy cache is implemented it should be more. For accessing on-prem ZGY files it probably makes no difference.  
Defaults to `$OPENZGY_MAXHOLE_MB` if not specified, or 2 MB.

`aligned: int` in MB between 0 and 1024.  
This is similar to the `maxhole` parameter. If set, starting and ending offsets are extended so they both align to the specified value. Set this parameter if the lower levels implement a cache with a fixed blocksize and when there is an assumption that most reads will be aligned anyway.  
TODO-Worry: Handling reads past EOF may become a challenge for the implementation.  
Defaults to `$OPENZGY_ALIGNED_MB` if not specified, or zero.

`segsz: int` in MB between 0 and 16\*1024 (i.e. 16 GB).  
Defaults to `$OPENZGY_SEGSIZE_MB` if not specified, or 1 GB.

`threads: int` between 1 and 1024.  
Use up to this many parallel requests to seismic store in order to speed up processing. This applies to individual reads in the main API. So the reads must be for a large

area (i.e. covering many bricks) for the setting to be of any use. Set to \$OPENZGY\_NUMTHREADS if not found, and 1 (i.e. no threading) if the environment setting is also missing.

Whether it is useful to set the variable depends on the application. Apps such as Petrel/BASE generally do their own multi threading, issuing multiple read requests to the high level API in parallel. In that case it might not be useful to also parallelize individual requests.

legaltag: string, possibly empty.  
The legaltag stored in the file. Used only on create.

writeid:  
I don't know what this is for. Ask the seismic store team.

seismicmeta:  
a dictionary of additional information to be associated with this dataset in the data ecosystem. Currently used only on create, although SDAPI allows this to be set on an existing file by calling {get,set}SeismicMeta().

When set via an environment variable (strictly for testing) this needs to be the string representation of the json data. When set from a program a Python dict is expected.

\_debug\_trace:  
For debugging and unit tests only.  
Callback to be invoked immediately before a read or write is passed on to seismic store. Typically used to verify that consolidating bricks works as expected. Can only be set programmatically. Not by an environment variable.

## 7.45.2 Member Function Documentation

### 7.45.2.1 extra()

```
def openzgy.impl.file.SDConfig.extra (
    self )
```

Legaltag, writeid, and seismicmeta are usually packed into a single "extra" dictionary when creating a new file. If any of them are unset they will be excluded from the dictionary instead of being passed as some default value.

CAVEAT: The keys in the "extra" dictionary are not supposed to be hard coded as I do here. They are defined in seismic-store-client-api-cpp/src/src/core/Constants.{cc,h}. Cannot access that file here.

NOTE: Python dicts have an undefined sort order, as does json. To simplify testing I sort the keys in the "extra" dict. If SDAPI for some reason should require a specific ordering then "seismicmeta" needs to be passed as a string.

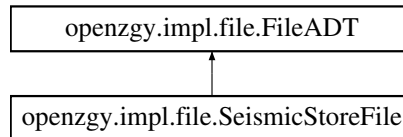
The documentation for this class was generated from the following file:

- openzgy/impl/file.py



### 7.46 openzgy.impl.file.SeismicStoreFile Class Reference

Inheritance diagram for openzgy.impl.file.SeismicStoreFile:



#### Public Member Functions

- def `__init__` (self, filename, mode, iocontext)
- def `xx_eof` (self)
- def `xx_close` (self)
- def `xx_read` (self, in\_offset, in\_size, \*usagehint=UsageHint.Unknown)
- def `xx_readv` (self, requests, \*parallel\_ok=False, immutable\_ok=False, transient\_ok=False, usagehint=UsageHint.Unknown)
- def `xx_write` (self, data, offset, \*usagehint=UsageHint.Unknown)
- def `threadsafe` (self)
- def `xx_iscloud` (self)

#### 7.46.1 Detailed Description

Access data in seismic store as a linear file even when the dataset has multiple segments. There are some limitations on write.

- \* Writes starting at EOF are allowed, and will cause a new segment to be written.
- \* Writes starting past EOF, signifying a hole in the data, are not allowed.
- \* Writes starting before EOF are only allowed if offset,size exactly matches a previous write. This will cause that segment to be rewritten.
- \* Possible future extension: For the last segment only offset needs to match. This means the last segment may be resized.

For read the class provides a `readv()` method to do scatter/gather reads. The code will then consolidate adjacent bricks to get larger brick size sent to SDAPI. Optionally parallelize requests that cannot be consolidated.

#### 7.46.2 Constructor & Destructor Documentation

## 7.46.2.1 `__init__()`

```
def openzgy.impl.file.SeismicStoreFile.__init__ (
    self,
    filename,
    mode,
    iocontext )
```

Open a file in the specified mode, which must be "rb" or "w+b".  
Caller should use a "with" block to ensure the file gets closed.  
The iocontext is an optional data structure that the user may specify when a reader is created. It might be used to hold user credentials etc. needed to access the low level file.  
TODO-Low: support "r+b" (update) at some point in the future.

Reimplemented from [openzgy.impl.file.FileADT](#).

## 7.46.3 Member Function Documentation

### 7.46.3.1 `xx_close()`

```
def openzgy.impl.file.SeismicStoreFile.xx_close (
    self )
```

Close a previously opened file.  
No action if the file is already closed.

Reimplemented from [openzgy.impl.file.FileADT](#).

### 7.46.3.2 `xx_read()`

```
def openzgy.impl.file.SeismicStoreFile.xx_read (
    self,
    offset,
    size,
    * usagehint = UsageHint.Unknown )
```

Read binary data from the file. Both size and offset are mandatory.  
I.e. caller is not allowed to read "the entire file", and not allowed to "read from where I left off the last time".  
The actual reading will be done in a derived class.  
The base class only validates the arguments.

Reimplemented from [openzgy.impl.file.FileADT](#).

### 7.46.3.3 xx\_readv()

```
def openzgy.impl.file.SeismicStoreFile.xx_readv (
    self,
    requests,
    * parallel_ok = False,
    immutable_ok = False,
    transient_ok = False,
    usagehint = UsageHint.Unknown )
```

Handle both brick consolidation and multi threading.

This implementation will issue a single readv() request to the seismic store wrapper, wait for all threads to complete, and then deliver all the results. For this reason it needs to allocate a buffer to hold the entire data to be read.

In the future it might be possible to have the seismic store wrapper support delivery callbacks and for it to allocate the result buffers itself. This saves some memory and also allows data to be decompressed if needed and copied out to user space as the bricks become available. Caveat: requests may need to be split if they cross a segment boundary. This means that we need support for partial delivery. Which would complicate things a lot.

Reimplemented from [openzgy.impl.file.FileADT](#).

### 7.46.3.4 xx\_write()

```
def openzgy.impl.file.SeismicStoreFile.xx_write (
    self,
    data,
    offset,
    * usagehint = UsageHint.Unknown )
```

Write binary data to the file. Offset is mandatory. I.e. caller is not allowed to "write to where I left off the last time". The actual writing will be done in a derived class. The base class only validates the arguments.

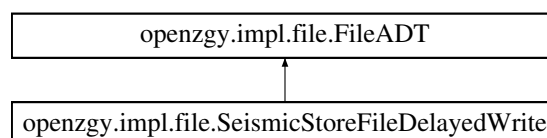
Reimplemented from [openzgy.impl.file.FileADT](#).

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

- openzgy/impl/file.py

## 7.47 openzgy.impl.file.SeismicStoreFileDelayedWrite Class Reference

Inheritance diagram for openzgy.impl.file.SeismicStoreFileDelayedWrite:



## Public Member Functions

- `def __init__` (self, filename, mode, iocontext)
- `def __enter__` (self)
- `def __exit__` (self, type, value, traceback)
- `def xx_eof` (self)
- `def xx_write` (self, data, offset, \*usagehint=UsageHint.Unknown)
- `def xx_read` (self, offset, size, \*usagehint=UsageHint.Unknown)
- `def xx_readv` (self, requests, \*parallel\_ok=False, immutable\_ok=False, transient\_ok=False, usagehint=UsageHint.Unknown, \*\*kwargs)
- `def xx_close` (self)
- `def threadsafe` (self)
- `def xx_iscloud` (self)

### 7.47.1 Detailed Description

Improve on `SeismicStoreFile`, have it buffer large chunks of data before writing it out to a new segment.

- \* Writes starting at EOF are allowed, and will buffer data in the "open segment" until explicitly flushed.
- \* Writes starting past EOF, signifying a hole in the data, are not allowed.
- \* Writes fully inside the open segment are allowed.
- \* Writes starting before the open segment are only allowed if offset,size exactly matches a previous write. This will cause that segment to be rewritten. As a corollary, writes cannot span the closed segment / open segment boundary.
- \* Possible future extension: For the last segment only offset needs to match. This means the last segment may be resized. Why we might want this: On opening a file with existing data bricks we might choose to read the last segment and turn it into an open segment. Then delete (in memory only) the last segment. When it is time to flush the data it gets rewritten. This allows adding bricks to a file, while still ensuring that all segments except first and last need to be the same size. Note that there are other tasks such as incrementally updating statistics and histogram that might turn out to be a lot of work.
- \* When used to create ZGY files, caller must honor the convention that all segments except the first and last must have the same size.
- \* Caveat: The fact that random writes are sometimes allowed, sometimes not depending on the segment number violates the principle of least surprise. And makes for more elaborate testing. For ZGY it is quite useful though. ZGY can recover from a `ZgySegmentIsClosed` exception by abandoning (leaking) the current block and write it to a new location. With a typical access pattern this will happen only occasionally.

### 7.47.2 Constructor & Destructor Documentation

### 7.47.2.1 `__init__()`

```
def openzgy.impl.file.SeismicStoreFileDelayedWrite.__init__ (
    self,
    filename,
    mode,
    iocontext )
```

Open a file in the specified mode, which must be "rb" or "w+b".  
Caller should use a "with" block to ensure the file gets closed.  
The iocontext is an optional data structure that the user may  
specify when a reader is created. It might be used to hold  
user credentials etc. needed to access the low level file.  
TODO-Low: support "r+b" (update) at some point in the future.

Reimplemented from [openzgy.impl.file.FileADT](#).

## 7.47.3 Member Function Documentation

### 7.47.3.1 `xx_close()`

```
def openzgy.impl.file.SeismicStoreFileDelayedWrite.xx_close (
    self )
```

Close a previously opened file.  
No action if the file is already closed.

Reimplemented from [openzgy.impl.file.FileADT](#).

### 7.47.3.2 `xx_eof()`

```
def openzgy.impl.file.SeismicStoreFileDelayedWrite.xx_eof (
    self )
```

Current size of the zgy file, including any buffered unwritten data.

### 7.47.3.3 `xx_read()`

```
def openzgy.impl.file.SeismicStoreFileDelayedWrite.xx_read (
    self,
    offset,
    size,
    * usagehint = UsageHint.Unknown )
```

Read binary data from the file. Both size and offset are mandatory. I.e. caller is not allowed to read "the entire file", and not allowed to "read from where I left off the last time". The actual reading will be done in a derived class. The base class only validates the arguments.

Reimplemented from [openzgy.impl.file.FileADT](#).

### 7.47.3.4 `xx_write()`

```
def openzgy.impl.file.SeismicStoreFileDelayedWrite.xx_write (
    self,
    data,
    offset,
    * usagehint = UsageHint.Unknown )
```

Write data to seismic store, buffering the writes to get larger segment sizes. Writes are only allowed at offset 0 and at EOF. This is less general then the parent type which lets us rewrite any segment as long as its size does not change.

Segment 0 contains just the headers and is always written in one operation, so this is not buffered. Segment 0 can be both smaller and larger than segsize. Which is another reason to bypass the buffering code. Also, if we are rewriting data we bypass the buffering and require that the caller updates the entire segment. ZGY will currently only rewrite segment 0.

If segsize is zero no buffering is done and each write will either create a new segment or completely rewrite an existing segment.

Reimplemented from [openzgy.impl.file.FileADT](#).

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

- `openzgy/impl/file.py`

## 7.48 `openzgy.impl.stats.StatisticData` Class Reference

### Public Member Functions

- `def __init__ (self, other=None)`
- `def __repr__ (self)`
- `def __str__ (self)`
- `def add (self, data, factor)`
- `def scale (self, slope, intercept)`
- `def __eq__ (self, other)`
- `def __ne__ (self, other)`
- `def __add__ (self, other)`
- `def __iadd__ (self, other)`
- `def __mul__ (self, factor)`
- `def __rmul__ (self, factor)`
- `def __imul__ (self, factor)`

### 7.48.1 Detailed Description

Accumulate statistics: count, sum, sum of squares, and value range.

### 7.48.2 Member Function Documentation

#### 7.48.2.1 scale()

```
def openzgy.impl.stats.StatisticData.scale (
    self,
    slope,
    intercept )
```

Calculate the linear transform needed to convert from one range (typically the natural data range of the integral storage type) to the data range that the application wants to see. Then update the statistics in place so they look like the transform had been done on every single data point before adding it.

The decoded value Y is given by a linear transform of the coded value X:

$$Y = \text{intercept} + \text{slope} * X$$

where intercept and slope are given by the coding range and the value range of type T (see below). The statistics of Y are then:

```
SUM_Y = SUM(intercept + slope*x)
= n*intercept + slope*SUM(x) = n*intercept + slope*SUM_X

SSQ_Y = SUM((intercept + slope*x)^2)
= SUM(intercept^2 + 2*intercept*slope*x + slope^2*x^2)
= n*intercept^2 + 2*intercept*slope*SUM(x) + slope^2*SUM(x^2)
= n*intercept^2 + 2*intercept*slope*SUM_X + slope^2*SSQ_X

MIN_Y = MIN(intercept + slope*x)
= intercept + slope*MIN(x)
= intercept + slope*MIN_X

MAX_Y = MAX(intercept + slope*x)
= intercept + slope*MAX(x)
= intercept + slope*MAX_X
```

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

- openzgy/impl/stats.py

## 7.49 openzgy.impl.meta.StringListV1 Class Reference

### Public Member Functions

- def **\_\_init\_\_** (self, di)
- def **size** (cls, oh=None, ih=None)
- def **load** (cls, di, buf)
- def **read** (cls, f, oh, ih)
- def **dump** (self, \*file=None)

## 7.49.1 Detailed Description

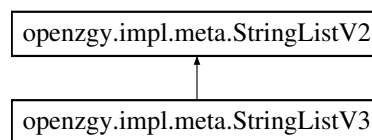
ZGY V1 had no string list, so this section is empty.

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

- `openzgy/impl/meta.py`

## 7.50 openzgy.impl.meta.StringListV2 Class Reference

Inheritance diagram for `openzgy.impl.meta.StringListV2`:



### Public Member Functions

- `def \_\_init\_\_ (self, buf=None, oh=None, ih=None)`
- `def size (cls, oh=None, ih=None)`
- `def read (cls, f, oh, ih)`
- `def dump (self, *file=None)`

### 7.50.1 Detailed Description

The string list holds 5 null terminated strings:  
`srcname`, `srcdesc`, `hprjsys`, `hunit.name`, `vunit.name`.  
The reader will ignore trailing garbage. This is  
the one place we might add additional information  
without breaking existing readers.

### 7.50.2 Constructor & Destructor Documentation

#### 7.50.2.1 `__init__()`

```
def openzgy.impl.meta.StringListV2.__init__ (
    self,
    buf = None,
    oh = None,
    ih = None )
```

Unpack a byte buffer into a new instance of this header.  
Caller is responsible for all I/O, so we don't need an `iocontext`.



### 7.50.3 Member Function Documentation

#### 7.50.3.1 size()

```
def openzgy.impl.meta.StringListV2.size (
    cls,
    oh = None,
    ih = None )
```

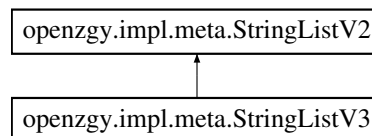
On read the size is stored in the offsetheader,  
which may have gotten it from the infoheader.  
On write the size depends on the strings written  
and we might not be able to trust the offsetheader.

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

- openzgy/impl/meta.py

## 7.51 openzgy.impl.meta.StringListV3 Class Reference

Inheritance diagram for openzgy.impl.meta.StringListV3:



### Additional Inherited Members

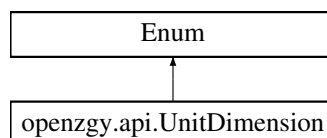
The documentation for this class was generated from the following file:

- openzgy/impl/meta.py

## 7.52 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

### 7.52.1 Detailed Description

Horizontal or vertical dimension as used in the public API.

Horizontal or vertical dimension as used in the public API.

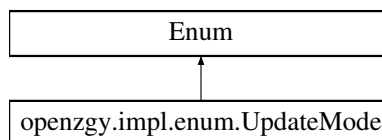
Horizontal dimension may be length or arc angle, although most applications only support length. Vertical dimension may be time or length. Vertical length is of course the same as depth. Arguably there should have been separate enums for horizontal and vertical dimension since the allowed values differ.

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

- openzgy/api.py

## 7.53 openzgy.impl.enum.UpdateMode Class Reference

Inheritance diagram for openzgy.impl.enum.UpdateMode:



## Static Public Attributes

- int **Never** = 0
- int **Constant** = 1
- int **Always** = 4
- int **Pedantic** = 5

### 7.53.1 Detailed Description

WORK IN PROGRESS, potential configurable behavior.

A ZGY file cannot be updated once created, but individual bricks might be written to more than once while the file is still open for create.

Updating a brick might cause loss of quality if the update was made as part of a read/modify/write cycle. It might also cause space to be wasted in the file since ZGY does not try to recycle freed bricks. For this reason the application should explicitly indicate that it accepts the loss of quality and/or leakage.

Kinds of leakage:

- Brick to be overwritten is in a closed segment. This is expected to be rare, and only relevant for cloud storage.
- Brick to be overwritten and/or new brick is compressed and the new data is smaller. Leaks the size difference, although for implementation reasons we might want to leak the entire old brick ("Pedantic" mode).
- Brick to be overwritten and/or new brick is compressed and the new data is larger. Leaks the old brick.

The default is "Always" for uncompressed local files and "Constant" otherwise.

It is fairly safe to set an uncompressed cloud file to "Always" but there are some scenarios where very small regions are written to a large file where this might cause much leakage. So the caller needs to confirm he knows what he is doing.

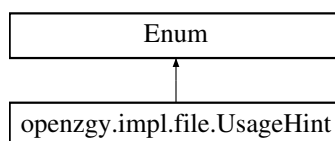
Compressed files should only be set to "Always" in very special cases or in unit tests. The potential leakage is much larger, as is the problem of multiple compress and decompress cycles causing noise.

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

- openzgy/impl/enum.py

## 7.54 openzgy.impl.file.UsageHint Class Reference

Inheritance diagram for openzgy.impl.file.UsageHint:



### Static Public Attributes

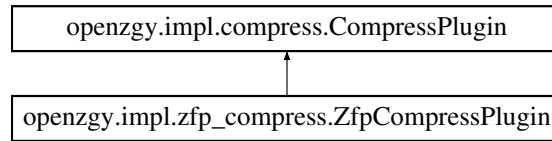
- int **Unknown** = 0x00
- int **TextFile** = 0x01
- int **Header** = 0x10
- int **Data** = 0x20
- int **Compressed** = 0x40,

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

- openzgy/impl/file.py

## 7.55 openzgy.impl.zfp\_compress.ZfpCompressPlugin Class Reference

Inheritance diagram for openzgy.impl.zfp\_compress.ZfpCompressPlugin:



### Public Member Functions

- def **\_\_init\_\_** (self, snr=30)
- def **\_\_call\_\_** (self, data)
- def **\_\_str\_\_** (self)
- def **dump** (self, msg=None, \*outfile=None, text=True, csv=False, reset=True)

### Static Public Member Functions

- def **compress** (data, snr=30, stats=None)
- def **decompress** (cdata, status, shape, file\_dtype, user\_dtype)
- def **factory** (snr=30)

### 7.55.1 Detailed Description

Implement ZFP compression. See the CompressPlugin base type for details. Most methods are static or class methods. The exception is `__init__`, `__call__`, and `dump()`.

### 7.55.2 Member Function Documentation

#### 7.55.2.1 `__call__()`

```
def openzgy.impl.zfp_compress.ZfpCompressPlugin.__call__ (
    self,
    data )
```

Invoke the compressor with arguments passed by the constructor.

Reimplemented from [openzgy.impl.compress.CompressPlugin](#).

### 7.55.2.2 decompress()

```
def openzgy.impl.zfp_compress.ZfpCompressPlugin.decompress (
    cdata,
    status,
    shape,
    file_dtype,
    user_dtype ) [static]
```

This is an abstract method.

Decompress bytes or similar into a `numpy.ndarray`.

Arguments:

```
cdata      -- bytes or bytes-like compressed data,
              possibly with trailing garbage.
status      -- Currently always BrickStatus.Compressed,
              in the future the status might be used to
              distinguish between different compression
              algorithms instead of relying on magic numbers.
shape       -- Rank and size of the result in case this is
              not encoded by the compression algorithm.
file_dtype  -- Original value type before compression,
              in case the decompressor cannot figure it out.
              This will exactly match the dtype of the
              data buffer passed to the compressor.
user_dtype  -- Required value type of returned array.
```

Passing an uncompressed brick to this function is an error.  
We don't have enough context to handle uncompressed bricks  
that might require byteswapping and fix for legacy quirks.  
Also cannot handle constant bricks, missing bricks, etc.

The reason `user_dtype` is needed is to avoid additional  
quantization noise when the user requests integer compressed data  
to be read as float. the decompressor might need to convert  
float data to int, only to have it converted back to float later.

Current assumptions made of all candidate algorithms:

- The compressed data stream may have trailing garbage;  
this will be silently ignored by the decompressor.
- The compressed data stream will never be longer than  
the uncompressed data. This needs to be enforced by  
the compressor. The compressor is allowed to give up  
and tell the caller to not compress this brick.
- The reason for the two assumptions above is an  
implementation detail; the reported size of a  
compressed brick is not completely reliable.  
This might change in the next version
- The compressed data stream must start with a magic  
number so the decompressor can figure out whether  
this is the correct algorithm to use.

If the assumptions cannot be met, the compressor / decompressor  
for this particular type could be modified to add an extra header  
with the compressed size and a magic number. Or we might add a  
(size, algorithm number) header to every compressed block to  
relieve the specific compressor / decompressor from worrying  
about this. Or the brick status could be used to encode which  
algorithm was used, picked up from the MSB of the lup entry.  
Which would also require the compressor to return both the  
actual compressed data and the code to identify the decompressor.  
That is the main reason we are also passed the "status" arg.  
Caveat: If adding an extra header, keep in mind that this header  
must be included when checking that the compressed stream is not  
too big.

Reimplemented from [openzgy.impl.compress.CompressPlugin](#).

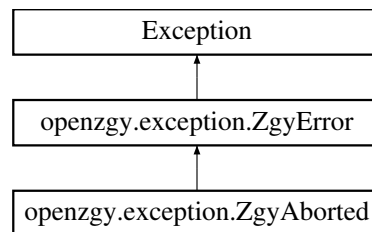
The documentation for this class was generated from the following file:

- [openzgy/impl/zfp\\_compress.py](#)

## 7.56 openzgy.exception.ZgyAborted Class Reference

User aborted the operation.

Inheritance diagram for openzgy.exception.ZgyAborted:



### 7.56.1 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.

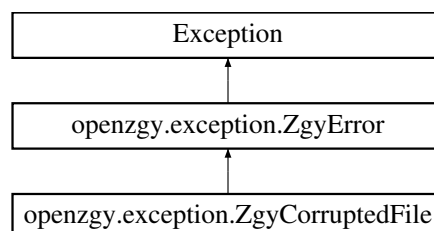
The documentation for this class was generated from the following file:

- [openzgy/exception.py](#)

## 7.57 openzgy.exception.ZgyCorruptedFile Class Reference

The ZGY file became corrupted while writing to it.

Inheritance diagram for openzgy.exception.ZgyCorruptedFile:



### 7.57.1 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.
```

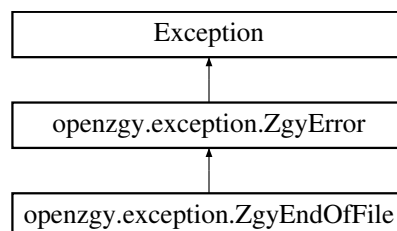
The documentation for this class was generated from the following file:

- openzgy/exception.py

## 7.58 openzgy.exception.ZgyEndOfFile Class Reference

Trying to read past EOF.

Inheritance diagram for openzgy.exception.ZgyEndOfFile:



### 7.58.1 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.
```

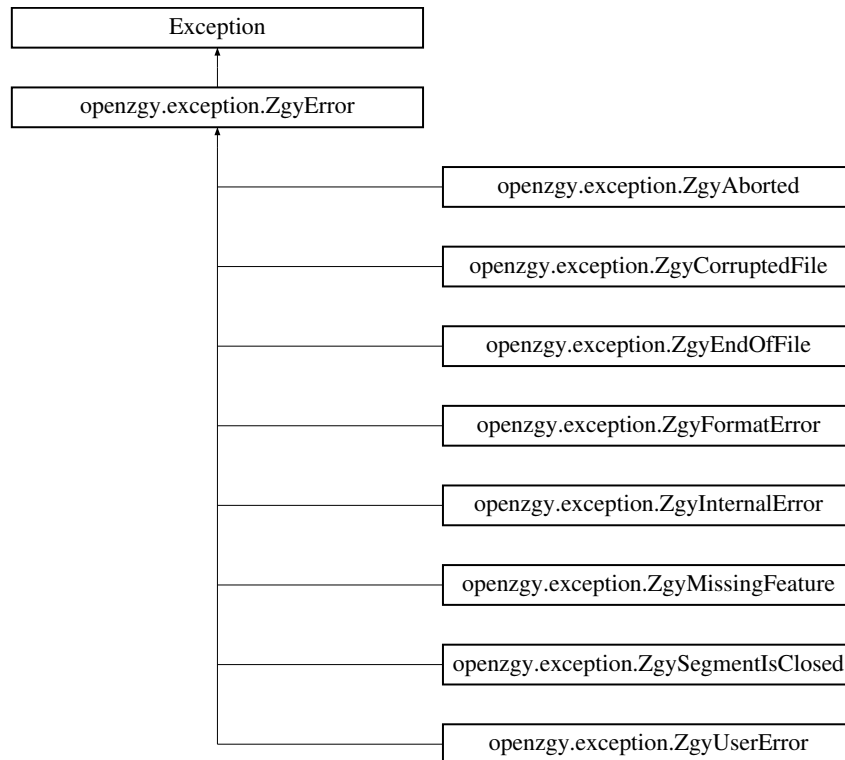
The documentation for this class was generated from the following file:

- openzgy/exception.py

## 7.59 openzgy.exception.ZgyError Class Reference

Base class for all exceptions thrown by OpenZGY.

Inheritance diagram for openzgy.exception.ZgyError:



### 7.59.1 Detailed Description

Base class for all exceptions thrown by OpenZGY.

Base class for all exceptions thrown by %OpenZGY.

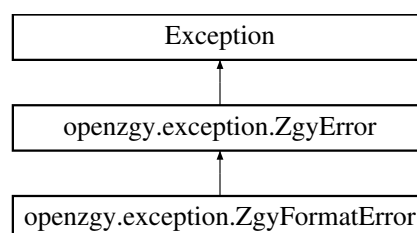
The documentation for this class was generated from the following file:

- openzgy/exception.py

## 7.60 openzgy.exception.ZgyFormatError Class Reference

Corrupted or unsupported ZGY file.

Inheritance diagram for openzgy.exception.ZgyFormatError:





### 7.60.1 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`

## 7.61 openzgy.impl.bulk.ZgyInternalBulk Class Reference

### Public Member Functions

- `def __init__(self, f, metadata, *compressed=False)`
- `def readConstantValue(self, start, size, lod=0, as_float=True, *verbose=None)`
- `def readToExistingBuffer(self, result, start, lod, as_float, *verbose=None)`

### 7.61.1 Detailed Description

Read or write bulk data. The meta data needs to have been read already. The user-callable API will forward its read requests here.

### 7.61.2 Member Function Documentation

#### 7.61.2.1 readConstantValue()

```
def openzgy.impl.bulk.ZgyInternalBulk.readConstantValue (
    self,
    start,
    size,
    lod = 0,
    as_float = True,
    * verbose = None )
```

Check to see if the specified region is known to have all samples set to the same value. A return value that is not `None` signifies that a regular read would return all samples set to that value. A return value of `None` means we don't know. This method is only intended as a hint to improve performance.

## 7.61.2.2 readToExistingBuffer()

```
def openzgy.impl.bulk.ZgyInternalBulk.readToExistingBuffer (
    self,
    result,
    start,
    lod,
    as_float,
    * verbose = None )
```

Read bulk data starting at "start" in index space and store the result in the provided 3d numpy array. Start should be in the range (0,0,0) to Size-1. The count of samples to read is implied by the size of the provided result array that is passed in. The valid data types for the result array are float32 (in which case samples stored as int8 or int16 will be scaled) or the files's storage value type (in which case there is no scaling). It is valid to pass a count that includes the padding area between the survey and the end of the current brick, but not past that point.

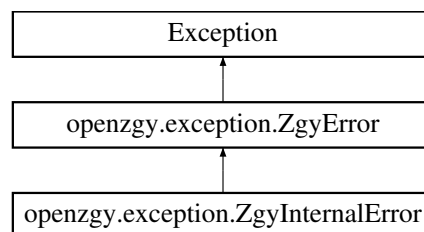
The documentation for this class was generated from the following file:

- openzgy/impl/bulk.py

## 7.62 openzgy.exception.ZgyInternalError Class Reference

Exception that might be caused by a bug in OpenZGY.

Inheritance diagram for openzgy.exception.ZgyInternalError:



### 7.62.1 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

## 7.63 openzgy.impl.meta.ZgyInternalMeta Class Reference

### Public Member Functions

- def `__init__` (self, myfile)
- def `dumpRaw` (self, \*file=None)

### 7.63.1 Detailed Description

Holds references to all the individual headers needed to access ZGY. The information is stored in the instance in a format that is tightly coupled to the file format, so there will be one layer (but hopefully just one) above us that is responsible for the public API.

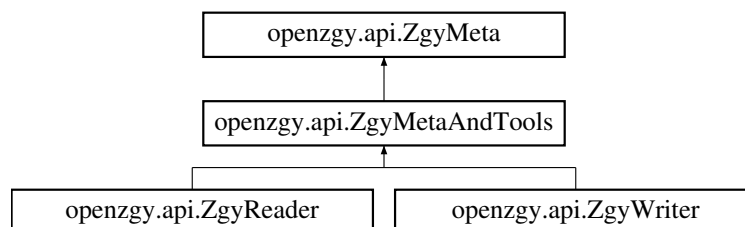
The documentation for this class was generated from the following file:

- openzgy/impl/meta.py

## 7.64 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 [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 [bricksize](#) (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)

## 7.64.1 Detailed Description

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

```
Base class shared betewwn ZgyReader and ZgyWriter.
```

## 7.64.2 Constructor & Destructor Documentation

### 7.64.2.1 `__init__()`

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

Create an instance, providing the `ZgyInternalMeta` to use.

## 7.64.3 Member Function Documentation

### 7.64.3.1 `annotcorners()`

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

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

### 7.64.3.2 annotinc()

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

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

### 7.64.3.3 annotstart()

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

First inline and crossline numbers.

### 7.64.3.4 brickcount()

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

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

### 7.64.3.5 bricksizes()

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

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

### 7.64.3.6 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))

### 7.64.3.7 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.

### 7.64.3.8 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.

### 7.64.3.9 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.

### 7.64.3.10 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.

### 7.64.3.11 hunitfactor()

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

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

### 7.64.3.12 hunitname()

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

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

### 7.64.3.13 indexcorners()

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

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

## 7.64.3.14 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.

## 7.64.3.15 nlods()

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

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

## 7.64.3.16 numthreads()

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

How many threads to use when reading. Currently ignored.

## 7.64.3.17 size()

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

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

## 7.64.3.18 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.



### 7.64.3.19 zinc()

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

Sample interval, given in the vertical unit.

### 7.64.3.20 zstart()

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

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

### 7.64.3.21 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.

### 7.64.3.22 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.

### 7.64.3.23 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.

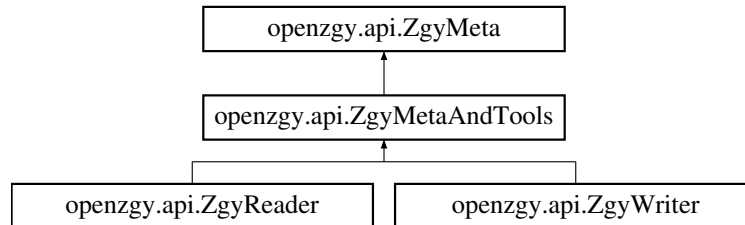
The documentation for this class was generated from the following file:

- openzgy/api.py

## 7.65 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)

### 7.65.1 Detailed Description

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

Base class shared betewwn `ZgyReader` and `ZgyWriter`.  
Adds coordinate conversion tools.

### 7.65.2 Member Function Documentation

#### 7.65.2.1 `annotToIndex()`

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

Convert inline, crossline to ordinal

### 7.65.2.2 annotToWorld()

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

Convert inline, crossline to world X,Y

### 7.65.2.3 indexToAnnot()

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

Convert ordinal to inline, crossline

### 7.65.2.4 indexToWorld()

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

Convert ordinal to world X,Y

### 7.65.2.5 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)

## 7.65.2.6 worldToAnnot()

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

Convert world X,Y to inline, crossline

## 7.65.2.7 worldToIndex()

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

Convert world X,Y to ordinal

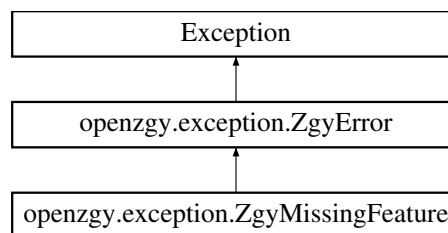
The documentation for this class was generated from the following file:

- openzgy/api.py

## 7.66 openzgy.exception.ZgyMissingFeature Class Reference

Missing feature.

Inheritance diagram for openzgy.exception.ZgyMissingFeature:



### 7.66.1 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.

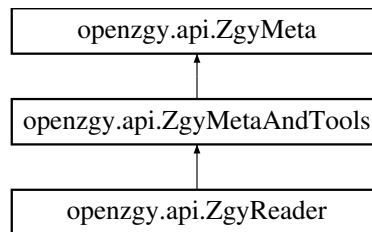
The documentation for this class was generated from the following file:

- openzgy/exception.py

### 7.67 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

##### 7.67.1 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.

##### 7.67.2 Member Function Documentation

## 7.67.2.1 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

## 7.67.2.2 readconst()

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

Get hint about all constant region.

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.

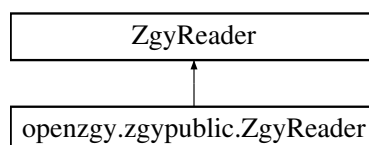
For int8 and int16 files the caller may specify whether to scale the values or not.

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

- openzgy/api.py

## 7.68 openzgy.zgypublic.ZgyReader Class Reference

Inheritance diagram for openzgy.zgypublic.ZgyReader:



### Public Member Functions

- def `__init__` (self, \*args, \*\*kwargs)  
*Ignore iocontext and verbose arguments.*
- def `read` (self, \*args, \*\*kwargs)  
*Ignore iocontext and verbose arguments.*
- def `meta` (self)  
*Old: strings, new: enums for 4 properties.*
- def `datatype` (self)  
*Old: string, new: [openzgy.api.SampleDataType](#).*
- def `zunitdim` (self)  
*Old: string, new: [openzgy.api.UnitDimension](#).*
- def `hunitdim` (self)  
*Old: string, new: [openzgy.api.UnitDimension](#).*
- def `bricksize` (self)  
*Old: missing from api.*

### 7.68.1 Member Function Documentation

#### 7.68.1.1 `bricksize()`

```
def openzgy.zgypublic.ZgyReader.bricksize (  
    self )
```

Old: missing from api.

For the old API the bricksize will always be returned as (64,64,64) on read and always set to that value on file create.

#### 7.68.1.2 `meta()`

```
def openzgy.zgypublic.ZgyReader.meta (  
    self )
```

Old: strings, new: enums for 4 properties.

datatype, sourcetype, zunitdim, hunitdim need to be mapped.

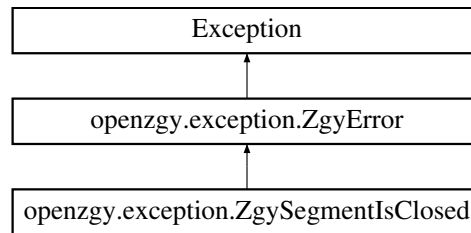
The documentation for this class was generated from the following file:

- openzgy/zgypublic.py

## 7.69 openzgy.exception.ZgySegmentIsClosed Class Reference

Exception used internally to request a retry.

Inheritance diagram for openzgy.exception.ZgySegmentIsClosed:



### 7.69.1 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.

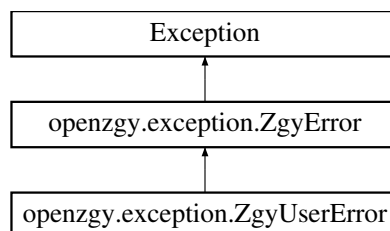
The documentation for this class was generated from the following file:

- openzgy/exception.py

## 7.70 openzgy.exception.ZgyUserError Class Reference

Exception that might be caused by the calling application.

Inheritance diagram for openzgy.exception.ZgyUserError:



### 7.70.1 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



### 7.71 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.*

#### 7.71.1 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.

#### 7.71.2 Constructor & Destructor Documentation

##### 7.71.2.1 `__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.

#### 7.71.3 Member Function Documentation

### 7.71.3.1 delete()

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

Delete a file.

Works both for local and cloud files.

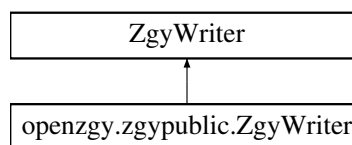
Delete a file. Works both for local and cloud files.  
Note that the instance must be of the correct (local or cloud) type.

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

- openzgy/api.py

## 7.72 openzgy.zgypublic.ZgyWriter Class Reference

Inheritance diagram for openzgy.zgypublic.ZgyWriter:



### Public Member Functions

- def `__init__` (self, \*args, \*\*kwargs)  
*Ignore iocontext and verbose, and map 4 enums to string.*
- def `write` (self, \*args, \*\*kwargs)  
*Ignore iocontext and verbose arguments.*
- def `finalize` (\*args, \*\*kwargs)  
*Ignore and let `__exit__` handle it instead.*
- def `meta` (self)  
*Old: strings, new: enums for 4 properties.*
- def `datatype` (self)  
*Old: string, new: [openzgy.api.SampleData Type](#).*
- def `zunitdim` (self)  
*Old: string, new: [openzgy.api.UnitDimension](#).*
- def `hunitdim` (self)  
*Old: string, new: [openzgy.api.UnitDimension](#).*
- def `bricksize` (self)  
*Old: missing from api.*

### 7.72.1 Member Function Documentation

### 7.72.1.1 bricksizes()

```
def openzgy.zgypublic.ZgyWriter.bricksizes (
    self )
```

Old: missing from api.

For the old API the bricksizes will always be returned as (64,64,64) on read and always set to that value on file create.

### 7.72.1.2 meta()

```
def openzgy.zgypublic.ZgyWriter.meta (
    self )
```

Old: strings, new: enums for 4 properties.

datatype, sourcetype, zunitdim, hunitdim need to be mapped.

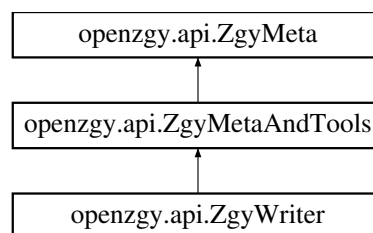
The documentation for this class was generated from the following file:

- openzgy/zgypublic.py

## 7.73 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)
- def `errorflag` (self)
- def `errorflag` (self, value)

## Additional Inherited Members

### 7.73.1 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.

### 7.73.2 Constructor & Destructor Documentation

#### 7.73.2.1 `__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 argumants 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.

### 7.73.3 Member Function Documentation

#### 7.73.3.1 close()

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

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

### 7.73.3.2 errorflag() [1/2]

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

If true this means that an error happened during a critical operation such as a write. This means the file being written is probably bad. It is recommended to close and delete the file. In the future OpenZGY might even do that automatically.

### 7.73.3.3 errorflag() [2/2]

```
def openzgy.api.ZgyWriter.errorflag (  
    self,  
    value )
```

Set or reset the flag indicating that the file got corrupted.

### 7.73.3.4 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.

### 7.73.3.5 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

### 7.73.3.6 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\_storage: 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





# Alpha version. Expect changes.

## Index

- `__call__`
    - `openzgy.impl.compress.CompressPlugin`, 34
    - `openzgy.impl.genlod.GenLodImpl`, 45
    - `openzgy.impl.zfp_compress.ZfpCompressPlugin`, 82
  - `__init__`
    - `openzgy.api.ZgyMeta`, 90
    - `openzgy.api.ZgyUtils`, 103
    - `openzgy.api.ZgyWriter`, 106
    - `openzgy.impl.compress.CompressPlugin`, 34
    - `openzgy.impl.file.FileADT`, 40
    - `openzgy.impl.file.FileConfig`, 42
    - `openzgy.impl.file.LocalFile`, 54
    - `openzgy.impl.file.SDConfig`, 68
    - `openzgy.impl.file.SeismicStoreFile`, 71
    - `openzgy.impl.file.SeismicStoreFileDelayedWrite`, 74
    - `openzgy.impl.meta.HistHeaderV1`, 48
    - `openzgy.impl.meta.HistHeaderV2`, 49
    - `openzgy.impl.meta.InfoHeaderV1`, 51
    - `openzgy.impl.meta.InfoHeaderV2`, 52
    - `openzgy.impl.meta.LookupTable`, 59
    - `openzgy.impl.meta.OffsetHeaderV1`, 60
    - `openzgy.impl.meta.OffsetHeaderV2`, 61
    - `openzgy.impl.meta.StringListV2`, 78
- `acpToOcp`
  - `openzgy.impl.transform`, 22
- `annotcorners`
  - `openzgy.api.ZgyMeta`, 90
- `annotinc`
  - `openzgy.api.ZgyMeta`, 90
- `annotstart`
  - `openzgy.api.ZgyMeta`, 91
- `annotToIndex`
  - `openzgy.api.ZgyMetaAndTools`, 96
- `annotToWorld`
  - `openzgy.api.ZgyMetaAndTools`, 96
- `binvalue`
  - `openzgy.impl.histogram.HistogramData`, 50
- `brickcount`
  - `openzgy.api.ZgyMeta`, 91
- `bricksize`
  - `openzgy.api.ZgyMeta`, 91
  - `openzgy.zgypublic.ZgyReader`, 101
  - `openzgy.zgypublic.ZgyWriter`, 104
- `calculate`
  - `openzgy.impl.meta.OffsetHeaderV1`, 60
  - `openzgy.impl.meta.OffsetHeaderV2`, 62
- `calculate_write`
  - `openzgy.impl.meta.InfoHeaderV2`, 53
- `checkformats`
  - `openzgy.impl.meta.HeaderBase`, 46
- `close`
  - `openzgy.api.ZgyWriter`, 107
- `compress`
  - `openzgy.impl.compress.CompressPlugin`, 35
- `corners`
  - `openzgy.api.ZgyMeta`, 91
- `datarange`
  - `openzgy.api.ZgyMeta`, 91
- `datatype`
  - `openzgy.api.ZgyMeta`, 92
- `decimate`
  - `openzgy.impl.lodalgo`, 18
- `decimate8`
  - `openzgy.impl.lodalgo`, 18
- `decompress`
  - `openzgy.impl.compress.CompressFactoryImpl`, 31
  - `openzgy.impl.compress.CompressPlugin`, 35
  - `openzgy.impl.zfp_compress.ZfpCompressPlugin`, 82
- `delete`
  - `openzgy.api.ZgyUtils`, 103
- `dump`
  - `openzgy.impl.compress.CompressPlugin`, 36
  - `openzgy.impl.meta.HeaderBase`, 47
- `errorflag`
  - `openzgy.api.ZgyWriter`, 108
- `extra`
  - `openzgy.impl.file.SDConfig`, 70
- `FileFactory`
  - `openzgy.impl.file`, 17
- `finalize`
  - `openzgy.api.ZgyWriter`, 108
- `generalTransform`
  - `openzgy.impl.transform`, 22
- `headersize`
  - `openzgy.impl.meta.HeaderBase`, 47
- `histogram`
  - `openzgy.api.ZgyMeta`, 92
- `hunitdim`
  - `openzgy.api.ZgyMeta`, 92
- `hunitfactor`

- openzgy.api.ZgyMeta, 93
- hunitname
  - openzgy.api.ZgyMeta, 93
- indexcorners
  - openzgy.api.ZgyMeta, 93
- indexToAnnot
  - openzgy.api.ZgyMetaAndTools, 97
- indexToWorld
  - openzgy.api.ZgyMetaAndTools, 97
- knownCompressors
  - openzgy.impl.compress.CompressFactoryImpl, 31
- knownDecompressors
  - openzgy.impl.compress.CompressFactoryImpl, 31
- meta
  - openzgy.api.ZgyMeta, 93
  - openzgy.zgypublic.ZgyReader, 101
  - openzgy.zgypublic.ZgyWriter, 105
- nlods
  - openzgy.api.ZgyMeta, 94
- np\_range
  - openzgy.impl.histogram.HistogramData, 50
- numthreads
  - openzgy.api.ZgyMeta, 94
- openzgy, 11
- openzgy.api, 11
  - ZgyCompressFactory, 13
  - ZgyKnownCompressors, 13
  - ZgyKnownDecompressors, 13
- openzgy.api.\_internal, 27
- openzgy.api.ProgressWithDots, 62
- openzgy.api.SampleDataType, 66
- openzgy.api.UnitDimension, 79
- openzgy.api.ZgyMeta, 89
  - \_\_init\_\_, 90
  - annotcorners, 90
  - annotinc, 90
  - annotstart, 91
  - brickcount, 91
  - bricksize, 91
  - corners, 91
  - datarange, 91
  - datatype, 92
  - histogram, 92
  - hunitdim, 92
  - hunitfactor, 93
  - hunitname, 93
  - indexcorners, 93
  - meta, 93
  - nlods, 94
  - numthreads, 94
  - size, 94
  - statistics, 94
  - zinc, 94
  - zstart, 95
  - zunitdim, 95
  - zunitfactor, 95
  - zunitname, 95
- openzgy.api.ZgyMetaAndTools, 96
  - annotToIndex, 96
  - annotToWorld, 96
  - indexToAnnot, 97
  - indexToWorld, 97
  - transform, 97
  - worldToAnnot, 97
  - worldToIndex, 98
- openzgy.api.ZgyReader, 99
  - read, 99
  - readconst, 100
- openzgy.api.ZgyUtils, 103
  - \_\_init\_\_, 103
  - delete, 103
- openzgy.api.ZgyWriter, 105
  - \_\_init\_\_, 106
  - close, 107
  - errorflag, 108
  - finalize, 108
  - write, 108
  - writeconst, 109
- openzgy.exception, 13
- openzgy.exception.ZgyAborted, 84
- openzgy.exception.ZgyCorruptedFile, 84
- openzgy.exception.ZgyEndOfFile, 85
- openzgy.exception.ZgyError, 86
- openzgy.exception.ZgyFormatError, 86
- openzgy.exception.ZgyInternalError, 88
- openzgy.exception.ZgyMissingFeature, 98
- openzgy.exception.ZgySegmentIsClosed, 102
- openzgy.exception.ZgyUserError, 102
- openzgy.impl, 14
- openzgy.impl.bulk, 15
- openzgy.impl.bulk.ErrorsWillCorruptFile, 38
- openzgy.impl.bulk.ScalarBuffer, 67
- openzgy.impl.bulk.ZgyInternalBulk, 87
  - readConstantValue, 87
  - readToExistingBuffer, 87
- openzgy.impl.compress.CompressFactoryImpl, 31
  - decompress, 31
  - knownCompressors, 31
  - knownDecompressors, 31
  - registerCompressor, 32
  - registerDecompressor, 32
- openzgy.impl.compress.CompressPlugin, 33
  - \_\_call\_\_, 34
  - \_\_init\_\_, 34
  - compress, 35
  - decompress, 35
  - dump, 36
- openzgy.impl.compress.CompressStats, 37
- openzgy.impl.enum, 15
- openzgy.impl.enum.BrickStatus, 30
- openzgy.impl.enum.RawCoordType, 63
- openzgy.impl.enum.RawDataType, 64

- openzgy.impl.enum.RawGridDefinition, 65
- openzgy.impl.enum.RawHorizontalDimension, 65
- openzgy.impl.enum.RawVerticalDimension, 66
- openzgy.impl.enum.UpdateMode, 80
- openzgy.impl.file, 16
  - FileFactory, 17
- openzgy.impl.file.Config, 37
- openzgy.impl.file.FileADT, 39
  - \_\_init\_\_, 40
  - xx\_close, 40
  - xx\_read, 40
  - xx\_readv, 41
  - xx\_write, 41
- openzgy.impl.file.FileConfig, 42
  - \_\_init\_\_, 42
- openzgy.impl.file.LocalFile, 53
  - \_\_init\_\_, 54
  - xx\_close, 54
- openzgy.impl.file.LocalFileLinux, 55
  - xx\_read, 55
  - xx\_readv, 55
  - xx\_write, 56
- openzgy.impl.file.LocalFileOther, 57
  - xx\_read, 57
  - xx\_readv, 57
  - xx\_write, 58
- openzgy.impl.file.SDConfig, 68
  - \_\_init\_\_, 68
  - extra, 70
- openzgy.impl.file.SeismicStoreFile, 71
  - \_\_init\_\_, 71
  - xx\_close, 72
  - xx\_read, 72
  - xx\_readv, 72
  - xx\_write, 73
- openzgy.impl.file.SeismicStoreFileDelayedWrite, 73
  - \_\_init\_\_, 74
  - xx\_close, 75
  - xx\_eof, 75
  - xx\_read, 75
  - xx\_write, 76
- openzgy.impl.file.UsageHint, 81
- openzgy.impl.genlod.GenLodBase, 43
- openzgy.impl.genlod.GenLodC, 44
- openzgy.impl.genlod.GenLodImpl, 45
  - \_\_call\_\_, 45
- openzgy.impl.histogram.HistogramData, 50
  - binvalue, 50
  - np\_range, 50
  - vv\_range, 51
- openzgy.impl.lodalgo, 17
  - decimate, 18
  - decimate8, 18
- openzgy.impl.lodalgo.DecimationType, 37
- openzgy.impl.meta, 19
- openzgy.impl.meta.AlphaLUPV1, 28
- openzgy.impl.meta.AlphaLUPV2, 28
- openzgy.impl.meta.AlphaLUPV3, 28
- openzgy.impl.meta.BrickLUPV1, 29
- openzgy.impl.meta.BrickLUPV2, 29
- openzgy.impl.meta.BrickLUPV3, 30
- openzgy.impl.meta.ErrorsWillCorruptFile, 39
- openzgy.impl.meta.FileHeader, 43
- openzgy.impl.meta.HeaderBase, 46
  - checkformats, 46
  - dump, 47
  - headersize, 47
  - pack, 47
  - read, 47
  - unpack, 48
- openzgy.impl.meta.HistHeaderV1, 48
  - \_\_init\_\_, 48
- openzgy.impl.meta.HistHeaderV2, 49
  - \_\_init\_\_, 49
- openzgy.impl.meta.HistHeaderV3, 50
- openzgy.impl.meta.InfoHeaderV1, 51
  - \_\_init\_\_, 51
- openzgy.impl.meta.InfoHeaderV2, 52
  - \_\_init\_\_, 52
  - calculate\_write, 53
- openzgy.impl.meta.InfoHeaderV3, 53
- openzgy.impl.meta.LookupTable, 59
  - \_\_init\_\_, 59
- openzgy.impl.meta.OffsetHeaderV1, 60
  - \_\_init\_\_, 60
  - calculate, 60
- openzgy.impl.meta.OffsetHeaderV2, 61
  - \_\_init\_\_, 61
  - calculate, 62
- openzgy.impl.meta.OffsetHeaderV3, 62
- openzgy.impl.meta.StringListV1, 77
- openzgy.impl.meta.StringListV2, 78
  - \_\_init\_\_, 78
  - size, 79
- openzgy.impl.meta.StringListV3, 79
- openzgy.impl.meta.ZgyInternalMeta, 89
- openzgy.impl.stats, 20
- openzgy.impl.stats.StatisticData, 76
  - scale, 77
- openzgy.impl.transform, 21
  - acpToOcp, 22
  - generalTransform, 22
- openzgy.impl.zfp\_compress.ZfpCompressPlugin, 82
  - \_\_call\_\_, 82
  - decompress, 82
- openzgy.iterator, 23
  - readall, 24
- openzgy.zgypublic, 24
- openzgy.zgypublic.ZgyReader, 100
  - bricksize, 101
  - meta, 101
- openzgy.zgypublic.ZgyWriter, 104
  - bricksize, 104
  - meta, 105
- pack
  - openzgy.impl.meta.HeaderBase, 47

- read
  - openzgy.api.ZgyReader, 99
  - openzgy.impl.meta.HeaderBase, 47
- readall
  - openzgy.iterator, 24
- readconst
  - openzgy.api.ZgyReader, 100
- readConstantValue
  - openzgy.impl.bulk.ZgyInternalBulk, 87
- readToExistingBuffer
  - openzgy.impl.bulk.ZgyInternalBulk, 87
- registerCompressor
  - openzgy.impl.compress.CompressFactoryImpl, 32
- registerDecompressor
  - openzgy.impl.compress.CompressFactoryImpl, 32
- scale
  - openzgy.impl.stats.StatisticData, 77
- size
  - openzgy.api.ZgyMeta, 94
  - openzgy.impl.meta.StringListV2, 79
- statistics
  - openzgy.api.ZgyMeta, 94
- transform
  - openzgy.api.ZgyMetaAndTools, 97
- unpack
  - openzgy.impl.meta.HeaderBase, 48
- vv\_range
  - openzgy.impl.histogram.HistogramData, 51
- worldToAnnot
  - openzgy.api.ZgyMetaAndTools, 97
- worldToIndex
  - openzgy.api.ZgyMetaAndTools, 98
- write
  - openzgy.api.ZgyWriter, 108
- writeconst
  - openzgy.api.ZgyWriter, 109
- xx\_close
  - openzgy.impl.file.FileADT, 40
  - openzgy.impl.file.LocalFile, 54
  - openzgy.impl.file.SeismicStoreFile, 72
  - openzgy.impl.file.SeismicStoreFileDelayedWrite, 75
- xx\_eof
  - openzgy.impl.file.SeismicStoreFileDelayedWrite, 75
- xx\_read
  - openzgy.impl.file.FileADT, 40
  - openzgy.impl.file.LocalFileLinux, 55
  - openzgy.impl.file.LocalFileOther, 57
  - openzgy.impl.file.SeismicStoreFile, 72
  - openzgy.impl.file.SeismicStoreFileDelayedWrite, 75
- xx\_readv
  - openzgy.impl.file.FileADT, 41
- openzgy.impl.file.LocalFileLinux, 55
- openzgy.impl.file.LocalFileOther, 57
- openzgy.impl.file.SeismicStoreFile, 72
- xx\_write
  - openzgy.impl.file.FileADT, 41
  - openzgy.impl.file.LocalFileLinux, 56
  - openzgy.impl.file.LocalFileOther, 58
  - openzgy.impl.file.SeismicStoreFile, 73
  - openzgy.impl.file.SeismicStoreFileDelayedWrite, 76
- ZgyCompressFactory
  - openzgy.api, 13
- ZgyKnownCompressors
  - openzgy.api, 13
- ZgyKnownDecompressors
  - openzgy.api, 13
- zinc
  - openzgy.api.ZgyMeta, 94
- zstart
  - openzgy.api.ZgyMeta, 95
- zunitdim
  - openzgy.api.ZgyMeta, 95
- zunitfactor
  - openzgy.api.ZgyMeta, 95
- zunitname
  - openzgy.api.ZgyMeta, 95