

## OpenZGY/C++ Public API (ALPHA)

Generated by Doxygen 1.8.17



<b>1 Main Page</b>	<b>1</b>
<b>2 Example</b>	<b>3</b>
<b>3 Module Index</b>	<b>5</b>
3.1 Modules	5
<b>4 Namespace Index</b>	<b>7</b>
4.1 Namespace List	7
<b>5 Hierarchical Index</b>	<b>9</b>
5.1 Class Hierarchy	9
<b>6 Class Index</b>	<b>11</b>
6.1 Class List	11
<b>7 File Index</b>	<b>13</b>
7.1 File List	13
<b>8 Module Documentation</b>	<b>15</b>
8.1 List of exceptions	15
8.1.1 Detailed Description	15
<b>9 Namespace Documentation</b>	<b>17</b>
9.1 OpenZGY Namespace Reference	17
9.1.1 Detailed Description	18
9.2 OpenZGY::Errors Namespace Reference	18
9.2.1 Detailed Description	18
9.3 OpenZGY::Formatters Namespace Reference	18
9.3.1 Detailed Description	19
9.4 OpenZGY::Impl Namespace Reference	19
9.4.1 Detailed Description	19
<b>10 Class Documentation</b>	<b>21</b>
10.1 OpenZGY::IOContext Class Reference	21
10.1.1 Detailed Description	21
10.1.2 Member Function Documentation	21
10.1.2.1 toString()	22
10.2 OpenZGY::IZgyMeta Class Reference	22
10.2.1 Detailed Description	24
10.3 OpenZGY::IZgyReader Class Reference	24
10.3.1 Detailed Description	25
10.3.2 Member Function Documentation	25
10.3.2.1 close()	25
10.3.2.2 read() [1/3]	25
10.3.2.3 read() [2/3]	26

10.3.2.4 read() [3/3]	26
10.3.2.5 readconst()	26
10.4 OpenZGY::IZgyTools Class Reference	27
10.4.1 Detailed Description	28
10.4.2 Member Function Documentation	28
10.4.2.1 transform()	28
10.4.2.2 transform1()	29
10.5 OpenZGY::IZgyUtils Class Reference	29
10.5.1 Detailed Description	30
10.5.2 Member Function Documentation	30
10.5.2.1 alturl()	30
10.5.2.2 deletefile()	31
10.5.2.3 utils()	31
10.6 OpenZGY::IZgyWriter Class Reference	31
10.6.1 Detailed Description	32
10.6.2 Member Function Documentation	33
10.6.2.1 close()	33
10.6.2.2 close_incomplete()	33
10.6.2.3 finalize()	33
10.6.2.4 reopen()	34
10.6.2.5 write() [1/3]	35
10.6.2.6 write() [2/3]	35
10.6.2.7 write() [3/3]	36
10.6.2.8 writeconst() [1/3]	36
10.6.2.9 writeconst() [2/3]	36
10.6.2.10 writeconst() [3/3]	37
10.7 OpenZGY::ProgressWithDots Class Reference	37
10.7.1 Detailed Description	37
10.7.2 Constructor & Destructor Documentation	38
10.7.2.1 ProgressWithDots()	38
10.7.3 Member Function Documentation	38
10.7.3.1 operator()()	38
10.8 OpenZGY::SampleHistogram Class Reference	39
10.8.1 Detailed Description	40
10.9 OpenZGY::SampleStatistics Class Reference	40
10.9.1 Detailed Description	41
10.10 OpenZGY::SeismicStoreIOContext Class Reference	41
10.10.1 Detailed Description	42
10.10.2 Member Function Documentation	42
10.10.2.1 aligned()	42
10.10.2.2 debug_trace()	43
10.10.2.3 legaltag()	43

10.10.2.4 maxhole()	43
10.10.2.5 maxsize()	43
10.10.2.6 sdapikey()	44
10.10.2.7 sdtoken()	44
10.10.2.8 sdtokencb()	44
10.10.2.9 sdurl()	44
10.10.2.10 segsize()	45
10.10.2.11 segsplit()	45
10.10.2.12 seismicmeta()	45
10.10.2.13 threads()	45
10.10.2.14 toString()	46
10.10.2.15 writeid()	46
10.11 OpenZGY::Errors::ZgyAborted Class Reference	46
10.11.1 Detailed Description	47
10.12 OpenZGY::Errors::ZgyCorruptedFile Class Reference	47
10.12.1 Detailed Description	47
10.13 OpenZGY::Errors::ZgyEndOfFile Class Reference	48
10.13.1 Detailed Description	48
10.14 OpenZGY::Errors::ZgyError Class Reference	49
10.14.1 Detailed Description	49
10.15 OpenZGY::Errors::ZgyFormatError Class Reference	50
10.15.1 Detailed Description	50
10.16 OpenZGY::Errors::ZgyInternalError Class Reference	50
10.16.1 Detailed Description	51
10.17 OpenZGY::Errors::ZgyIoError Class Reference	51
10.17.1 Detailed Description	52
10.18 OpenZGY::Impl::ZgyMeta Class Reference	52
10.18.1 Detailed Description	53
10.19 OpenZGY::Impl::ZgyMetaAndTools Class Reference	54
10.19.1 Detailed Description	55
10.19.2 Member Function Documentation	55
10.19.2.1 transform()	55
10.19.2.2 transform1()	55
10.20 OpenZGY::Errors::ZgyMissingFeature Class Reference	56
10.20.1 Detailed Description	56
10.21 OpenZGY::Impl::ZgyReader Class Reference	57
10.21.1 Detailed Description	57
10.21.2 Constructor & Destructor Documentation	57
10.21.2.1 ZgyReader()	57
10.21.3 Member Function Documentation	58
10.21.3.1 close()	58
10.21.3.2 read() [1/3]	58

10.21.3.3 read() [2/3]	59
10.21.3.4 read() [3/3]	59
10.21.3.5 readconst()	59
10.22 OpenZGY::Errors::ZgySegmentIsClosed Class Reference	60
10.22.1 Detailed Description	61
10.23 OpenZGY::Errors::ZgyUserError Class Reference	61
10.23.1 Detailed Description	61
10.24 OpenZGY::Impl::ZgyUtils Class Reference	62
10.24.1 Detailed Description	62
10.24.2 Constructor & Destructor Documentation	62
10.24.2.1 ZgyUtils()	62
10.24.3 Member Function Documentation	63
10.24.3.1 alturl()	63
10.24.3.2 deletefile()	63
10.25 OpenZGY::Impl::ZgyWriter Class Reference	64
10.25.1 Detailed Description	65
10.25.2 Constructor & Destructor Documentation	65
10.25.2.1 ZgyWriter()	65
10.25.2.2 ~ZgyWriter()	65
10.25.3 Member Function Documentation	65
10.25.3.1 close()	66
10.25.3.2 close_incomplete()	66
10.25.3.3 errorflag()	66
10.25.3.4 finalize()	67
10.25.3.5 set_errorflag()	67
10.25.3.6 write() [1/3]	67
10.25.3.7 write() [2/3]	68
10.25.3.8 write() [3/3]	68
10.25.3.9 writeconst() [1/3]	68
10.25.3.10 writeconst() [2/3]	69
10.25.3.11 writeconst() [3/3]	69
10.26 OpenZGY::ZgyWriterArgs Class Reference	69
10.26.1 Detailed Description	71
10.26.2 Member Function Documentation	71
10.26.2.1 bricksizes()	71
10.26.2.2 compressor()	71
10.26.2.3 corners()	72
10.26.2.4 datarange()	72
10.26.2.5 filename()	72
10.26.2.6 hunit()	72
10.26.2.7 ilinc()	73
10.26.2.8 ilstart()	73

10.26.2.9 iocontext()	73
10.26.2.10 lodcompressor()	73
10.26.2.11 merge()	73
10.26.2.12 metafrom()	74
10.26.2.13 size()	74
10.26.2.14 xline()	74
10.26.2.15 xstart()	74
10.26.2.16 zfp_compressor()	75
10.26.2.17 zfp_lodcompressor()	75
10.26.2.18 zinc()	75
10.26.2.19 zstart()	75
10.26.2.20 zunit()	75
<b>11 File Documentation</b>	<b>77</b>
11.1 api.cpp File Reference	77
11.1.1 Detailed Description	78
11.2 api.h File Reference	78
11.2.1 Detailed Description	80
11.3 example.h File Reference	81
11.3.1 Detailed Description	82
11.4 exception.h File Reference	82
11.4.1 Detailed Description	83
11.5 iocontext.h File Reference	83
11.5.1 Detailed Description	83
<b>Index</b>	<b>85</b>





# Chapter 1

## Main Page

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

- [IZgyReader](#) and its [IZgyMeta](#) base class.
- [IZgyWriter](#) and its [ZgyWriterArgs](#) argument package.
- [IZgyUtils](#) for anything not read or write.
- [List of exceptions](#) that you might want to catch.
- [SeismicStoreIOContext](#) for cloud credentials.
- [ProgressWithDots](#) example of progress reporting.
- [Example](#) Example application.

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



## Chapter 2

# Example

```
// 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.
#include <openzgy/api.h>
#include <iostream>
#include <stdexcept>
#include <stdlib.h>
void copy(const std::string& srcname, const std::string& dstname)
{
    using namespace OpenZGY;
    ProgressWithDots p1, p2;
    std::shared_ptr<IZgyReader> r = IZgyReader::open(srcname);
    std::shared_ptr<IZgyWriter> w = IZgyWriter::open(ZgyWriterArgs().metafrom(r).filename(dstname));
    const std::array<std::int64_t,3> size = r->size();
    const std::array<std::int64_t,3> brick = r->bricksize();
    const std::array<std::int64_t,3> bs{brick[0], brick[1], size[2]};
    const std::int64_t total = ((size[0] + bs[0] - 1) / bs[0]) *
                               ((size[1] + bs[1] - 1) / bs[1]);

    std::int64_t done{0};
    std::unique_ptr<float> buf(new float[bs[0]*bs[1]*bs[2]]);
    std::array<std::int64_t,3> pos;
    for (pos[0] = 0; pos[0] < size[0]; pos[0] += bs[0]) {
        for (pos[1] = 0; pos[1] < size[1]; pos[1] += bs[1]) {
            for (pos[2] = 0; pos[2] < size[2]; pos[2] += bs[2]) {
                r->read(pos, bs, buf.get(), 0);
                w->write(pos, bs, buf.get());
                p1(++done, total);
            }
        }
    }
    w->finalize(std::vector<DecimationType>(), p2);
    w->close();
}
int main(int argc, const char **argv)
{
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " infile outfile" << std::endl;
        exit(1);
    }
    try {
        copy(argv[1], argv[2]);
    }
    catch (const std::exception& ex) {
        std::cerr << argv[0] << ": " << ex.what() << std::endl;
        exit(1);
    }
}
```



## Chapter 3

# Module Index

### 3.1 Modules

Here is a list of all modules:

List of exceptions . . . . .	15
------------------------------	----



## Chapter 4

# Namespace Index

### 4.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">OpenZGY</a>	The entire public API is in this namespace . . . . .	17
<a href="#">OpenZGY::Errors</a>	Exceptions that can be thrown by <a href="#">OpenZGY</a> . . . . .	18
<a href="#">OpenZGY::Formatters</a>	Operator<< for readable output of enums etc . . . . .	18
<a href="#">OpenZGY::Impl</a>	Implementation of the abstract interfaces in <a href="#">OpenZGY</a> . . . . .	19





## Chapter 5

# Hierarchical Index

### 5.1 Class Hierarchy

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

std::exception	
std::runtime_error	
OpenZGY::Errors::ZgyError . . . . .	49
OpenZGY::Errors::ZgyAborted . . . . .	46
OpenZGY::Errors::ZgyCorruptedFile . . . . .	47
OpenZGY::Errors::ZgyEndOfFile . . . . .	48
OpenZGY::Errors::ZgyFormatError . . . . .	50
OpenZGY::Errors::ZgyInternalError . . . . .	50
OpenZGY::Errors::ZgyIoError . . . . .	51
OpenZGY::Errors::ZgyMissingFeature . . . . .	56
OpenZGY::Errors::ZgySegmentIsClosed . . . . .	60
OpenZGY::Errors::ZgyUserError . . . . .	61
OpenZGY::IOContext . . . . .	21
OpenZGY::SeismicStoreIOContext . . . . .	41
OpenZGY::IZgyMeta . . . . .	22
OpenZGY::Impl::ZgyMeta . . . . .	52
OpenZGY::Impl::ZgyMetaAndTools . . . . .	54
OpenZGY::Impl::ZgyReader . . . . .	57
OpenZGY::Impl::ZgyWriter . . . . .	64
OpenZGY::IZgyTools . . . . .	27
OpenZGY::Impl::ZgyMetaAndTools . . . . .	54
OpenZGY::IZgyReader . . . . .	24
OpenZGY::Impl::ZgyReader . . . . .	57
OpenZGY::IZgyWriter . . . . .	31
OpenZGY::Impl::ZgyWriter . . . . .	64
OpenZGY::IZgyUtils . . . . .	29
OpenZGY::Impl::ZgyUtils . . . . .	62
OpenZGY::ProgressWithDots . . . . .	37
OpenZGY::SampleHistogram . . . . .	39
OpenZGY::SampleStatistics . . . . .	40
OpenZGY::ZgyWriterArgs . . . . .	69



## Chapter 6

# Class Index

### 6.1 Class List

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

<a href="#">OpenZGY::IOContext</a>	Base class for backend specific context . . . . .	21
<a href="#">OpenZGY::IZgyMeta</a>	Base class of <a href="#">IZgyReader</a> and <a href="#">IZgyWriter</a> . . . . .	22
<a href="#">OpenZGY::IZgyReader</a>	Main API for reading ZGY files . . . . .	24
<a href="#">OpenZGY::IZgyTools</a>	Base class of <a href="#">IZgyReader</a> and <a href="#">IZgyWriter</a> . . . . .	27
<a href="#">OpenZGY::IZgyUtils</a>	Operations other than read and write . . . . .	29
<a href="#">OpenZGY::IZgyWriter</a>	Main API for creating ZGY files . . . . .	31
<a href="#">OpenZGY::ProgressWithDots</a>	Simple progress bar . . . . .	37
<a href="#">OpenZGY::SampleHistogram</a>	Histogram of all sample values on the file . . . . .	39
<a href="#">OpenZGY::SampleStatistics</a>	Statistics of all sample values on the file . . . . .	40
<a href="#">OpenZGY::SeismicStoreIOContext</a>	Credentials and configuration for Seismic Store . . . . .	41
<a href="#">OpenZGY::Errors::ZgyAborted</a>	User aborted the computation . . . . .	46
<a href="#">OpenZGY::Errors::ZgyCorruptedFile</a>	The ZGY file became corrupted while writing to it . . . . .	47
<a href="#">OpenZGY::Errors::ZgyEndOfFile</a>	Trying to read past EOF . . . . .	48
<a href="#">OpenZGY::Errors::ZgyError</a>	Base class for all exceptions thrown by OpenZGY . . . . .	49
<a href="#">OpenZGY::Errors::ZgyFormatError</a>	Corrupted or unsupported ZGY file . . . . .	50
<a href="#">OpenZGY::Errors::ZgyInternalError</a>	Exception that might be caused by a bug in OpenZGY . . . . .	50
<a href="#">OpenZGY::Errors::ZgyIoError</a>	Exception from the I/O layer . . . . .	51
<a href="#">OpenZGY::Impl::ZgyMeta</a>	High level API for reading and writing ZGY files . . . . .	52

<a href="#">OpenZGY::Impl::ZgyMetaAndTools</a>	
Add coordinate conversion to the concrete <a href="#">ZgyMeta</a> class	54
<a href="#">OpenZGY::Errors::ZgyMissingFeature</a>	
Missing feature	56
<a href="#">OpenZGY::Impl::ZgyReader</a>	
Concrete implementation of <a href="#">IZgyReader</a>	57
<a href="#">OpenZGY::Errors::ZgySegmentIsClosed</a>	
Exception used internally to request a retry	60
<a href="#">OpenZGY::Errors::ZgyUserError</a>	
Exception that might be caused by the calling application	61
<a href="#">OpenZGY::Impl::ZgyUtils</a>	
Concrete implementation of <a href="#">IZgyUtils</a>	62
<a href="#">OpenZGY::Impl::ZgyWriter</a>	
Concrete implementation of <a href="#">IZgyWriter</a>	64
<a href="#">OpenZGY::ZgyWriterArgs</a>	
Argument package for creating a ZGY file	69

## Chapter 7

# File Index

### 7.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">api.cpp</a>	Implements the pure interfaces of the API . . . . .	77
<a href="#">api.h</a>	IZgyReader, IZgyWriter, and other user visible classes . . . . .	78
<a href="#">example.h</a>	Example program using the C++ API . . . . .	81
<a href="#">exception.h</a>	Defines exceptions that may be raised by OpenZGY . . . . .	82
<a href="#">iocontext.h</a>	Backend specific context . . . . .	83



## Chapter 8

# Module Documentation

### 8.1 List of exceptions

#### Classes

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

#### 8.1.1 Detailed Description





## Chapter 9

# Namespace Documentation

### 9.1 OpenZGY Namespace Reference

The entire public API is in this namespace.

#### Namespaces

- [Errors](#)  
*Exceptions that can be thrown by [OpenZGY](#).*
- [Formatters](#)  
*operator<< for readable output of enums etc.*
- [Impl](#)  
*Implementation of the abstract interfaces in [OpenZGY](#).*

#### Classes

- class [IOContext](#)  
*Base class for backend specific context.*
- class [IZgyMeta](#)  
*Base class of [IZgyReader](#) and [IZgyWriter](#).*
- class [IZgyReader](#)  
*Main API for reading ZGY files.*
- class [IZgyTools](#)  
*Base class of [IZgyReader](#) and [IZgyWriter](#).*
- class [IZgyUtils](#)  
*Operations other than read and write.*
- class [IZgyWriter](#)  
*Main API for creating ZGY files.*
- class [ProgressWithDots](#)  
*Simple progress bar.*
- class [SampleHistogram](#)  
*Histogram of all sample values on the file.*
- class [SampleStatistics](#)  
*Statistics of all sample values on the file.*
- class [SeismicStoreIOContext](#)  
*Credentials and configuration for Seismic Store.*
- class [ZgyWriterArgs](#)  
*Argument package for creating a ZGY file.*

### 9.1.1 Detailed Description

The entire public API is in this namespace.

The main interface is [IZgyReader](#) and [IZgyWriter](#).

## 9.2 OpenZGY::Errors Namespace Reference

Exceptions that can be thrown by [OpenZGY](#).

### Classes

- class [ZgyAborted](#)  
*User aborted the computation.*
- 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 [ZgyIoError](#)  
*Exception from the I/O layer.*
- class [ZgyMissingFeature](#)  
*Missing feature.*
- class [ZgySegmentIsClosed](#)  
*Exception used internally to request a retry.*
- class [ZgyUserError](#)  
*Exception that might be caused by the calling application.*

### 9.2.1 Detailed Description

Exceptions that can be thrown by [OpenZGY](#).

## 9.3 OpenZGY::Formatters Namespace Reference

operator<< for readable output of enums etc.

## Functions

- OPENZGY\_API std::ostream & [operator<<](#) (std::ostream &os, SampleDataType value)  
*Output the string representation of the input enum type.*
- OPENZGY\_API std::ostream & [operator<<](#) (std::ostream &os, UnitDimension value)  
*Output the string representation of the input enum type.*
- OPENZGY\_API std::ostream & [operator<<](#) (std::ostream &os, DecimationType value)  
*Output the string representation of the input enum type.*
- std::string [enumToString](#) (SampleDataType value)  
*Return the string representation of the input enum type.*
- std::string [enumToString](#) (UnitDimension value)  
*Return the string representation of the input enum type.*
- std::string [enumToString](#) (DecimationType value)  
*Return the string representation of the input enum type.*

### 9.3.1 Detailed Description

[operator<<](#) for readable output of enums etc.

## 9.4 OpenZGY::Impl Namespace Reference

Implementation of the abstract interfaces in [OpenZGY](#).

## Classes

- class [ZgyMeta](#)  
*High level API for reading and writing ZGY files.*
- class [ZgyMetaAndTools](#)  
*Add coordinate conversion to the concrete [ZgyMeta](#) class.*
- class [ZgyReader](#)  
*Concrete implementation of [IZgyReader](#).*
- class [ZgyUtils](#)  
*Concrete implementation of [IZgyUtils](#).*
- class [ZgyWriter](#)  
*Concrete implementation of [IZgyWriter](#).*

### 9.4.1 Detailed Description

Implementation of the abstract interfaces in [OpenZGY](#).



## Chapter 10

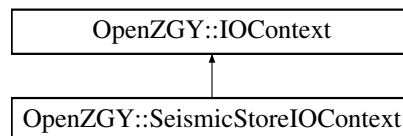
# Class Documentation

### 10.1 OpenZGY::IOContext Class Reference

Base class for backend specific context.

```
#include <iocontext.h>
```

Inheritance diagram for OpenZGY::IOContext:



#### Public Member Functions

- virtual std::string [toString](#) () const =0

#### 10.1.1 Detailed Description

Base class for backend specific context.

Thread safety: Modification may lead to a data race. This should not be an issue, because instances are only meant to be modified when created or copied or assigned prior to being made available to others.

#### 10.1.2 Member Function Documentation

### 10.1.2.1 toString()

```
virtual std::string OpenZGY::IOContext::toString ( ) const [pure virtual]
```

Display the context in a human readable format for debugging.

Implemented in [OpenZGY::SeismicStoreIOContext](#).

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

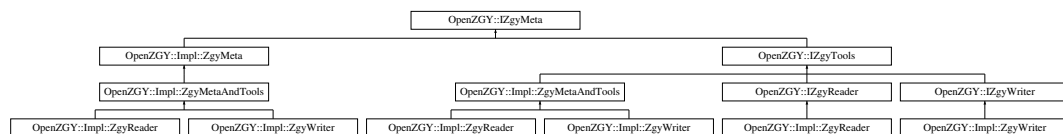
- [iocontext.h](#)

## 10.2 OpenZGY::IZgyMeta Class Reference

Base class of [IZgyReader](#) and [IZgyWriter](#).

```
#include <api.h>
```

Inheritance diagram for OpenZGY::IZgyMeta:



## Public Types

- typedef std::int8\_t **int8\_t**
- typedef std::int16\_t **int16\_t**
- typedef std::int32\_t **int32\_t**
- typedef std::int64\_t **int64\_t**
- typedef float **float32\_t**
- typedef double **float64\_t**
- typedef std::array< int64\_t, 3 > **size3i\_t**
- typedef std::array< std::array< float64\_t, 2 >, 4 > **corners\_t**
- typedef std::pair< std::shared\_ptr< const void >, std::int64\_t > **rawdata\_t**
- typedef std::function< rawdata\_t(const rawdata\_t &, const std::array< int64\_t, 3 > &> > **compressor\_t**

## Public Member Functions

- virtual size3i\_t [size](#) () const =0  
*Size in inline, crossline, vertical directions.*
- virtual SampleDataType [datatype](#) () const =0  
*Type of samples in each brick.*
- virtual std::array< float32\_t, 2 > [datarange](#) () const =0  
*Used for float to int scaling.*
- virtual std::array< float32\_t, 2 > [raw\\_datarange](#) () const =0  
*datarange before adjustment.*
- virtual UnitDimension [zunitdim](#) () const =0  
*Vertical dimension.*
- virtual UnitDimension [hunitdim](#) () const =0  
*Horizontal dimension.*
- virtual std::string [zunitname](#) () const =0  
*For annotation only. Use hunitfactor, not the name, to convert to or from SI.*
- virtual std::string [hunitname](#) () const =0  
*For annotation only. Use hunitfactor, not the name, to convert to or from SI.*
- virtual float64\_t [zunitfactor](#) () const =0  
*Multiply by this factor to convert from storage units to SI units.*
- virtual float64\_t [hunitfactor](#) () const =0  
*Multiply by this factor to convert from storage units to SI units.*
- virtual float32\_t [zstart](#) () const =0  
*First time/depth.*
- virtual float32\_t [zinc](#) () const =0  
*Increment in vertical direction.*
- virtual std::array< float32\_t, 2 > [annotstart](#) () const =0  
*First inline, crossline.*
- virtual std::array< float32\_t, 2 > [annotinc](#) () const =0  
*Increment in inline, crossline directions.*
- virtual const corners\_t & [corners](#) () const =0  
*Survey corner points in world coordinates.*
- virtual const corners\_t & [indexcorners](#) () const =0  
*Survey corner points in ordinal (i,j) coordinates.*
- virtual const corners\_t & [annotcorners](#) () const =0  
*Survey corner points in inline, crossline coordinates.*
- virtual size3i\_t [bricksize](#) () const =0  
*Size of one brick. Almost always (64,64,64), change at your own peril.*
- virtual std::vector< size3i\_t > [brickcount](#) () const =0  
*Number of bricks at each resolution (LOD) level.*
- virtual int32\_t [nlods](#) () const =0  
*Number of resolution (LOD) levels.*
- virtual void [dump](#) (std::ostream &) const =0  
*Output in human readable form for debugging.*
- virtual [SampleStatistics](#) [statistics](#) () const =0  
*Statistics of all sample values on the file.*
- virtual [SampleHistogram](#) [histogram](#) () const =0  
*Histogram of all sample values on the file.*

### 10.2.1 Detailed Description

Base class of [IZgyReader](#) and [IZgyWriter](#).

Thread safety: Interfaces do not have race conditions.

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

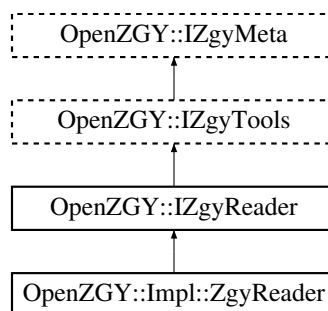
- [api.h](#)
- [api.cpp](#)

## 10.3 OpenZGY::IZgyReader Class Reference

Main API for reading ZGY files.

```
#include <api.h>
```

Inheritance diagram for OpenZGY::IZgyReader:



### Public Member Functions

- virtual void [read](#) (const size3i\_t &start, const size3i\_t &[size](#), float \*data, int lod=0) const =0  
*Read an arbitrary region.*
- virtual void [read](#) (const size3i\_t &start, const size3i\_t &[size](#), std::int16\_t \*data, int lod=0) const =0  
*Read an arbitrary region with no conversion.*
- virtual void [read](#) (const size3i\_t &start, const size3i\_t &[size](#), std::int8\_t \*data, int lod=0) const =0  
*Read an arbitrary region with no conversion.*
- virtual std::pair< bool, double > [readconst](#) (const size3i\_t &start, const size3i\_t &[size](#), int lod=0, bool as\_↔ float=true) const =0  
*Get hint about all constant region.*
- virtual void [close](#) ()=0  
*Close the file and release resources.*

### Static Public Member Functions

- static std::shared\_ptr< [IZgyReader](#) > [open](#) (const std::string &filename, const [IOContext](#) \*iocontext=nullptr)  
*Open a ZGY file for reading.*



## Additional Inherited Members

### 10.3.1 Detailed Description

Main API for reading ZGY files.

Obtain a concrete instance by calling the factory method [IZgyReader::open\(\)](#). 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.

Note: Yes, I understand that the [open\(\)](#) factory method should not have been lexically scoped inside the ostensibly pure IZgyReader interface. But this reduces the number of classes a library user needs to relate to.

Thread safety: Interfaces do not have race conditions.

### 10.3.2 Member Function Documentation

#### 10.3.2.1 close()

```
virtual void OpenZGY::IZgyReader::close ( ) [pure virtual]
```

Close the file and release resources.

Unlike [ZgyWriter::close\(\)](#), forgetting to close a file that was only open for read is not a major faux pas. It can still lead to problems though.

- The destructor of `_fd` will catch and ignore any exceptions because if a destructor throws then this will normally cause the application to crash.
- If a callback is used to refresh the token this will not happen in our destructor (it won't call `xx_close`) because it is too risky to invoke the callback this late. It might not be valid any longer. This means that if the token has expired since the last read then the close will fail. Exactly why SDAPI requires a token just to close a file is a different question. Possibly this is for removing any read locks.

Implemented in [OpenZGY::Impl::ZgyReader](#).

#### 10.3.2.2 read() [1/3]

```
virtual void OpenZGY::IZgyReader::read (
    const size3i_t & start,
    const size3i_t & size,
    float * data,
    int lod = 0 ) const [pure virtual]
```

Read an arbitrary region.

The data is read into a buffer provided by the caller. The method will apply conversion storage -> float if needed.

Data is ordered inline(slowest), crossline, vertical(fastest).

The start position refers to the specified lod level. At lod 0 start + data.size can be up to the survey size. At lod 1 the maximum is just half that, rounded up.

It is valid to pass a size that includes the padding area between the survey and the end of the current brick. But not more. In other words, the limit for lod 0 is actually `reader()->size()` rounded up to a multiple of `reader()->bricksizes()`.

Implemented in [OpenZGY::Impl::ZgyReader](#).

### 10.3.2.3 read() [2/3]

```
virtual void OpenZGY::IZgyReader::read (
    const size3i_t & start,
    const size3i_t & size,
    std::int16_t * data,
    int lod = 0 ) const [pure virtual]
```

Read an arbitrary region with no conversion.

As the read overload with a float buffer but only works for files with SampleDataType::int16 and does not scale the samples.

Implemented in [OpenZGY::Impl::ZgyReader](#).

### 10.3.2.4 read() [3/3]

```
virtual void OpenZGY::IZgyReader::read (
    const size3i_t & start,
    const size3i_t & size,
    std::int8_t * data,
    int lod = 0 ) const [pure virtual]
```

Read an arbitrary region with no conversion.

As the read overload with a float buffer but only works for files with SampleDataType::int8 and does not scale the samples.

Implemented in [OpenZGY::Impl::ZgyReader](#).

### 10.3.2.5 readconst()

```
virtual std::pair<bool,double> OpenZGY::IZgyReader::readconst (
    const size3i_t & start,
    const size3i_t & size,
    int lod = 0,
    bool as_float = true ) const [pure virtual]
```

Get hint about all constant region.

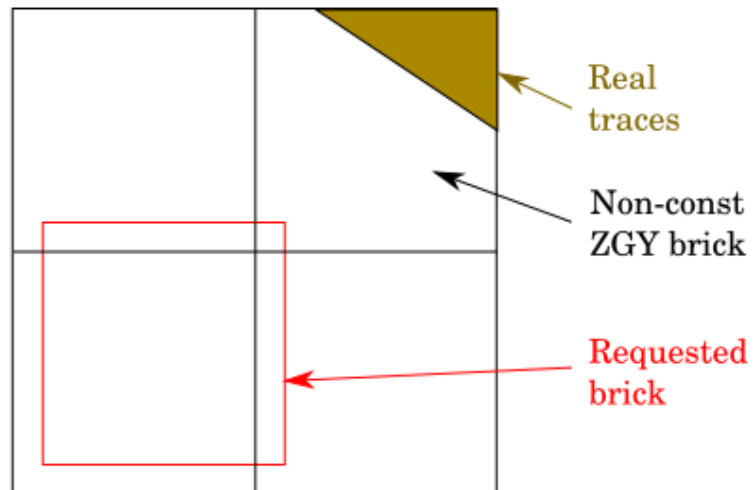
Check to see if the specified region is known to have all samples set to the same value. Returns a pair of (is\_const, const\_value). If it returns is\_const=true you know what the region contains; otherwise you need to use the regular [read\(\)](#) method.

For int8 and int16 files the caller may specify whether to scale the values or not. Even if unscaled the function returns the value as a double.

The function only makes inexpensive checks so it might return is\_const=false 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. The following figure might clarify how it works.

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

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



If the region requested in `readconst()` is exactly one ZGY brick then this just checks whether that brick is known to be constant. If called with a larger, arbitrary region it fails unless all bricks in that region are known to be constant and all have the same value. It will not tell you which brick(s) were not const. If you need to know then you must loop over each ZGY brick and make multiple calls to `readconst()`.

Implemented in [OpenZGY::Impl::ZgyReader](#).

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

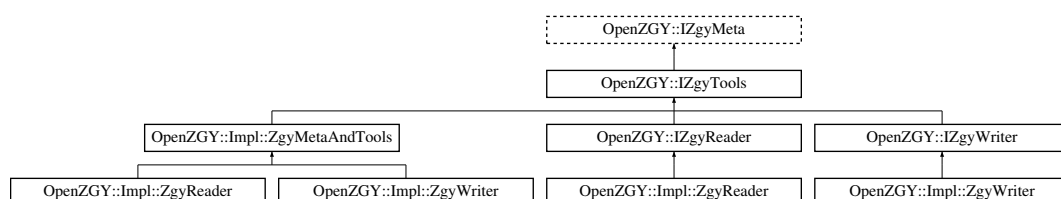
- [api.h](#)
- [api.cpp](#)

## 10.4 OpenZGY::IZgyTools Class Reference

Base class of [IZgyReader](#) and [IZgyWriter](#).

```
#include <api.h>
```

Inheritance diagram for OpenZGY::IZgyTools:



## Public Member Functions

- virtual void [transform](#) (const corners\_t &from, const corners\_t &to, std::vector< std::array< float64\_t, 2 >> &) const =0  
*General coordinate conversion of an array of points. (NOT IMPLEMENTED YET)*
- virtual std::array< float64\_t, 2 > [transform1](#) (const corners\_t &from, const corners\_t &to, const std::array< float64\_t, 2 > &) const =0  
*General coordinate conversion of a single coordinate pair.*
- virtual std::array< float64\_t, 2 > [annotToIndex](#) (const std::array< float64\_t, 2 > &) const =0  
*Convert a single coordinate pair.*
- virtual std::array< float64\_t, 2 > [annotToWorld](#) (const std::array< float64\_t, 2 > &) const =0  
*Convert a single coordinate pair.*
- virtual std::array< float64\_t, 2 > [indexToAnnot](#) (const std::array< float64\_t, 2 > &) const =0  
*Convert a single coordinate pair.*
- virtual std::array< float64\_t, 2 > [indexToWorld](#) (const std::array< float64\_t, 2 > &) const =0  
*Convert a single coordinate pair.*
- virtual std::array< float64\_t, 2 > [worldToAnnot](#) (const std::array< float64\_t, 2 > &) const =0  
*Convert a single coordinate pair.*
- virtual std::array< float64\_t, 2 > [worldToIndex](#) (const std::array< float64\_t, 2 > &) const =0  
*Convert a single coordinate pair.*

## Additional Inherited Members

### 10.4.1 Detailed Description

Base class of [IZgyReader](#) and [IZgyWriter](#).

Thread safety: Interfaces do not have race conditions.

### 10.4.2 Member Function Documentation

#### 10.4.2.1 transform()

```
virtual void OpenZGY::IZgyTools::transform (
    const corners_t & from,
    const corners_t & to,
    std::vector< std::array< float64_t, 2 >> & ) const [pure virtual]
```

General coordinate conversion of an array of points. (NOT IMPLEMENTED YET)

#### Parameters

<i>from</i>	control points in the current coordinate system.
<i>to</i>	control points in the desired coordinate system.

Convert coordinates in place, with the conversion defined by giving the values of 3 arbitrary control points in both

the "from" and "to" coordinate system. A common choice of arbitrary points is to use three of the lattice corners. As a convenience the "from" and "to" parameters are declared as `corners_t` so the caller can pass [corners\(\)](#), [annotcorners\(\)](#), or [indexcorners\(\)](#) directly.

Implemented in [OpenZGY::Impl::ZgyMetaAndTools](#).

#### 10.4.2.2 transform1()

```
virtual std::array<float64_t,2> OpenZGY::IZgyTools::transform1 (
    const corners_t & from,
    const corners_t & to,
    const std::array< float64_t, 2 > & ) const [pure virtual]
```

General coordinate conversion of a single coordinate pair.

##### Parameters

<i>from</i>	control points in the current coordinate system.
<i>to</i>	control points in the desired coordinate system.

Convert coordinates in place, with the conversion defined by giving the values of 3 arbitrary control points in both the "from" and "to" coordinate system. A common choice of arbitrary points is to use three of the lattice corners. As a convenience the "from" and "to" parameters are declared as `corners_t` so the caller can pass [corners\(\)](#), [annotcorners\(\)](#), or [indexcorners\(\)](#) directly.

Implemented in [OpenZGY::Impl::ZgyMetaAndTools](#).

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

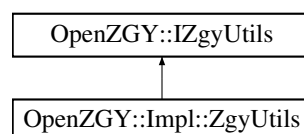
- [api.h](#)
- [api.cpp](#)

## 10.5 OpenZGY::IZgyUtils Class Reference

Operations other than read and write.

```
#include <api.h>
```

Inheritance diagram for OpenZGY::IZgyUtils:



## Public Member Functions

- virtual void [deletefile](#) (const std::string &filename, bool missing\_ok=true)=0  
*Delete a file. Works both for local and cloud files.*
- virtual std::string [alturl](#) (const std::string &filename)=0  
*Return a url that might be opened faster.*

## Static Public Member Functions

- static std::shared\_ptr< [IZgyUtils](#) > [utils](#) (const std::string &prefix, const [IOContext](#) \*iocontext)  
*Create a new concrete instance of [IZgyUtils](#).*

### 10.5.1 Detailed Description

Operations other than read and write.

Any operations that don't fit into [IZgyReader](#) or [IZgyWriter](#) go here. Such as deleting a file. Or any other operation that does not need the file to be open first.

Thread safety: Interfaces do not have race conditions.

### 10.5.2 Member Function Documentation

#### 10.5.2.1 alturl()

```
virtual std::string OpenZGY::IZgyUtils::alturl (
    const std::string & filename ) [pure virtual]
```

Return a url that might be opened faster.

Using this method might help performance in a distributed system where a large number of processing nodes need to open the same file for read. One node would call this method which costs about the same as opening the file locally. All other nodes can then use the alternate url which opens a lot faster due to the meta data having already been retrieved and serialized in the alturl.

The alternate url is valid for a limited time only. Typically it expires when the current access token expires but it might expire earlier.

Note that the alternate url might contain secrets, so it needs to be protected just as you would protect an access token.

The method is not useful for local files where the file name will just be returned unmodified. The method is unlikely to be useful in a single-node application. In that case the application can simply keep the file open instead of keeping the alturl reference.

Implemented in [OpenZGY::Impl::ZgyUtils](#).

### 10.5.2.2 deletefile()

```
virtual void OpenZGY::IZgyUtils::deletefile (
    const std::string & filename,
    bool missing_ok = true ) [pure virtual]
```

Delete a file. Works both for local and cloud files.

Note that the instance must be of the correct (local or cloud) type.

Implemented in [OpenZGY::Impl::ZgyUtils](#).

### 10.5.2.3 utils()

```
std::shared_ptr< IZgyUtils > OpenZGY::IZgyUtils::utils (
    const std::string & prefix,
    const IOContext * iocontext ) [static]
```

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

#### Parameters

<i>prefix</i>	File name or file name prefix.
<i>iocontext</i>	Credentials and other configuration.

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 both "sd://some/bogus/file.zgy" and just "sd://" will produce an instance that works for the seismic store.

For performance reasons you should consider caching one [IZgyUtils](#) 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.

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

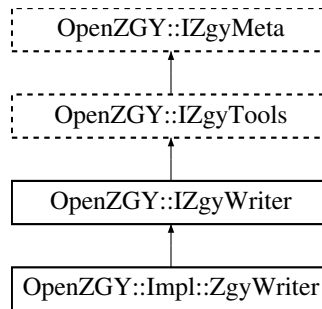
- [api.h](#)
- [api.cpp](#)

## 10.6 OpenZGY::IZgyWriter Class Reference

Main API for creating ZGY files.

```
#include <api.h>
```

Inheritance diagram for OpenZGY::IZgyWriter:



## Public Member Functions

- virtual void [write](#) (const size3i\_t &start, const size3i\_t &size, const float \*data)=0  
*Write an arbitrary region.*
- virtual void [write](#) (const size3i\_t &start, const size3i\_t &size, const std::int16\_t \*data)=0  
*Write an arbitrary region with no conversion.*
- virtual void [write](#) (const size3i\_t &start, const size3i\_t &size, const std::int8\_t \*data)=0  
*Write an arbitrary region with no conversion.*
- virtual void [writeconst](#) (const size3i\_t &start, const size3i\_t &size, const float \*data)=0  
*Write all-constant data.*
- virtual void [writeconst](#) (const size3i\_t &start, const size3i\_t &size, const std::int16\_t \*data)=0  
*Write an arbitrary region with no conversion.*
- virtual void [writeconst](#) (const size3i\_t &start, const size3i\_t &size, const std::int8\_t \*data)=0  
*Write an arbitrary region with no conversion.*
- virtual void [finalize](#) (const std::vector< DecimationType > &decimation=std::vector< DecimationType >(), const std::function< bool(std::int64\_t, std::int64\_t)> &progress=nullptr, bool force=false)=0  
*Generate low resolution data, statistics, and histogram.*
- virtual void [close\\_incomplete](#) ()=0  
*Flush the file to disk and close it.*
- virtual void [close](#) ()=0  
*Flush the file to disk and close it.*

## Static Public Member Functions

- static std::shared\_ptr< [IZgyWriter](#) > [open](#) (const [ZgyWriterArgs](#) &args)  
*Create a ZGY file and open it for writing.*
- static std::shared\_ptr< [IZgyWriter](#) > [reopen](#) (const [ZgyWriterArgs](#) &args)  
*Open an existing ZGY file for writing.*

## Additional Inherited Members

### 10.6.1 Detailed Description

Main API for creating ZGY files.

Obtain a concrete instance by calling the factory method [IZgyWriter::open\(\)](#). 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\(\)](#) after all bulk has been written. but if you forget this will be called from [close\(\)](#) with default arguments. It is required to explicitly [close\(\)](#) the file when done with it, Technically the destructor will call [close\(\)](#) but being a destructor it needs to swallow any exceptions.

Note: Yes, I understand that the [open\(\)](#) factory method should not have been lexically scoped inside the ostensibly pure [IZgyWriter](#) interface. But this reduces the number of classes a library user needs to relate to.

Thread safety: Interfaces do not have race conditions.



## 10.6.2 Member Function Documentation

### 10.6.2.1 close()

```
virtual void OpenZGY::IZgyWriter::close ( ) [pure virtual]
```

Flush the file to disk and close it.

If the file has been written to, the application is encouraged to call [finalize\(\)](#) before [close\(\)](#). This gives more control over the process and allows using a progress callback to track generation of low resolution data.

The function won't bother with statistics, histogram, lowres if there has been an unrecoverable error. The headers might still be written out in case somebody wants to try some forensics.

The ZgyWriter destructor will call [close\(\)](#) if not done already, but that will catch and swallow any exception. Relying on the destructor to close the file is strongly discouraged.

Implemented in [OpenZGY::Impl::ZgyWriter](#).

### 10.6.2.2 close\_incomplete()

```
virtual void OpenZGY::IZgyWriter::close_incomplete ( ) [pure virtual]
```

Flush the file to disk and close it.

This version of [close\(\)](#) will not calculate statistics and low resolution bricks. Currently this makes the file useless in most cases. The function may be useful for performance measurements.

In the future it might be possible to re-open the file at some later date and continue writing data to it. Calling the regular [close\(\)](#) only when all data has been output.

Implemented in [OpenZGY::Impl::ZgyWriter](#).

### 10.6.2.3 finalize()

```
virtual void OpenZGY::IZgyWriter::finalize (
    const std::vector< DecimationType > & decimation = std::vector< DecimationType >(),
    const std::function< bool(std::int64_t, std::int64_t)> & progress = nullptr,
    bool force = false ) [pure virtual]
```

Generate low resolution data, statistics, and histogram.

This method 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.

The C++ code is very different from Python because it needs an entirely different approach to be performant.

## Parameters

<i>decimation</i>	Optionally override the decimation algorithms by passing an array of DecimationType with one entry for each level of detail. If the array is too short then the last entry is used for subsequent levels.
<i>progress</i>	Function(done, total) called to report progress. If it returns False the computation is aborted. The callback implementation should not invoke any <a href="#">OpenZGY</a> methods except for any calls expressly documented as safe for this purpose and deadlock-free. Will be called at least one, even if there is no work.
<i>force</i>	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. passing force=true is currently required if cerating an empty file, no samples written at all, if the statistics and histogram data should still be stored.

Implemented in [OpenZGY::Impl::ZgyWriter](#).

#### 10.6.2.4 reopen()

```
std::shared_ptr< IZgyWriter > OpenZGY::IZgyWriter::reopen (
    const ZgyWriterArgs & args ) [static]
```

Open an existing ZGY file for writing.

There are several restrictions on using this feature. Some of those might be relaxed in the future.

- The file must have the latest version. Currently not enforced.
- size, bricksize, datatype, and datarange are not allowed to change.
- filename, iocontext, compressor, and lodcompressor are not stored in the file itself. So the caller needs to re-specify them.
- ilstart, ilinc, xstart, xline, zstart, zinc, zunit, hunt, corners can all be changed but note that this can only be done in the actual call to [reopen\(\)](#).
- The operation is expensive, even when the only change is to update some metadata. The current implementation will delete and re-create the file behind the scenes.
- The file to be opened cannot contain any bulk data. The only exception is that the entire file may have been initialized to some arbitrary constant.
- If the file to be updated is stored on a cloud back-end then it must have been created by [OpenZGY](#) and written directly to the cloud. The reason: Even though the ZGY-Public and [OpenZGY](#) file formats are identical, the data on the cloud can be split into multiple chunks which are logically treated as a single byte stream. The choice of where to split differs between [OpenZGY](#), ZGY-Public, and uploads done by sduil. This restriction is currently not active but it will become active when a better implementation (removing most of the other rules) has been coded.

- If the file does contain bulk data (which currently isn't allowed) then it is strongly advised to not change from compressed to uncompressed or vice versa. Even though this might not be checked. If you ignore this advice then the new compression mode will only apply to subsequent writes.
- If the file does not contain bulk data but is flagged as v4 anyway, it is ok to reopen it as uncompressed. In that case it is unspecified whether the version is changed back from 4 to 3 (which is the current behavior) or not.
- If the file does contain bulk data (which currently isn't allowed) then it is strongly advised to not update any already existing data. Including overwrite due to a read/modify/write cycle. With a cloud backend this could waste storage and might also be refused at runtime.
- A file that has never been written to may end up as a file that has been explicitly written as the default (usually zero) value. The [OpenZGY](#) api hides that distinction anyway. So it shouldn't matter.
- The operation can leave the file unusable or even deleted if some error occurs. Since we are creating a brand new file it would be possible to create this with a new name and only rename it and discard the original once we know that the new file is good. But, (a) rename on the cloud is difficult and (b) once this code gets implemented properly that trick isn't possible anyway.

#### 10.6.2.5 write() [1/3]

```
virtual void OpenZGY::IZgyWriter::write (
    const size3i_t & start,
    const size3i_t & size,
    const float * data ) [pure virtual]
```

Write an arbitrary region.

This will apply conversion float -> storage if needed.

Data is ordered inline(slowest), crossline, vertical(fastest).

A read/modify/write will be done if the region's start and size doesn't align with bricksize. When writing to the cloud this read/modify/write may incur performance and size penalties. So do write brick aligned data if possible. The same applies to writing compressed data where r/m/w can cause a severe loss of quality.

The start position refers to the specified lod level. At lod 0 start + data.size can be up to the survey size. At lod 1 the maximum is just half that, rounded up.

Implemented in [OpenZGY::Impl::ZgyWriter](#).

#### 10.6.2.6 write() [2/3]

```
virtual void OpenZGY::IZgyWriter::write (
    const size3i_t & start,
    const size3i_t & size,
    const std::int16_t * data ) [pure virtual]
```

Write an arbitrary region with no conversion.

As the write overload with a float buffer but only works for files with SampleDataType::int16 and does not scale the samples.

Implemented in [OpenZGY::Impl::ZgyWriter](#).

### 10.6.2.7 write() [3/3]

```
virtual void OpenZGY::IZgyWriter::write (
    const size3i_t & start,
    const size3i_t & size,
    const std::int8_t * data ) [pure virtual]
```

Write an arbitrary region with no conversion.

As the write overload with a float buffer but only works for files with SampleDataType::int8 and does not scale the samples.

Implemented in [OpenZGY::Impl::ZgyWriter](#).

### 10.6.2.8 writeconst() [1/3]

```
virtual void OpenZGY::IZgyWriter::writeconst (
    const size3i_t & start,
    const size3i_t & size,
    const float * data ) [pure virtual]
```

Write all-constant data.

Works as the corresponding write but the entire region is set to the same value. So the provided data buffer needs just one value, or alternatively can be passed as &scalar\_value.

Calling this method is faster than filling a buffer with constant values and calling write. But it produces the exact same result. This is because write will automatically detect whether the input buffer is all constant.

Implemented in [OpenZGY::Impl::ZgyWriter](#).

### 10.6.2.9 writeconst() [2/3]

```
virtual void OpenZGY::IZgyWriter::writeconst (
    const size3i_t & start,
    const size3i_t & size,
    const std::int16_t * data ) [pure virtual]
```

Write an arbitrary region with no conversion.

As the writeconst overload with a float buffer but only works for files with SampleDataType::int16 and does not scale the samples.

Implemented in [OpenZGY::Impl::ZgyWriter](#).

**10.6.2.10 writeconst() [3/3]**

```
virtual void OpenZGY::IZgyWriter::writeconst (
    const size3i_t & start,
    const size3i_t & size,
    const std::int8_t * data ) [pure virtual]
```

Write an arbitrary region with no conversion.

As the write overload with a float buffer but only works for files with SampleDataType::int8 and does not scale the samples.

Implemented in [OpenZGY::Impl::ZgyWriter](#).

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

- [api.h](#)
- [api.cpp](#)

**10.7 OpenZGY::ProgressWithDots Class Reference**

Simple progress bar.

```
#include <api.h>
```

**Public Member Functions**

- [ProgressWithDots](#) (int length=51, std::ostream &outfile=std::cerr)  
*Simple progress bar.*
- bool [operator\(\)](#) (std::int64\_t done, std::int64\_t total)  
*Callback invoked to report progress.*

**10.7.1 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.

When passed as the second parameter to [IZgyWriter::finalize\(\)](#) you may need to pass it as std::ref(p) since the instance is noncopyable. finalize() expects a reference to a function, but apparently the compiler needs to copy the instance anyway.

Thread safety: Protected by a mutex.

TODO-Low: Add a pimpl pattern to avoid needing the <mutex> header. I really want to keep the include count low in this main header file.

## 10.7.2 Constructor & Destructor Documentation

### 10.7.2.1 ProgressWithDots()

```
OpenZGY::ProgressWithDots::ProgressWithDots (
    int length = 51,
    std::ostream & outfile = std::cerr )
```

Simple progress bar.

#### Parameters

<i>length</i>	Size of progress bar, default 51 dots.
<i>outfile</i>	Stream to write output, default std::cerr

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.

When passed as the second parameter to [IZgyWriter::finalize\(\)](#) you may need to pass it as std::ref(p) since the instance is noncopyable. finalize() expects a reference to a function, but apparently the compiler needs to copy the instance anyway.

Thread safety: Protected by a mutex.

TODO-Low: Add a pimpl pattern to avoid needing the <mutex> header. I really want to keep the include count low in this main header file.

## 10.7.3 Member Function Documentation

### 10.7.3.1 operator()()

```
bool OpenZGY::ProgressWithDots::operator() (
    std::int64_t done,
    std::int64_t total )
```

Callback invoked to report progress.

## Parameters

<i>done</i>	Number of work units done.
<i>total</i>	Number of work units in total.

The callback will normally get called exactly once with `done==0`, before processing starts but after "total" is known. And exactly once with `done==total` signifying that the work is finished and the function being monitored should soon return.

This particular callback will always return true, meaning that the operation is not to be aborted.

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

- [api.h](#)
- [api.cpp](#)

## 10.8 OpenZGY::SampleHistogram Class Reference

Histogram of all sample values on the file.

```
#include <api.h>
```

### Public Member Functions

- [SampleHistogram](#) (std::int64\_t samplecount\_in, double minvalue\_in, double maxvalue\_in, const std::vector< std::int64\_t > &bins\_in)

*Create a new instance with all contents filled in.*

### Public Attributes

- std::int64\_t [samplecount](#)  
*Sum of counts of all bins.*
- double [minvalue](#)  
*Center value of data in first bin.*
- double [maxvalue](#)  
*Center value of data in last bin.*
- std::vector< std::int64\_t > [bins](#)  
*array of counts by bin*

### 10.8.1 Detailed Description

Histogram of all sample values on the file.

The histogram is described by the fixed total number of bins, the center value of the samples in the first bin, and the center value of the samples in the last bin.

The width of each bin is given by  $(\text{max} - \text{min}) / (\text{nbins} - 1)$ . Bin 0 holds samples with values  $\text{min}_\pm \pm \text{binwidth}/2$ .

This means that the total range of samples that can be represented in the histogram is actually  $\text{min} - \text{binwidth}/2$  to  $\text{max} + \text{binwidth}/2$ , which is slightly larger than just  $\text{min}.. \text{max}$  *unless* each of the bins can only hold a single value (e.g. data converted from 8-bit storage, in 256 bins).

This definition has some subtle effects, best illustrated by a few examples.

If the source data is `ui8_t`, the logical range is `0..255`. Assume `nbins=256`. This gives `binwidth=1`, and the true range of the histogram `-0.5..+255.5`. But since the input is integral, the actual range is just `0..255` inclusive. Try to fill the histogram with evenly distributed random data and you end up with each bin having roughly the same number of elements.

Now consider `ui16_t`, range `0..65535` and `nbins` is still 256. This gives `binwidth=257`, not 256. The true range of the histogram is `-128.5..+65663.5`. Try to fill the histogram with evenly distributed random data and you end up with the first and the last bin having approximately half as many elements as all the others. This is not really a problem, but may seem a bit surprising.

The range should have the zero-centric property. Zero, if present in the range, should map to the center of a bin. Otherwise the histogram close to zero might look odd.

Thread safety: Modification may lead to a data race. This should not be an issue, because instances are only meant to be modified when created or copied or assigned prior to being made available to others.

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

- [api.h](#)

## 10.9 OpenZGY::SampleStatistics Class Reference

Statistics of all sample values on the file.

```
#include <api.h>
```

### Public Member Functions

- [SampleStatistics](#) (`std::int64_t cnt_in, double sum_in, double ssq_in, double min_in, double max_in`)  
*Create a new instance with all contents filled in.*



## Public Attributes

- `std::int64_t cnt`  
*Number of added samples.*
- `double sum`  
*Sum of added samples.*
- `double ssq`  
*Sum-of-squares of added samples.*
- `double min`  
*Minimum added sample value.*
- `double max`  
*Maximum added sample value.*

### 10.9.1 Detailed Description

Statistics of all sample values on the file.

Thread safety: Modification may lead to a data race. This should not be an issue, because instances are only meant to be modified when created or copied or assigned prior to being made available to others.

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

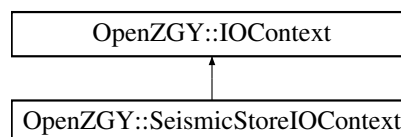
- [api.h](#)

## 10.10 OpenZGY::SeismicStoreIOContext Class Reference

Credentials and configuration for Seismic Store.

```
#include <iocontext.h>
```

Inheritance diagram for OpenZGY::SeismicStoreIOContext:



## Public Types

- `typedef std::function< std::string()> tokencb_t`
- `typedef std::function< void(const std::string &, std::int64_t, std::int64_t, std::int64_t, const std::vector< std::int64_t > &)> debugtrace_t`

## Public Member Functions

- virtual std::string [toString](#) () const override
- [SeismicStoreIOContext](#) & [sdurl](#) (const std::string &value)
- [SeismicStoreIOContext](#) & [sdapikey](#) (const std::string &value)
- [SeismicStoreIOContext](#) & [sdtoken](#) (const std::string &value, const std::string &type)
- [SeismicStoreIOContext](#) & [sdtokencb](#) (const tokencb\_t &value, const std::string &type)
- [SeismicStoreIOContext](#) & [maxsize](#) (int value)
- [SeismicStoreIOContext](#) & [maxhole](#) (int value)
- [SeismicStoreIOContext](#) & [aligned](#) (int value)
- [SeismicStoreIOContext](#) & [segsizes](#) (int value)
- [SeismicStoreIOContext](#) & [segsplit](#) (int value)
- [SeismicStoreIOContext](#) & [threads](#) (int value)
- [SeismicStoreIOContext](#) & [legaltag](#) (const std::string &value)
- [SeismicStoreIOContext](#) & [writeid](#) (const std::string &value)
- [SeismicStoreIOContext](#) & [seismicmeta](#) (const std::string &value)
- [SeismicStoreIOContext](#) & [debug\\_trace](#) (const debugtrace\_t &value)

### 10.10.1 Detailed Description

Credentials and configuration for Seismic Store.

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

TODO-Low: Still undecided whether I should allow SeismicStoreFile to use this class directly or whether I should map the contents to an internal SDConfig class.

TODO-Low: Move this class to a separate extensions/seismic\_store.h to be included by applications if and only if they need that access.

Thread safety: Modification may lead to a data race. This should not be an issue, because instances are only meant to be modified when created or copied or assigned prior to being made available to others.

### 10.10.2 Member Function Documentation

#### 10.10.2.1 aligned()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::aligned (
    int value )
```

File alignment, in MB. Must be 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.

### 10.10.2.2 debug\_trace()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::debug_trace (
    const debugtrace_t & value )
```

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.

The callback implementation should not invoke any [OpenZGY](#) methods except for any calls expressly documented as safe for this purpose and deadlock-free.

### 10.10.2.3 legaltag()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::legaltag (
    const std::string & value )
```

The legaltag stored in the file. Used only on create.

### 10.10.2.4 maxhole()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::maxhole (
    int value )
```

Maximum size to waste, in MB. Must be 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.

### 10.10.2.5 maxsize()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::maxsize (
    int value )
```

Maximum size of consolidated requests, in MB. Must be 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.

### 10.10.2.6 sdapikey()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::sdapikey (
    const std::string & value )
```

Authorization for application to access the seismic store API. Defaults to \$OPENZGY\_SDAPIKEY. There is no hard coded fallback in case that variable isn't found either. This is to mitigate the risk of code stopping to work in the PROD environment.

### 10.10.2.7 sdtoken()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::sdtoken (
    const std::string & value,
    const std::string & type )
```

Provide the SAuth token used to validate requests to seismic store. The token is associated with the open file and cannot be changed. tokentype is currently ignored, but may in the future be specified as e.g. "stoken" (normal), "imptoken" (impersonation), etc. Currently the token will (usually) not be automatically refreshed. If you need this or need more detailed control then you should use sdtokenCb() instead of [sdtoken\(\)](#).

If neither [sdtoken\(\)](#) nor sdtokenCb() are called then the environment variable "OPENZGY\_TOKEN" is tried. Older versions of the accessor had a final fallback that would try to pick up sdutil's saved credentials but that is now considered too insecure. You can set OPENZGY\_TOKEN=FILE:carbon.slbapp.com if you want that behavior.

### 10.10.2.8 sdtokencb()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::sdtokencb (
    const tokencb_t & value,
    const std::string & type )
```

Register a callback that will be invoked each time an access token is needed for seismic store. This is an alternative to [sdtoken\(\)](#). You don't need both. The callback might be called very frequently so it should cache the token. The callback should ensure that the token is not about to expire, and refresh it if needed. The optional tokentype is currently unused. It has the same meaning as in [sdtoken\(\)](#). Note that the callback itself does not provide the token type. You need to specify the type in this call. Promising that whenever the callback returns a token it will be of this type.

The callback implementation should not invoke any [OpenZGY](#) methods except for any calls expressly documented as safe and deadlock-free. The callback implementation must also be thread safe. Set a lock if needed. Finally, the callback needs to be legal to call at least until close() is called on the reader or writer instance and preferably also until the instance goes out of scope.

The library will try to close the reader or writer instance if the application fails to do so before the instance goes out of scope. E.g. so any locks may be reset. The library *tries* to avoid invoking the callback in that situation. On the suspicion that it might have gone out of scope. But the application code should not rely on that.

### 10.10.2.9 sdurl()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::sdurl (
    const std::string & value )
```

Where to contact the seismic store service. Defaults to \$OPENZGY\_SDURL. There is no hard coded fallback in case that variable isn't found either. This is to mitigate the risk of code stopping to work in the PROD environment.

**10.10.2.10 segsize()**

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::segsize (
    int value )
```

Segment size used when writing, in MB. Must be between 1 and 16\*1024 (i.e. 16 GB). Defaults to \$OPENZGY\_SEGSIZE\_MB if not specified, or 256 MB. The default should work fine in almost all cases. The write buffer needs to allocate segsize \* segsplit bytes, so make sure to take segsplit into account when deciding on the segment size. With both of those left at the default value the buffer will need 256 MB \* 8 = 2 GB.

**10.10.2.11 segsplit()**

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::segsplit (
    int value )
```

Maximum number of threads to be used when writing data to the cloud. Set to 1 if you don't want multithreaded uploads. Default is 8.

Multi-threading is achieved by using an internal buffer size of segsize\*segsplit, then later splitting that up into separate SDAPI objects than can be uploaded in parallel. So, beware that if you increase segsplit you might need to decrease segsize to avoid running out of memory.

Using multiple threads for uploads can often improve throughput. But if the limiting factor is the bandwidth between the client and the cloud server then multiple threads won't help.

**10.10.2.12 seismicmeta()**

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::seismicmeta (
    const std::string & value )
```

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(). This setting cannot be set from the environment.

**10.10.2.13 threads()**

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::threads (
    int value )
```

Use up to this many parallel requests to seismic store in order to speed up processing. Set between 1 and 1024, 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.

#### 10.10.2.14 toString()

```
virtual std::string OpenZGY::SeismicStoreIOContext::toString ( ) const [override], [virtual]
```

Display the context in a human readable format for debugging.

Implements [OpenZGY::IOContext](#).

#### 10.10.2.15 writeid()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::writeid (
    const std::string & value )
```

If set, re-use this lock instead of creating a new one. Works both for read and write locks; the name reflects what SDAPI calls it.

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

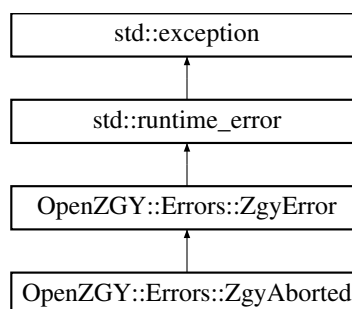
- [iocontext.h](#)

## 10.11 OpenZGY::Errors::ZgyAborted Class Reference

User aborted the computation.

```
#include <exception.h>
```

Inheritance diagram for OpenZGY::Errors::ZgyAborted:



### Public Member Functions

- [ZgyAborted](#) (const std::string &arg)  
*User aborted the computation.*

## Additional Inherited Members

### 10.11.1 Detailed Description

User aborted the computation.

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.

Thread safety: Exceptions defined in this module are safe.

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

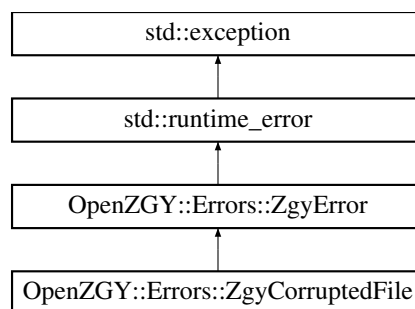
- [exception.h](#)

## 10.12 OpenZGY::Errors::ZgyCorruptedFile Class Reference

The ZGY file became corrupted while writing to it.

```
#include <exception.h>
```

Inheritance diagram for OpenZGY::Errors::ZgyCorruptedFile:



## Public Member Functions

- [ZgyCorruptedFile](#) (const std::string &arg)  
*The ZGY file became corrupted while writing to it.*

## Additional Inherited Members

### 10.12.1 Detailed Description

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.

Thread safety: Exceptions defined in this module are safe.

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

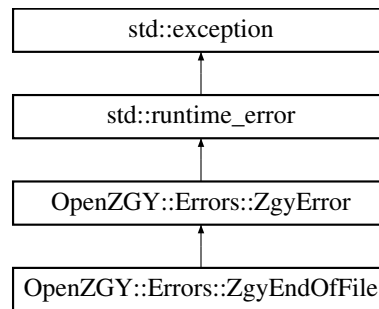
- [exception.h](#)

## 10.13 OpenZGY::Errors::ZgyEndOfFile Class Reference

Trying to read past EOF.

```
#include <exception.h>
```

Inheritance diagram for OpenZGY::Errors::ZgyEndOfFile:



### Public Member Functions

- [ZgyEndOfFile](#) (const std::string &arg)

*Trying to read past EOF.*

### Additional Inherited Members

#### 10.13.1 Detailed Description

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](#).

Thread safety: Exceptions defined in this module are safe.

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

- [exception.h](#)

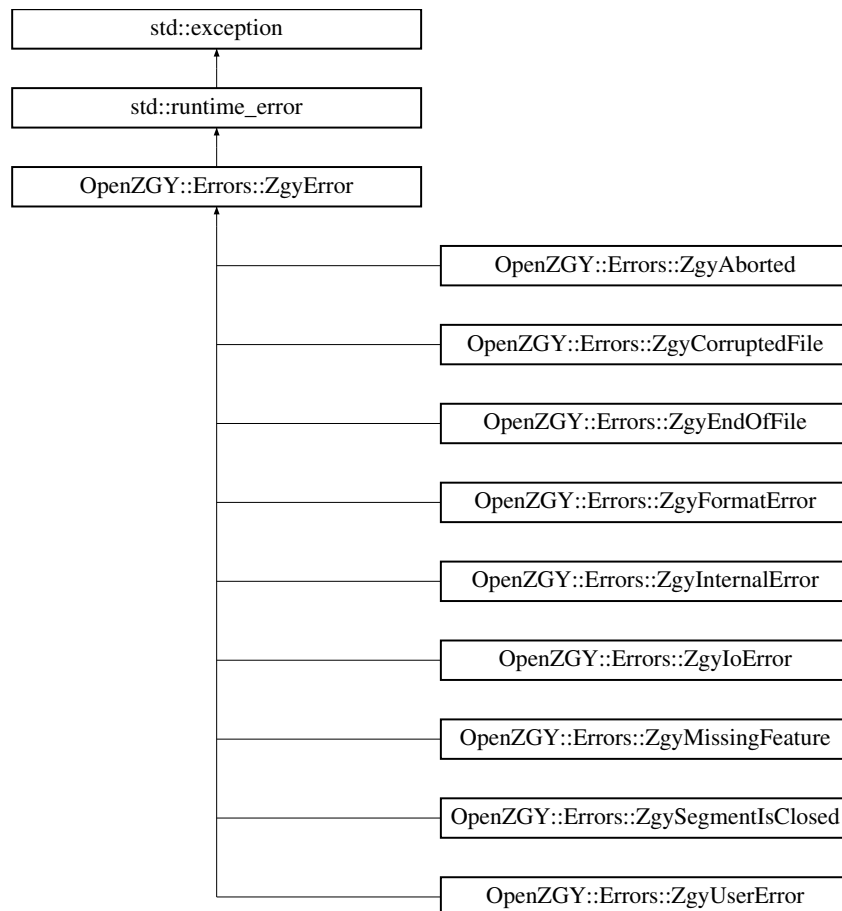


## 10.14 OpenZGY::Errors::ZgyError Class Reference

Base class for all exceptions thrown by OpenZGY.

```
#include <exception.h>
```

Inheritance diagram for OpenZGY::Errors::ZgyError:



### Protected Member Functions

- [ZgyError](#) (const std::string &arg)

*Base class for all exceptions thrown by OpenZGY.*

#### 10.14.1 Detailed Description

Base class for all exceptions thrown by OpenZGY.

Thread safety: Exceptions defined in this module are safe.

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

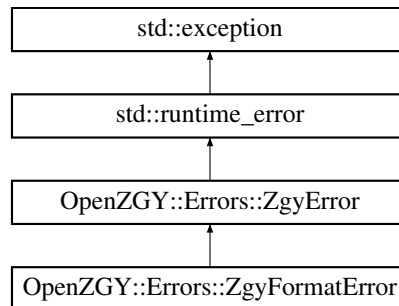
- [exception.h](#)

## 10.15 OpenZGY::Errors::ZgyFormatError Class Reference

Corrupted or unsupported ZGY file.

```
#include <exception.h>
```

Inheritance diagram for OpenZGY::Errors::ZgyFormatError:



### Public Member Functions

- [ZgyFormatError](#) (const std::string &arg)  
*Corrupted or unsupported ZGY file.*

### Additional Inherited Members

#### 10.15.1 Detailed Description

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.

Thread safety: Exceptions defined in this module are safe.

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

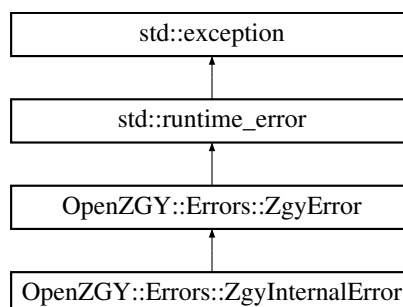
- [exception.h](#)

## 10.16 OpenZGY::Errors::ZgyInternalError Class Reference

Exception that might be caused by a bug in OpenZGY.

```
#include <exception.h>
```

Inheritance diagram for OpenZGY::Errors::ZgyInternalError:



## Public Member Functions

- [ZgyInternalError](#) (const std::string &arg)  
*Exception that might be caused by a bug in OpenZGY.*

## Additional Inherited Members

### 10.16.1 Detailed Description

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](#).

Thread safety: Exceptions defined in this module are safe.

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

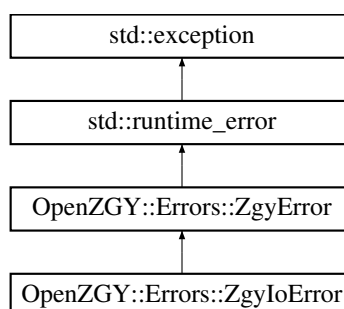
- [exception.h](#)

## 10.17 OpenZGY::Errors::ZgyloError Class Reference

Exception from the I/O layer.

```
#include <exception.h>
```

Inheritance diagram for OpenZGY::Errors::ZgyloError:



## Public Member Functions

- [ZgyloError](#) (const std::string &filename, int system\_errno)  
*Exception from the I/O layer.*

## Additional Inherited Members

### 10.17.1 Detailed Description

Exception from the I/O layer.

Some error was received from a linux syscall acting on a file. For cloud access the actual exception thrown by the back end might be reported instead of wrapping it into a [ZgyIoError](#).

Thread safety: Exceptions defined in this module are safe.

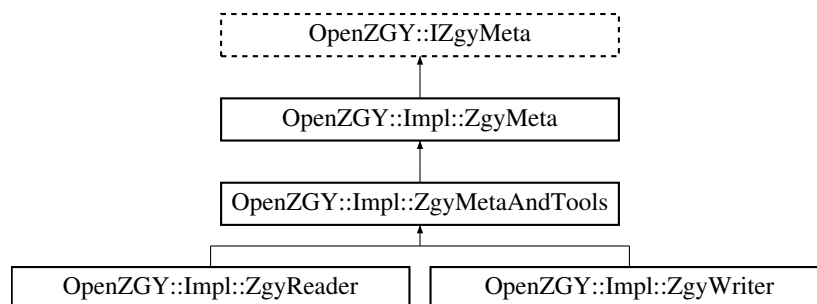
The documentation for this class was generated from the following file:

- [exception.h](#)

## 10.18 OpenZGY::Impl::ZgyMeta Class Reference

High level API for reading and writing ZGY files.

Inheritance diagram for OpenZGY::Impl::ZgyMeta:



## Public Member Functions

- virtual `std::array< int64_t, 3 > size ()` const override  
*Size in inline, crossline, vertical directions.*
- virtual `SampleDataType datatype ()` const override  
*Type of samples in each brick.*
- virtual `std::array< float32_t, 2 > datarange ()` const override  
*Used for float to int scaling.*
- virtual `std::array< float32_t, 2 > raw_datarange ()` const override  
*datarange before adjustment.*
- virtual `UnitDimension zunitdim ()` const override  
*Vertical dimension.*
- virtual `UnitDimension hunitdim ()` const override  
*Horizontal dimension.*
- virtual `std::string zunitname ()` const override  
*For annotation only. Use hunitfactor, not the name, to convert to or from SI.*
- virtual `std::string hunitname ()` const override  
*For annotation only. Use hunitfactor, not the name, to convert to or from SI.*

- virtual float64\_t [zunitfactor](#) () const override  
*Multiply by this factor to convert from storage units to SI units.*
- virtual float64\_t [hunitfactor](#) () const override  
*Multiply by this factor to convert from storage units to SI units.*
- virtual float32\_t [zstart](#) () const override  
*First time/depth.*
- virtual float32\_t [zinc](#) () const override  
*Increment in vertical direction.*
- virtual std::array< float32\_t, 2 > [annotstart](#) () const override  
*First inline, crossline.*
- virtual std::array< float32\_t, 2 > [annotinc](#) () const override  
*Increment in inline, crossline directions.*
- virtual const corners\_t & [corners](#) () const  
*Survey corner points in world coordinates.*
- virtual const corners\_t & [indexcorners](#) () const  
*Survey corner points in ordinal (i,j) coordinates.*
- virtual const corners\_t & [annotcorners](#) () const  
*Survey corner points in inline, crossline coordinates.*
- virtual std::array< int64\_t, 3 > [bricksize](#) () const override  
*Size of one brick. Almost always (64,64,64), change at your own peril.*
- virtual std::vector< std::array< int64\_t, 3 > > [brickcount](#) () const override  
*Number of bricks at each resolution (LOD) level.*
- virtual int32\_t [nlods](#) () const override  
*Number of resolution (LOD) levels.*
- virtual void [dump](#) (std::ostream &os) const override  
*Output in human readable form for debugging.*
- virtual [SampleStatistics](#) [statistics](#) () const override  
*Statistics of all sample values on the file.*
- virtual [SampleHistogram](#) [histogram](#) () const override  
*Histogram of all sample values on the file.*

## Protected Attributes

- std::shared\_ptr< const InternalZGY::ZgyInternalMeta > [\\_meta](#)  
*Handle to the internal metadata layer which this class wraps.*

## Additional Inherited Members

### 10.18.1 Detailed Description

High level API for reading and writing ZGY files.

The base class contains properties common to both reader and writer.

The constructor should logically have had a ZgyInternalMeta parameter for accessing the implementation layer. But due to the way the code is structured the `_meta` pointer needs to be set in the constructor for the leaf types. The `_meta` pointer is guaranteed to not be empty except while constructig the instance.

Both [ZgyReader](#) and [ZgyWriter](#) need to retain a pointer to the file descriptor. Primarily in order to explicitly close it. Relying on the `shared_ptr` to do this when going out of scope is dangerous because of exception handling. [ZgyWriter](#) additionally needs it when flushing metadata to disk. Since there is no file descriptor in `ZgyInternalMeta`.

Both [ZgyReader](#) and [ZgyWriter](#) need a `ZgyInternalBulk` pointer to do bulk I/O. That instance in turn keeps private references to the file descriptor and the metadata but is not supposed to expose them.

The `FileADT` and `ZgyInternalBulk` pointers can be declared here since both the reader and the writer needs them. Or removed from here and duplicated in [ZgyReader](#) and [ZgyWriter](#).

Thread safety: The following applies both to this class and derived types. Modification may lead to a data race. The intent for this class is that any calls a [ZgyReader](#) instance might make will not modify data. Except possibly changes done while the file is being opened. To help enforce this, both `const` and mutable pointers are used for `ZgyInternalMeta` and `ZgyInternalBulk` with the latter being empty when instantiated from the [ZgyReader](#) constructor instead of [ZgyWriter](#). FUTURE: The instance holds one or more pointers to the data it

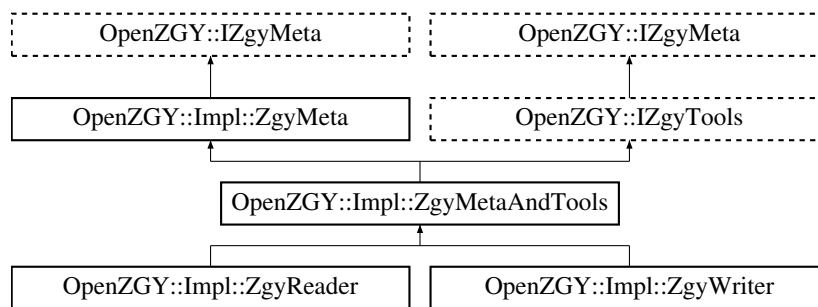
The documentation for this class was generated from the following file:

- [api.cpp](#)

## 10.19 OpenZGY::Impl::ZgyMetaAndTools Class Reference

Add coordinate conversion to the concrete [ZgyMeta](#) class.

Inheritance diagram for `OpenZGY::Impl::ZgyMetaAndTools`:



### Public Member Functions

- virtual void [transform](#) (const corners\_t &A, const corners\_t &B, std::vector< std::array< float64\_t, 2 >> &data) const override  
General coordinate conversion of an array of points. (NOT IMPLEMENTED YET)
- virtual std::array< float64\_t, 2 > [transform1](#) (const corners\_t &A, const corners\_t &B, const std::array< float64\_t, 2 > &point) const override  
General coordinate conversion of a single coordinate pair.
- virtual std::array< float64\_t, 2 > [annotToIndex](#) (const std::array< float64\_t, 2 > &point) const override  
Convert a single coordinate pair.
- virtual std::array< float64\_t, 2 > [annotToWorld](#) (const std::array< float64\_t, 2 > &point) const override  
Convert a single coordinate pair.
- virtual std::array< float64\_t, 2 > [indexToAnnot](#) (const std::array< float64\_t, 2 > &point) const override  
Convert a single coordinate pair.
- virtual std::array< float64\_t, 2 > [indexToWorld](#) (const std::array< float64\_t, 2 > &point) const override  
Convert a single coordinate pair.
- virtual std::array< float64\_t, 2 > [worldToAnnot](#) (const std::array< float64\_t, 2 > &point) const override  
Convert a single coordinate pair.
- virtual std::array< float64\_t, 2 > [worldToIndex](#) (const std::array< float64\_t, 2 > &point) const override  
Convert a single coordinate pair.

## Additional Inherited Members

### 10.19.1 Detailed Description

Add coordinate conversion to the concrete [ZgyMeta](#) class.

Thread safety: See the base class. This specialization does not add any additional concerns because all the new members are const. So they are safe to call concurrently with other read operations.

### 10.19.2 Member Function Documentation

#### 10.19.2.1 transform()

```
virtual void OpenZGY::Impl::ZgyMetaAndTools::transform (
    const corners_t & from,
    const corners_t & to,
    std::vector< std::array< float64_t, 2 >> & ) const [override], [virtual]
```

General coordinate conversion of an array of points. (NOT IMPLEMENTED YET)

##### Parameters

<i>from</i>	control points in the current coordinate system.
<i>to</i>	control points in the desired coordinate system.

Convert coordinates in place, with the conversion defined by giving the values of 3 arbitrary control points in both the "from" and "to" coordinate system. A common choice of arbitrary points is to use three of the lattice corners. As a convenience the "from" and "to" parameters are declared as `corners_t` so the caller can pass [corners\(\)](#), [annotcorners\(\)](#), or [indexcorners\(\)](#) directly.

Implements [OpenZGY::IZgyTools](#).

#### 10.19.2.2 transform1()

```
virtual std::array<float64_t,2> OpenZGY::Impl::ZgyMetaAndTools::transform1 (
    const corners_t & from,
    const corners_t & to,
    const std::array< float64_t, 2 > & ) const [override], [virtual]
```

General coordinate conversion of a single coordinate pair.

##### Parameters

<i>from</i>	control points in the current coordinate system.
<i>to</i>	control points in the desired coordinate system.

Convert coordinates in place, with the conversion defined by giving the values of 3 arbitrary control points in both the "from" and "to" coordinate system. A common choice of arbitrary points is to use three of the lattice corners. As a convenience the "from" and "to" parameters are declared as `corners_t` so the caller can pass `corners()`, `annotcorners()`, or `indexcorners()` directly.

Implements `OpenZGY::IZgyTools`.

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

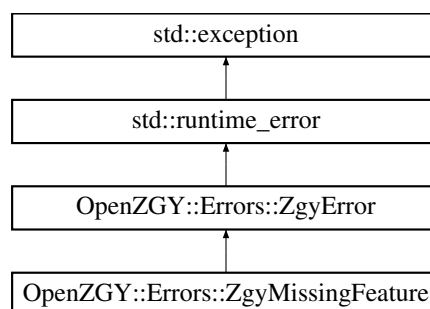
- `api.cpp`

## 10.20 OpenZGY::Errors::ZgyMissingFeature Class Reference

Missing feature.

```
#include <exception.h>
```

Inheritance diagram for `OpenZGY::Errors::ZgyMissingFeature`:



### Public Member Functions

- `ZgyMissingFeature` (const std::string &arg)  
*Missing feature.*

### Additional Inherited Members

#### 10.20.1 Detailed Description

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.

Thread safety: Exceptions defined in this module are safe.

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

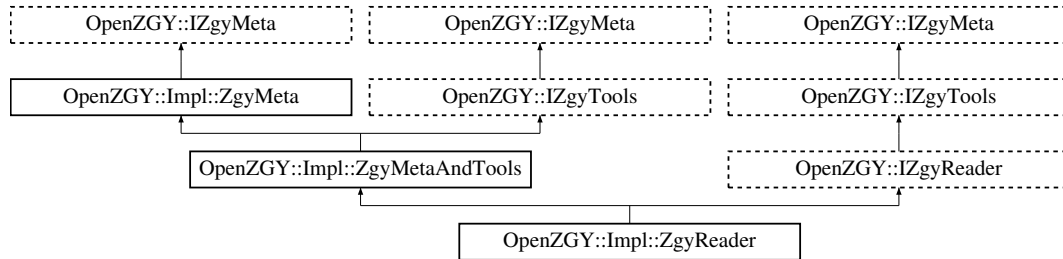
- `exception.h`



## 10.21 OpenZGY::Impl::ZgyReader Class Reference

Concrete implementation of [IZgyReader](#).

Inheritance diagram for OpenZGY::Impl::ZgyReader:



### Public Member Functions

- [ZgyReader](#) (const std::string &filename, const [IOContext](#) \*iocontext)  
*Open a ZGY file for reading.*
- virtual void [read](#) (const size3i\_t &start, const size3i\_t &[size](#), float \*data, int lod) const override  
*Read an arbitrary region.*
- virtual void [read](#) (const size3i\_t &start, const size3i\_t &[size](#), std::int16\_t \*data, int lod) const override  
*Read an arbitrary region with no conversion.*
- virtual void [read](#) (const size3i\_t &start, const size3i\_t &[size](#), std::int8\_t \*data, int lod) const override  
*Read an arbitrary region with no conversion.*
- virtual std::pair< bool, double > [readconst](#) (const size3i\_t &start, const size3i\_t &[size](#), int lod, bool as\_float) const override  
*Get hint about all constant region.*
- void [close](#) ()  
*Close the file and release resources.*

### Additional Inherited Members

#### 10.21.1 Detailed Description

Concrete implementation of [IZgyReader](#).

Thread safety: See the base class.

#### 10.21.2 Constructor & Destructor Documentation

##### 10.21.2.1 ZgyReader()

```

OpenZGY::Impl::ZgyReader::ZgyReader (
    const std::string & filename,
    const IOContext * iocontext )

```

Open a ZGY file for reading.

## 10.21.3 Member Function Documentation

### 10.21.3.1 close()

```
void OpenZGY::Impl::ZgyReader::close ( ) [virtual]
```

Close the file and release resources.

Unlike [ZgyWriter::close\(\)](#), forgetting to close a file that was only open for read is not a major faux pas. It can still lead to problems though.

- The destructor of `_fd` will catch and ignore any exceptions because if a destructor throws then this will normally cause the application to crash.
- If a callback is used to refresh the token this will not happen in our destructor (it won't call `xx_close`) because it is too risky to invoke the callback this late. It might not be valid any longer. This means that if the token has expired since the last read then the close will fail. Exactly why SDAPI requires a token just to close a file is a different question. Possibly this is for removing any read locks.

Implements [OpenZGY::IZgyReader](#).

### 10.21.3.2 read() [1/3]

```
virtual void OpenZGY::Impl::ZgyReader::read (
    const size3i_t & start,
    const size3i_t & size,
    float * data,
    int lod ) const [override], [virtual]
```

Read an arbitrary region.

The data is read into a buffer provided by the caller. The method will apply conversion storage -> float if needed.

Data is ordered inline(slowest), crossline, vertical(fastest).

The start position refers to the specified lod level. At lod 0 start + data.size can be up to the survey size. At lod 1 the maximum is just half that, rounded up.

It is valid to pass a size that includes the padding area between the survey and the end of the current brick. But not more. In other words, the limit for lod 0 is actually `reader()->size()` rounded up to a multiple of `reader()->bricksize()`.

Implements [OpenZGY::IZgyReader](#).

**10.21.3.3 read() [2/3]**

```
virtual void OpenZGY::Impl::ZgyReader::read (
    const size3i_t & start,
    const size3i_t & size,
    std::int16_t * data,
    int lod ) const [override], [virtual]
```

Read an arbitrary region with no conversion.

As the read overload with a float buffer but only works for files with SampleDataType::int16 and does not scale the samples.

Implements [OpenZGY::IZgyReader](#).

**10.21.3.4 read() [3/3]**

```
virtual void OpenZGY::Impl::ZgyReader::read (
    const size3i_t & start,
    const size3i_t & size,
    std::int8_t * data,
    int lod ) const [override], [virtual]
```

Read an arbitrary region with no conversion.

As the read overload with a float buffer but only works for files with SampleDataType::int8 and does not scale the samples.

Implements [OpenZGY::IZgyReader](#).

**10.21.3.5 readconst()**

```
virtual std::pair<bool,double> OpenZGY::Impl::ZgyReader::readconst (
    const size3i_t & start,
    const size3i_t & size,
    int lod,
    bool as_float ) const [override], [virtual]
```

Get hint about all constant region.

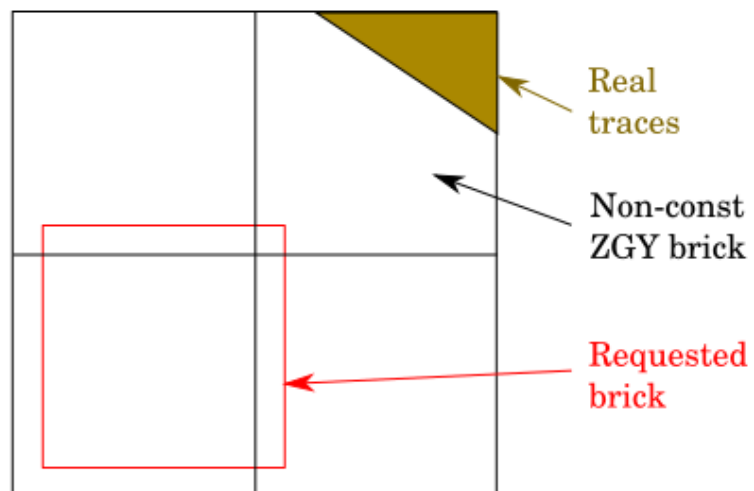
Check to see if the specified region is known to have all samples set to the same value. Returns a pair of (is\_const, const\_value). If it returns is\_const=true you know what the region contains; otherwise you need to use the regular [read\(\)](#) method.

For int8 and int16 files the caller may specify whether to scale the values or not. Even if unscaled the function returns the value as a double.

The function only makes inexpensive checks so it might return is\_const=false 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. The following figure might clarify how it works.

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

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



If the region requested in `readconst()` is exactly one ZGY brick then this just checks whether that brick is known to be constant. If called with a larger, arbitrary region it fails unless all bricks in that region are known to be constant and all have the same value. It will not tell you which brick(s) were not const. If you need to know then you must loop over each ZGY brick and make multiple calls to `readconst()`.

Implements [OpenZGY::IZgyReader](#).

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

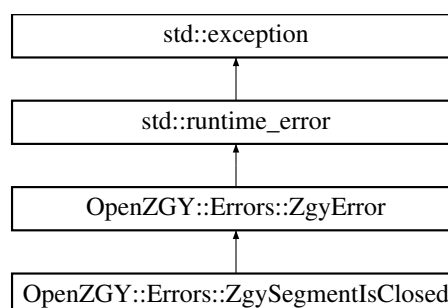
- [api.cpp](#)

## 10.22 OpenZGY::Errors::ZgySegmentIsClosed Class Reference

Exception used internally to request a retry.

```
#include <exception.h>
```

Inheritance diagram for OpenZGY::Errors::ZgySegmentIsClosed:



### Public Member Functions

- [ZgySegmentIsClosed](#) (const std::string &arg)  
*Exception used internally to request a retry.*

## Additional Inherited Members

### 10.22.1 Detailed Description

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.

Thread safety: Exceptions defined in this module are safe.

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

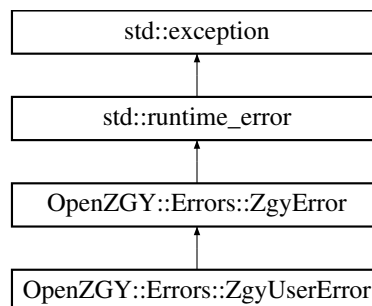
- [exception.h](#)

## 10.23 OpenZGY::Errors::ZgyUserError Class Reference

Exception that might be caused by the calling application.

```
#include <exception.h>
```

Inheritance diagram for OpenZGY::Errors::ZgyUserError:



## Public Member Functions

- [ZgyUserError](#) (const std::string &arg)  
*Exception that might be caused by the calling application.*

## Additional Inherited Members

### 10.23.1 Detailed Description

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.

Thread safety: Exceptions defined in this module are safe.

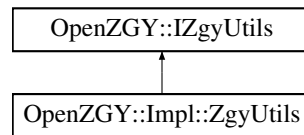
The documentation for this class was generated from the following file:

- [exception.h](#)

## 10.24 OpenZGY::Impl::ZgyUtils Class Reference

Concrete implementation of [IZgyUtils](#).

Inheritance diagram for OpenZGY::Impl::ZgyUtils:



### Public Member Functions

- [ZgyUtils](#) (const std::string &prefix, const [IOContext](#) \*iocontext)  
*Create a new concrete instance of [IZgyUtils](#).*
- void [deletefile](#) (const std::string &filename, bool missing\_ok)  
*Delete a file. Works both for local and cloud files.*
- std::string [alturl](#) (const std::string &filename)  
*Return a url that might be opened faster.*

### Additional Inherited Members

#### 10.24.1 Detailed Description

Concrete implementation of [IZgyUtils](#).

Thread safety: Depends on the function being executed. Currently implemented methods just forward the requests to SDAPI or some other cloud back end. So thread safety depends on the cloud provider.

#### 10.24.2 Constructor & Destructor Documentation

##### 10.24.2.1 ZgyUtils()

```

OpenZGY::Impl::ZgyUtils::ZgyUtils (
    const std::string & prefix,
    const IOContext * iocontext )
  
```

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

##### Parameters

<i>prefix</i>	File name or file name prefix.
<i>iocontext</i>	Credentials and other configuration.

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 both "sd://some/bogus/file.zgy" and just "sd://" will produce an instance that works for the seismic store.

For performance reasons you should consider caching one [IZgyUtils](#) 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.

## 10.24.3 Member Function Documentation

### 10.24.3.1 alturl()

```
std::string OpenZGY::Impl::ZgyUtils::alturl (
    const std::string & filename ) [virtual]
```

Return a url that might be opened faster.

Using this method might help performance in a distributed system where a large number of processing nodes need to open the same file for read. One node would call this method which costs about the same as opening the file locally. All other nodes can then use the alternate url which opens a lot faster due to the meta data having already been retrieved and serialized in the alturl.

The alternate url is valid for a limited time only. Typically it expires when the current access token expires but it might expire earlier.

Note that the alternate url might contain secrets, so it needs to be protected just as you would protect an access token.

The method is not useful for local files where the file name will just be returned unmodified. The method is unlikely to be useful in a single-node application. In that case the application can simply keep the file open instead of keeping the alturl reference.

Implements [OpenZGY::IZgyUtils](#).

### 10.24.3.2 deletefile()

```
void OpenZGY::Impl::ZgyUtils::deletefile (
    const std::string & filename,
    bool missing_ok ) [virtual]
```

Delete a file. Works both for local and cloud files.

Note that the instance must be of the correct (local or cloud) type.

Implements [OpenZGY::IZgyUtils](#).

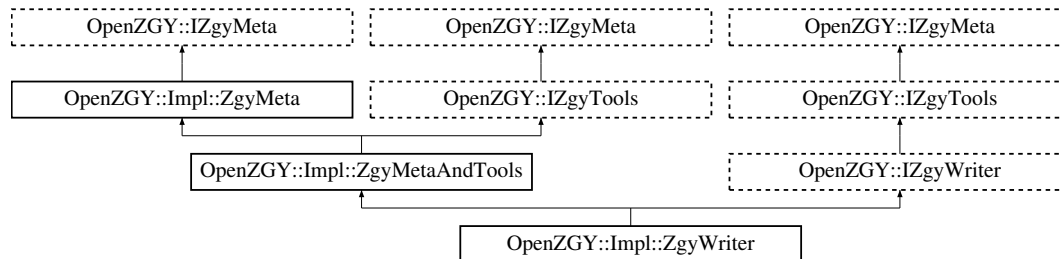
The documentation for this class was generated from the following file:

- [api.cpp](#)

## 10.25 OpenZGY::Impl::ZgyWriter Class Reference

Concrete implementation of [IZgyWriter](#).

Inheritance diagram for OpenZGY::Impl::ZgyWriter:



### Public Member Functions

- [ZgyWriter](#) (const [ZgyWriterArgs](#) &args)  
*Create a ZGY file and open it for writing.*
- virtual [~ZgyWriter](#) ()  
*Automatically close the file when it goes out of scope.*
- void [write](#) (const size3i\_t &start, const size3i\_t &size, const float \*data) override  
*Write an arbitrary region.*
- void [write](#) (const size3i\_t &start, const size3i\_t &size, const std::int16\_t \*data) override  
*Write an arbitrary region with no conversion.*
- void [write](#) (const size3i\_t &start, const size3i\_t &size, const std::int8\_t \*data) override  
*Write an arbitrary region with no conversion.*
- void [writeconst](#) (const size3i\_t &start, const size3i\_t &size, const float \*data) override  
*Write all-constant data.*
- void [writeconst](#) (const size3i\_t &start, const size3i\_t &size, const std::int16\_t \*data) override  
*Write an arbitrary region with no conversion.*
- void [writeconst](#) (const size3i\_t &start, const size3i\_t &size, const std::int8\_t \*data) override  
*Write an arbitrary region with no conversion.*
- void [finalize](#) (const std::vector< DecimationType > &decimation, const std::function< bool(std::int64\_t, std::int64\_t)> &progress, bool force=false)  
*Generate low resolution data, statistics, and histogram.*
- void [close\\_incomplete](#) ()  
*Flush the file to disk and close it.*
- void [close](#) ()  
*Flush the file to disk and close it.*
- bool [errorflag](#) () const override
- void [set\\_errorflag](#) (bool value) override



## Additional Inherited Members

### 10.25.1 Detailed Description

Concrete implementation of [IZgyWriter](#).

Thread safety: This class is single threaded. [IZgyWriter::open\(\)](#) may choose to return a wrapper that serializes all access, just in case the application didn't read the memo about no concurrent access. See class [ZgySafeWriter](#) for details.

Const-correctness: Most methods are non-const because writes will update metadata. If not in [ZgyWriter](#) itself then in the classes it refers to. The class refers to a mutable [ZgyInternalBulk](#) instance. And a mutable [ZgyInternalMeta](#) instance that is actually the same as the one in the base class except this one is not declared const. Now for the minor code smell: Any const method (currently just `errorflag`) will still be able to access those `_rw` pointers so the "const" declaration hardly protects against anything. There are ways of handling this better. But with just a single method affected I'll let it slide.

### 10.25.2 Constructor & Destructor Documentation

#### 10.25.2.1 ZgyWriter()

```
OpenZGY::Impl::ZgyWriter::ZgyWriter (
    const ZgyWriterArgs & args )
```

Create a ZGY file and open it for writing.

#### 10.25.2.2 ~ZgyWriter()

```
virtual OpenZGY::Impl::ZgyWriter::~ZgyWriter ( ) [virtual]
```

Automatically close the file when it goes out of scope.

Application code is encouraged to close the file explicitly. The destructor is just there as a fallback. [Errors](#) caught in the fallback will be logged to `stderr` and otherwise ignored.

### 10.25.3 Member Function Documentation

### 10.25.3.1 close()

```
void OpenZGY::Impl::ZgyWriter::close ( ) [virtual]
```

Flush the file to disk and close it.

If the file has been written to, the application is encouraged to call [finalize\(\)](#) before [close\(\)](#). This gives more control over the process and allows using a progress callback to track generation of low resolution data.

The function won't bother with statistics, histogram, lowres if there has been an unrecoverable error. The headers might still be written out in case somebody wants to try some forensics.

The [ZgyWriter](#) destructor will call [close\(\)](#) if not done already, but that will catch and swallow any exception. Relying on the destructor to close the file is strongly discouraged.

Implements [OpenZGY::IZgyWriter](#).

### 10.25.3.2 close\_incomplete()

```
void OpenZGY::Impl::ZgyWriter::close_incomplete ( ) [virtual]
```

Flush the file to disk and close it.

This version of [close\(\)](#) will not calculate statistics and low resolution bricks. Currently this makes the file useless in most cases. The function may be useful for performance measurements.

In the future it might be possible to re-open the file at some later date and continue writing data to it. Calling the regular [close\(\)](#) only when all data has been output.

Implements [OpenZGY::IZgyWriter](#).

### 10.25.3.3 errorflag()

```
bool OpenZGY::Impl::ZgyWriter::errorflag ( ) const [override]
```

Return true if this open file has encountered an unrecoverable error. The error should previously have caused an exception to be thrown. If this flag is set, no further writes will be allowed.

Application code might check this flag if they are considering trying to recover from an error. Internally the flag is also checked and if set it will (mostly) prevent other writes from being done.

Implementation note: Currently the [ZgyInternalMeta](#) and [ZgyInternalBulk](#) instances each contains an `_is_bad` member. The reader or writer is considered bad if either of those are set. This scheme improves isolation somewhat, but TODO-Low it might backfire. If writing metadata to file failed, the bulk accessor should probably behave as if it also has seen an error.

Currently only the [ZgyWriter](#) uses this mechanism. It might not make that much sense in [ZgyReader](#), because as long as opening the file succeeded no operation should manage to corrupt it.

Implementation note: Unlike most other [ZgyWriter](#) members, this one is declared `const`. But it can still access the mutable `_accessor_rw` and `_meta_rw` data members. You'll just have to trust me when I tell you they aren't being modified here. Or refactor somehow to allow the compiler (or at least a runtime check) catch it.

### 10.25.3.4 finalize()

```
void OpenZGY::Impl::ZgyWriter::finalize (
    const std::vector< DecimationType > & decimation,
    const std::function< bool(std::int64_t, std::int64_t)> & progress,
    bool force = false ) [virtual]
```

Generate low resolution data, statistics, and histogram.

This method 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`.

The C++ code is very different from Python because it needs an entirely different approach to be performant.

#### Parameters

<i>decimation</i>	Optionally override the decimation algorithms by passing an array of DecimationType with one entry for each level of detail. If the array is too short then the last entry is used for subsequent levels.
<i>progress</i>	Function(done, total) called to report progress. If it returns False the computation is aborted. The callback implementation should not invoke any <a href="#">OpenZGY</a> methods except for any calls expressly documented as safe for this purpose and deadlock-free. Will be called at least one, even if there is no work.
<i>force</i>	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. passing <code>force=true</code> is currently required if cerating an empty file, no samples written at all, if the statistics and histogram data should still be stored.

Implements [OpenZGY::IZgyWriter](#).

### 10.25.3.5 set\_errorflag()

```
void OpenZGY::Impl::ZgyWriter::set_errorflag (
    bool value ) [override]
```

Force the error flag for this open file to true or false. This should normally be done only for testing.

### 10.25.3.6 write() [1/3]

```
void OpenZGY::Impl::ZgyWriter::write (
    const size3i_t & start,
    const size3i_t & size,
    const float * data ) [override], [virtual]
```

Write an arbitrary region.

This will apply conversion float -> storage if needed.

Data is ordered inline(slowest), crossline, vertical(fastest).

A read/modify/write will be done if the region's start and size doesn't align with bricksize. When writing to the cloud this read/modify/write may incur performance and size penalties. So do write brick aligned data if possible. The same applies to writing compressed data where r/m/w can cause a severe loss of quality.

The start position refers to the specified lod level. At lod 0 start + data.size can be up to the survey size. At lod 1 the maximum is just half that, rounded up.

Implements [OpenZGY::IZgyWriter](#).

#### 10.25.3.7 write() [2/3]

```
void OpenZGY::Impl::ZgyWriter::write (
    const size3i_t & start,
    const size3i_t & size,
    const std::int16_t * data ) [override], [virtual]
```

Write an arbitrary region with no conversion.

As the write overload with a float buffer but only works for files with SampleDataType::int16 and does not scale the samples.

Implements [OpenZGY::IZgyWriter](#).

#### 10.25.3.8 write() [3/3]

```
void OpenZGY::Impl::ZgyWriter::write (
    const size3i_t & start,
    const size3i_t & size,
    const std::int8_t * data ) [override], [virtual]
```

Write an arbitrary region with no conversion.

As the write overload with a float buffer but only works for files with SampleDataType::int8 and does not scale the samples.

Implements [OpenZGY::IZgyWriter](#).

#### 10.25.3.9 writeconst() [1/3]

```
void OpenZGY::Impl::ZgyWriter::writeconst (
    const size3i_t & start,
    const size3i_t & size,
    const float * data ) [override], [virtual]
```

Write all-constant data.

Works as the corresponding write but the entire region is set to the same value. So the provided data buffer needs just one value, or alternatively can be passed as &scalar\_value.

Calling this method is faster than filling a buffer with constant values and calling write. But it produces the exact same result. This is because write will automatically detect whether the input buffer is all constant.

Implements [OpenZGY::IZgyWriter](#).

**10.25.3.10 writeconst() [2/3]**

```
void OpenZGY::Impl::ZgyWriter::writeconst (
    const size3i_t & start,
    const size3i_t & size,
    const std::int16_t * data ) [override], [virtual]
```

Write an arbitrary region with no conversion.

As the writeconst overload with a float buffer but only works for files with SampleDataType::int16 and does not scale the samples.

Implements [OpenZGY::IZgyWriter](#).

**10.25.3.11 writeconst() [3/3]**

```
void OpenZGY::Impl::ZgyWriter::writeconst (
    const size3i_t & start,
    const size3i_t & size,
    const std::int8_t * data ) [override], [virtual]
```

Write an arbitrary region with no conversion.

As the write overload with a float buffer but only works for files with SampleDataType::int8 and does not scale the samples.

Implements [OpenZGY::IZgyWriter](#).

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

- [api.cpp](#)

**10.26 OpenZGY::ZgyWriterArgs Class Reference**

Argument package for creating a ZGY file.

```
#include <api.h>
```

**Public Types**

- typedef double **float64\_t**
- typedef std::array< std::array< float64\_t, 2 >, 4 > **corners\_t**
- typedef std::pair< std::shared\_ptr< const void >, std::int64\_t > **rawdata\_t**
- typedef std::function< rawdata\_t(const rawdata\_t &, const std::array< int64\_t, 3 > &)> **compressor\_t**

## Public Member Functions

- void [dump](#) (std::ostream &out) const  
*Output in human readable form for debugging.*
- [ZgyWriterArgs](#) & [filename](#) (const std::string &value)  
*Set file to open.*
- [ZgyWriterArgs](#) & [iocontext](#) (const [IOContext](#) \*value)  
*Set credentials and other configuration.*
- [ZgyWriterArgs](#) & [compressor](#) (const compressor\_t &value)  
*Set functor for compressing full resolution data.*
- [ZgyWriterArgs](#) & [compressor](#) (const std::string &name, const std::vector< std::string > &args)  
*Set functor for compressing full resolution data.*
- [ZgyWriterArgs](#) & [zfp\\_compressor](#) (float snr)  
*Set functor for compressing full resolution data.*
- [ZgyWriterArgs](#) & [lodcompressor](#) (const compressor\_t &value)  
*Set functor for compressing low resolution data.*
- [ZgyWriterArgs](#) & [lodcompressor](#) (const std::string &name, const std::vector< std::string > &args)  
*Set functor for compressing low resolution data.*
- [ZgyWriterArgs](#) & [zfp\\_lodcompressor](#) (float snr)  
*Set functor for compressing low resolution data.*
- [ZgyWriterArgs](#) & [size](#) (std::int64\_t ni, std::int64\_t nj, std::int64\_t nk)  
*Set size of the survey.*
- [ZgyWriterArgs](#) & [bricksize](#) (std::int64\_t ni, std::int64\_t nj, std::int64\_t nk)  
*Set size of one brick.*
- [ZgyWriterArgs](#) & [datatype](#) (SampleDataType value)  
*Set type of samples in each brick.*
- [ZgyWriterArgs](#) & [datarange](#) (float lo, float hi)  
*Set scaling factors.*
- [ZgyWriterArgs](#) & [zunit](#) (UnitDimension dimension, const std::string &name, double factor)  
*Set vertical unit.*
- [ZgyWriterArgs](#) & [hunite](#) (UnitDimension dimension, const std::string &name, double factor)  
*Set horizontal unit.*
- [ZgyWriterArgs](#) & [ilstart](#) (float value)  
*Set first (ordinal 0) inline number.*
- [ZgyWriterArgs](#) & [ilinc](#) (float value)  
*Set inline number increment between two adjacent ordinal values.*
- [ZgyWriterArgs](#) & [xlstart](#) (float value)  
*Set first (ordinal 0) crossline number.*
- [ZgyWriterArgs](#) & [xlinc](#) (float value)  
*Set crossline number increment between two adjacent ordinal values.*
- [ZgyWriterArgs](#) & [zstart](#) (float value)  
*Set first time/depth.*
- [ZgyWriterArgs](#) & [zinc](#) (float value)  
*Set increment (distance between samples) in vertical direction.*
- [ZgyWriterArgs](#) & [corners](#) (const corners\_t &value)  
*Set survey corner points in world coordinates.*
- [ZgyWriterArgs](#) & [metafrom](#) (const std::shared\_ptr< [OpenZGY::IZgyReader](#) > &)  
*Copy metadata from existing file.*
- [ZgyWriterArgs](#) & [merge](#) (const [ZgyWriterArgs](#) &)  
*Copy metadata from another [ZgyWriterArgs](#).*

## 10.26.1 Detailed Description

Argument package for creating a ZGY file.

This is a short lived helper class for passing arguments to the functions that create a ZGY file. Needed because there are a lot of arguments and C++ doesn't allow keyword arguments. Do NOT use this class for holding on to the information.

Information not set explicitly will be defaulted. The following two statements give the same result:

```
ZgyWriterArgs default_args = ZgyWriterArgs();
ZgyWriterArgs default_args = ZgyWriterArgs()
    .filename("")
    .iocontext(nullptr)
    .compressor(nullptr)
    .lodcompressor(nullptr)
    .size(0, 0, 0)
    .bricksize(64, 64, 64)
    .datatype(SampleDataType::float32)
    .datarange(0, -1)
    .zunit(UnitDimension::unknown, "", 1.0)
    .hunit(UnitDimension::unknown, "", 1.0)
    .ilstart(0)
    .ilinc(0)
    .xlstart(0)
    .xlinc(0)
    .zstart(0)
    .zinc(0)
    .corners(ZgyWriterArgs::corners_t{0,0,0,0,0,0,0,0});
```

Thread safety: Modification may lead to a data race. This should not be an issue, because instances are only meant to be modified when created or copied or assigned prior to being made available to others.

## 10.26.2 Member Function Documentation

### 10.26.2.1 bricksize()

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::bricksize (
    std::int64_t ni,
    std::int64_t nj,
    std::int64_t nk )
```

Set size of one brick.

Almost always (64,64,64). Change at your own peril.

### 10.26.2.2 compressor()

```
ZgyWriterArgs & OpenZGY::ZgyWriterArgs::compressor (
    const std::string & name,
    const std::vector< std::string > & args )
```

Set functor for compressing full resolution data.

This overload uses a factory to look up the compressor.

### 10.26.2.3 corners()

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::corners (
    const corners_t & value )
```

Set survey corner points in world coordinates.

The corners are ordered origin, last inline (i.e. i=last, j=0), last crossline, diagonal.

### 10.26.2.4 datarange()

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::datarange (
    float lo,
    float hi )
```

Set scaling factors.

For integral storage this specifies the two floating point numbers that correspond to the lowest and highest representable integer value. So for int8 files, -128 will be converted to "lo" and +127 will be converted to "hi".

ZGY files require that min<max. This is not enforced here, but is checked when the file is actually created.

### 10.26.2.5 filename()

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::filename (
    const std::string & value )
```

Set file to open.

If starting with sd:// this opens a file on seismic store.

### 10.26.2.6 hunit()

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::hunit (
    UnitDimension dimension,
    const std::string & name,
    double factor )
```

Set horizontal unit.

#### Parameters

<i>dimension</i>	cartesian length or (unsupported) polat coordinates,
<i>name</i>	for annotation only.
<i>factor</i>	multiply by this factor to convert from storage units to SI units.



### 10.26.2.7 ilinc()

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::ilinc (
    float value )
```

Set inline number increment between two adjacent ordinal values.

For maximum portability the inline and crossline start and increment should be integral numbers. Some applications might choose to convert them to int.

### 10.26.2.8 ilstart()

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::ilstart (
    float value )
```

Set first (ordinal 0) inline number.

For maximum portability the inline and crossline start and increment should be integral numbers. Some applications might choose to convert them to int.

### 10.26.2.9 iocontext()

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::iocontext (
    const IOContext * value )
```

Set credentials and other configuration.

This depends on the back-end. For local files you normally don't need to pass anything here.

### 10.26.2.10 lodcompressor()

```
ZgyWriterArgs & OpenZGY::ZgyWriterArgs::lodcompressor (
    const std::string & name,
    const std::vector< std::string > & args )
```

Set functor for compressing low resolution data.

This overload uses a factory to look up the compressor.

### 10.26.2.11 merge()

```
ZgyWriterArgs & OpenZGY::ZgyWriterArgs::merge (
    const ZgyWriterArgs & other )
```

Copy metadata from another [ZgyWriterArgs](#).

Copy only those settings that have been explicitly changed in the supplied "other" [ZgyWriterArgs](#) into \*this. If other.metafrom() has been called it is unspecified what gets copied.

**10.26.2.12 metafrom()**

```
ZgyWriterArgs & OpenZGY::ZgyWriterArgs::metafrom (
    const std::shared_ptr< OpenZGY::IZgyReader > & reader )
```

Copy metadata from existing file.

Set most of the metadata from an open file descriptor. Typically used when the file to be created is to be computed from an existing file. Typically this will be called first so the settings don't inadvertently shadow something set explicitly.

**10.26.2.13 size()**

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::size (
    std::int64_t ni,
    std::int64_t nj,
    std::int64_t nk )
```

Set size of the survey.

**Parameters**

<i>ni</i>	number of inlines (slowest varying index).
<i>nj</i>	number of crosslines.
<i>nk</i>	number of samples per trace (fastest).

**10.26.2.14 xlinc()**

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::xlinc (
    float value )
```

Set crossline number increment between two adjacent ordinal values.

For maximum portability the inline and crossline start and increment should be integral numbers. Some applications might choose to convert them to int.

**10.26.2.15 xlstart()**

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::xlstart (
    float value )
```

Set first (ordinal 0) crossline number.

For maximum portability the inline and crossline start and increment should be integral numbers. Some applications might choose to convert them to int.

**10.26.2.16 zfp\_compressor()**

```
ZgyWriterArgs & OpenZGY::ZgyWriterArgs::zfp_compressor (
    float snr )
```

Set functor for compressing full resolution data.

This overload uses a factory to look up the ZGY compressor. It is just a convenience that is shorter to type.

**10.26.2.17 zfp\_lodcompressor()**

```
ZgyWriterArgs & OpenZGY::ZgyWriterArgs::zfp_lodcompressor (
    float snr )
```

Set functor for compressing low resolution data.

This overload uses a factory to look up the ZGY compressor. It is just a convenience that is shorter to type.

**10.26.2.18 zinc()**

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::zinc (
    float value )
```

Set increment (distance between samples) in vertical direction.

Vertical annotation is generally safe to have non-integral.

**10.26.2.19 zstart()**

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::zstart (
    float value )
```

Set first time/depth.

Vertical annotation is generally safe to have non-integral.

**10.26.2.20 zunit()**

```
ZgyWriterArgs& OpenZGY::ZgyWriterArgs::zunit (
    UnitDimension dimension,
    const std::string & name,
    double factor )
```

Set vertical unit.

**Parameters**

<i>dimension</i>	time or depth (a.k.a. length).
<i>name</i>	for annotation only.
<i>factor</i>	multiply by this factor to convert from storage units to SI units.

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

- [api.h](#)
- [api.cpp](#)

# Chapter 11

## File Documentation

### 11.1 `api.cpp` File Reference

Implements the pure interfaces of the API.

```
#include "api.h"
#include "exception.h"
#include "safewriter.h"
#include "impl/enum.h"
#include "impl/file.h"
#include "impl/file_sd.h"
#include "impl/meta.h"
#include "impl/bulk.h"
#include "impl/databuffer.h"
#include "impl/transform.h"
#include "impl/lodalgo.h"
#include "impl/statisticdata.h"
#include "impl/histogramdata.h"
#include "impl/genlod.h"
#include "impl/compression.h"
```

#### Classes

- class `OpenZGY::Impl::ZgyMeta`  
*High level API for reading and writing ZGY files.*
- class `OpenZGY::Impl::ZgyMetaAndTools`  
*Add coordinate conversion to the concrete `ZgyMeta` class.*
- class `OpenZGY::Impl::ZgyReader`  
*Concrete implementation of `IZgyReader`.*
- class `OpenZGY::Impl::ZgyWriter`  
*Concrete implementation of `IZgyWriter`.*
- class `OpenZGY::Impl::ZgyUtils`  
*Concrete implementation of `IZgyUtils`.*

## Namespaces

- [OpenZGY](#)  
*The entire public API is in this namespace.*
- [OpenZGY::Impl](#)  
*Implementation of the abstract interfaces in [OpenZGY](#).*
- [OpenZGY::Formatters](#)  
*operator<< for readable output of enums etc.*

## Functions

- `std::string OpenZGY::Formatters::enumToString (SampleDataType value)`  
*Return the string representation of the input enum type.*
- `std::string OpenZGY::Formatters::enumToString (UnitDimension value)`  
*Return the string representation of the input enum type.*
- `std::string OpenZGY::Formatters::enumToString (DecimationType value)`  
*Return the string representation of the input enum type.*
- `OPENZGY_API std::ostream & OpenZGY::Formatters::operator<< (std::ostream &os, SampleDataType value)`  
*Output the string representation of the input enum type.*
- `OPENZGY_API std::ostream & OpenZGY::Formatters::operator<< (std::ostream &os, UnitDimension value)`  
*Output the string representation of the input enum type.*
- `OPENZGY_API std::ostream & OpenZGY::Formatters::operator<< (std::ostream &os, DecimationType value)`  
*Output the string representation of the input enum type.*

### 11.1.1 Detailed Description

Implements the pure interfaces of the API.

## 11.2 api.h File Reference

IZgyReader, IZgyWriter, and other user visible classes.

```
#include <cstdint>
#include <vector>
#include <array>
#include <ostream>
#include <iostream>
#include <functional>
#include <memory>
#include <string>
#include <mutex>
#include "declspec.h"
```

## Classes

- class [OpenZGY::SampleStatistics](#)  
*Statistics of all sample values on the file.*
- class [OpenZGY::SampleHistogram](#)  
*Histogram of all sample values on the file.*
- class [OpenZGY::ZgyWriterArgs](#)  
*Argument package for creating a ZGY file.*
- class [OpenZGY::IZgyMeta](#)  
*Base class of [IZgyReader](#) and [IZgyWriter](#).*
- class [OpenZGY::IZgyTools](#)  
*Base class of [IZgyReader](#) and [IZgyWriter](#).*
- class [OpenZGY::IZgyReader](#)  
*Main API for reading ZGY files.*
- class [OpenZGY::IZgyWriter](#)  
*Main API for creating ZGY files.*
- class [OpenZGY::IZgyUtils](#)  
*Operations other than read and write.*
- class [OpenZGY::ProgressWithDots](#)  
*Simple progress bar.*

## Namespaces

- [OpenZGY](#)  
*The entire public API is in this namespace.*
- [OpenZGY::Errors](#)  
*Exceptions that can be thrown by [OpenZGY](#).*
- [OpenZGY::Impl](#)  
*Implementation of the abstract interfaces in [OpenZGY](#).*
- [OpenZGY::Formatters](#)  
*`operator<<` for readable output of enums etc.*

## Functions

- OPENZGY\_API std::ostream & [OpenZGY::Formatters::operator<<](#) (std::ostream &os, SampleDataType value)  
*Output the string representation of the input enum type.*
- OPENZGY\_API std::ostream & [OpenZGY::Formatters::operator<<](#) (std::ostream &os, UnitDimension value)  
*Output the string representation of the input enum type.*
- OPENZGY\_API std::ostream & [OpenZGY::Formatters::operator<<](#) (std::ostream &os, DecimationType value)  
*Output the string representation of the input enum type.*

## Variables

- **unknown** = 1000
- **int8** = 1001
- **int16** = 1002
- **float32** = 1003
- **time** = 2001
- **length** = 2002
- **arcangle** = 2003
- **LowPass** = 100  
*Lowpass Z / decimate XY.*
- **WeightedAverage**  
*Weighted averaging (depends on global stats).*
- **Average**  
*Simple averaging.*
- **Median**  
*Somewhat more expensive averaging.*
- **Minimum**  
*Minimum value.*
- **Maximum**  
*Maximum value.*
- **MinMax**  
*Checkerboard of minimum and maximum values.*
- **Decimate**  
*Simple decimation, use first sample.*
- **DecimateSkipNaN**  
*Use first sample that is not NaN.*
- **DecimateRandom**  
*Random decimation using a fixed seed.*
- **AllZero**  
*Just fill the LOD brick with zeroes.*
- **WhiteNoise**  
*Fill with white noise, hope nobody notices.*
- **MostFrequent**  
*The value that occurs most frequently.*
- **MostFrequentNon0**  
*The non-zero value that occurs most frequently.*
- **AverageNon0**  
*Average value, but treat 0 as NaN.*

### 11.2.1 Detailed Description

IZgyReader, IZgyWriter, and other user visible classes.

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.

[OpenZGY::IZgyMeta](#)



- 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.

`OpenZGY::ZgyMetaAndTools` extends `IZgyMeta`

- Add coordinate conversion routines to a `ZgyMeta` instance.

`OpenZGY::ZgyReader` extends `ZgyMetaAndTools` with `read`, `readconst`, `close` `OpenZGY::ZgyWriter` extends `ZgyMetaAndTools` with `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`.

Note that enums and exceptions are problematic when it comes to encapsulation. Enums might be:

- Only visible in the public api. Often end up being mapped to a corresponding internal enum. No problem, but the mapping can be a bit tedious to maintain.
- Only visible internally. Possibly representing integer values written to file. Which makes them an implementation detail, which must absolutely not be exposed publically.
- Defined by the public api but passed unchanged from the api layer to the implementation layer. So there will be an include in some `impl/xxx.h` file to a part of the public API. This is frowned upon.
- Defined by the internal api but may need to be used from applications, i.e. also visible in the public api. There will be an include of e.g. `"impl/ugly.h"` from one of the public header files and/or application code. This is strongly discouraged except for testing.

Exceptions have similar issues. It is possible to catch all exceptions raised in the impl layer and convert those to exceptions owned by the api layer. But re-throwing an exception makes debugging harder.

## 11.3 example.h File Reference

Example program using the C++ API.

```
#include <openzgy/api.h>
#include <iostream>
#include <stdexcept>
#include <stdlib.h>
```

## Functions

- void **copy** (const std::string &srcname, const std::string &dstname)
- int **main** (int argc, const char \*\*argv)

### 11.3.1 Detailed Description

Example program using the C++ API.

To build this e.g. on Ubuntu Xenial: Compile:

```
g++ -std=c++11 -Iinclude -oexample -x c++ example.h -Lxenial-gcc54 -Wl,-rpath-link=xenial-gcc54 -lopenzgy
```

Run:

```
env LD_LIBRARY_PATH=xenial-gcc54 ./example fromfile.zgy tofile.zgy
```

## 11.4 exception.h File Reference

Defines exceptions that may be raised by OpenZGY.

```
#include "declspec.h"
#include <stdexcept>
```

## Classes

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

## Namespaces

- [OpenZGY](#)  
*The entire public API is in this namespace.*
- [OpenZGY::Errors](#)  
*Exceptions that can be thrown by [OpenZGY](#).*

### 11.4.1 Detailed Description

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.

## 11.5 iocontext.h File Reference

Backend specific context.

```
#include "declspec.h"
#include "exception.h"
#include <stdint>
#include <string>
#include <vector>
#include <functional>
```

## Classes

- class [OpenZGY::IOContext](#)  
*Base class for backend specific context.*
- class [OpenZGY::SeismicStoreIOContext](#)  
*Credentials and configuration for Seismic Store.*

## Namespaces

- [OpenZGY](#)  
*The entire public API is in this namespace.*

### 11.5.1 Detailed Description

Backend specific context.

Class IOContext and derivatives are used to hold backend specific information such as authorization tokens.



# Index

- ~ZgyWriter
  - OpenZGY::Impl::ZgyWriter, [65](#)
- aligned
  - OpenZGY::SeismicStoreIOContext, [42](#)
- alturl
  - OpenZGY::Impl::ZgyUtils, [63](#)
  - OpenZGY::IZgyUtils, [30](#)
- api.cpp, [77](#)
- api.h, [78](#)
- bricksiz
  - OpenZGY::ZgyWriterArgs, [71](#)
- close
  - OpenZGY::Impl::ZgyReader, [58](#)
  - OpenZGY::Impl::ZgyWriter, [65](#)
  - OpenZGY::IZgyReader, [25](#)
  - OpenZGY::IZgyWriter, [33](#)
- close\_incomplete
  - OpenZGY::Impl::ZgyWriter, [66](#)
  - OpenZGY::IZgyWriter, [33](#)
- compressor
  - OpenZGY::ZgyWriterArgs, [71](#)
- corners
  - OpenZGY::ZgyWriterArgs, [71](#)
- datarange
  - OpenZGY::ZgyWriterArgs, [72](#)
- debug\_trace
  - OpenZGY::SeismicStoreIOContext, [42](#)
- deletefile
  - OpenZGY::Impl::ZgyUtils, [63](#)
  - OpenZGY::IZgyUtils, [30](#)
- errorflag
  - OpenZGY::Impl::ZgyWriter, [66](#)
- example.h, [81](#)
- exception.h, [82](#)
- filename
  - OpenZGY::ZgyWriterArgs, [72](#)
- finalize
  - OpenZGY::Impl::ZgyWriter, [66](#)
  - OpenZGY::IZgyWriter, [33](#)
- hunit
  - OpenZGY::ZgyWriterArgs, [72](#)
- ilinc
  - OpenZGY::ZgyWriterArgs, [72](#)
- ilstart
  - OpenZGY::ZgyWriterArgs, [73](#)
- iocontext
  - OpenZGY::ZgyWriterArgs, [73](#)
- iocontext.h, [83](#)
- legaltag
  - OpenZGY::SeismicStoreIOContext, [43](#)
- List of exceptions, [15](#)
- lodcompressor
  - OpenZGY::ZgyWriterArgs, [73](#)
- maxhole
  - OpenZGY::SeismicStoreIOContext, [43](#)
- maxsize
  - OpenZGY::SeismicStoreIOContext, [43](#)
- merge
  - OpenZGY::ZgyWriterArgs, [73](#)
- metafrom
  - OpenZGY::ZgyWriterArgs, [73](#)
- OpenZGY, [17](#)
- OpenZGY::Errors, [18](#)
- OpenZGY::Errors::ZgyAborted, [46](#)
- OpenZGY::Errors::ZgyCorruptedFile, [47](#)
- OpenZGY::Errors::ZgyEndOfFile, [48](#)
- OpenZGY::Errors::ZgyError, [49](#)
- OpenZGY::Errors::ZgyFormatError, [50](#)
- OpenZGY::Errors::ZgyInternalError, [50](#)
- OpenZGY::Errors::ZgyIoError, [51](#)
- OpenZGY::Errors::ZgyMissingFeature, [56](#)
- OpenZGY::Errors::ZgySegmentIsClosed, [60](#)
- OpenZGY::Errors::ZgyUserError, [61](#)
- OpenZGY::Formatters, [18](#)
- OpenZGY::Impl, [19](#)
- OpenZGY::Impl::ZgyMeta, [52](#)
- OpenZGY::Impl::ZgyMetaAndTools, [54](#)
- transform, [55](#)
- transform1, [55](#)
- OpenZGY::Impl::ZgyReader, [57](#)
- close, [58](#)
- read, [58](#), [59](#)
- readconst, [59](#)
- ZgyReader, [57](#)
- OpenZGY::Impl::ZgyUtils, [62](#)
- alturl, [63](#)
- deletefile, [63](#)
- ZgyUtils, [62](#)
- OpenZGY::Impl::ZgyWriter, [64](#)
- ~ZgyWriter, [65](#)

- close, 65
- close\_incomplete, 66
- errorflag, 66
- finalize, 66
- set\_errorflag, 67
- write, 67, 68
- writeconst, 68, 69
- ZgyWriter, 65
- OpenZGY::IOContext, 21
  - toString, 21
- OpenZGY::IZgyMeta, 22
- OpenZGY::IZgyReader, 24
  - close, 25
  - read, 25, 26
  - readconst, 26
- OpenZGY::IZgyTools, 27
  - transform, 28
  - transform1, 29
- OpenZGY::IZgyUtils, 29
  - alturl, 30
  - deletefile, 30
  - utils, 31
- OpenZGY::IZgyWriter, 31
  - close, 33
  - close\_incomplete, 33
  - finalize, 33
  - reopen, 34
  - write, 35
  - writeconst, 36
- OpenZGY::ProgressWithDots, 37
  - operator(), 38
  - ProgressWithDots, 38
- OpenZGY::SampleHistogram, 39
- OpenZGY::SampleStatistics, 40
- OpenZGY::SeismicStoreIOContext, 41
  - aligned, 42
  - debug\_trace, 42
  - legaltag, 43
  - maxhole, 43
  - maxsize, 43
  - sdapikey, 43
  - sdtoken, 44
  - sdtokencb, 44
  - sdurl, 44
  - segsize, 44
  - segsplit, 45
  - seismicmeta, 45
  - threads, 45
  - toString, 45
  - writeid, 46
- OpenZGY::ZgyWriterArgs, 69
  - bricksize, 71
  - compressor, 71
  - corners, 71
  - datarange, 72
  - filename, 72
  - hunit, 72
  - ilinc, 72
  - ilstart, 73
  - iocontext, 73
  - lodcompressor, 73
  - merge, 73
  - metafrom, 73
  - size, 74
  - xlinc, 74
  - xlstart, 74
  - zfp\_compressor, 74
  - zfp\_lodcompressor, 75
  - zinc, 75
  - zstart, 75
  - zunit, 75
- operator()
  - OpenZGY::ProgressWithDots, 38
- ProgressWithDots
  - OpenZGY::ProgressWithDots, 38
- read
  - OpenZGY::Impl::ZgyReader, 58, 59
  - OpenZGY::IZgyReader, 25, 26
- readconst
  - OpenZGY::Impl::ZgyReader, 59
  - OpenZGY::IZgyReader, 26
- reopen
  - OpenZGY::IZgyWriter, 34
- sdapikey
  - OpenZGY::SeismicStoreIOContext, 43
- sdtoken
  - OpenZGY::SeismicStoreIOContext, 44
- sdtokencb
  - OpenZGY::SeismicStoreIOContext, 44
- sdurl
  - OpenZGY::SeismicStoreIOContext, 44
- segsize
  - OpenZGY::SeismicStoreIOContext, 44
- segsplit
  - OpenZGY::SeismicStoreIOContext, 45
- seismicmeta
  - OpenZGY::SeismicStoreIOContext, 45
- set\_errorflag
  - OpenZGY::Impl::ZgyWriter, 67
- size
  - OpenZGY::ZgyWriterArgs, 74
- threads
  - OpenZGY::SeismicStoreIOContext, 45
- toString
  - OpenZGY::IOContext, 21
  - OpenZGY::SeismicStoreIOContext, 45
- transform
  - OpenZGY::Impl::ZgyMetaAndTools, 55
  - OpenZGY::IZgyTools, 28
- transform1
  - OpenZGY::Impl::ZgyMetaAndTools, 55
  - OpenZGY::IZgyTools, 29
- utils

- OpenZGY::IZgyUtils, [31](#)
- write
  - OpenZGY::Impl::ZgyWriter, [67](#), [68](#)
  - OpenZGY::IZgyWriter, [35](#)
- writeconst
  - OpenZGY::Impl::ZgyWriter, [68](#), [69](#)
  - OpenZGY::IZgyWriter, [36](#)
- writeid
  - OpenZGY::SeismicStoreIOContext, [46](#)
- xlinc
  - OpenZGY::ZgyWriterArgs, [74](#)
- xlstart
  - OpenZGY::ZgyWriterArgs, [74](#)
- zfp\_compressor
  - OpenZGY::ZgyWriterArgs, [74](#)
- zfp\_lodcompressor
  - OpenZGY::ZgyWriterArgs, [75](#)
- ZgyReader
  - OpenZGY::Impl::ZgyReader, [57](#)
- ZgyUtils
  - OpenZGY::Impl::ZgyUtils, [62](#)
- ZgyWriter
  - OpenZGY::Impl::ZgyWriter, [65](#)
- zinc
  - OpenZGY::ZgyWriterArgs, [75](#)
- zstart
  - OpenZGY::ZgyWriterArgs, [75](#)
- zunit
  - OpenZGY::ZgyWriterArgs, [75](#)