OpenZGY/C++ WORK IN PROGRESS, SUBJECT TO CHANGE

Generated by Doxygen 1.8.13

Contents

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.
- ProgressWithDots example of progress reporting.
- example.h Example application.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Namespace Index

3.1 Namespace List

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

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

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

Chapter 7

Module Documentation

7.1 List of exceptions

Classes

- class OpenZGY::Errors::ZgyError

  Exception base class for all exceptions thrown by OpenZGY.
- class OpenZGY::Errors::ZgyFormatError

  Exception for corrupted or unsupported ZGY file.
- class OpenZGY::Errors::ZgyCorruptedFile

  Exception when 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

  Exception trying to read past EOF.
- class OpenZGY::Errors::ZgySegmentIsClosed

  Exception used internally to request a retry.
- class OpenZGY::Errors::ZgyAborted

  Exception used to abort the computation.
- class OpenZGY::Errors::ZgyMissingFeature

  Exception for missing feature.
- class OpenZGY::Errors::ZgyIoError

  Exception from the I/O layer.

7.1.1 Detailed Description

Chapter 8

Namespace Documentation

8.1 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 ZgyWriterArgs

  Argument package for creating a ZGY file.

Enumerations

- enum SampleDataType { **unknown** = 1000, **int8** = 1001, **int16** = 1002, **float32** = 1003 }
- enum UnitDimension { **unknown** = 2000, **time** = 2001, **length** = 2002, **arcangle** = 2003 }
- enum DecimationType {
  DecimationType::LowPass = 100, DecimationType::WeightedAverage, DecimationType::Average, DecimationType::Median,
  DecimationType::Minimum, DecimationType::Maximum, DecimationType::MinMax, DecimationType::Decimate,
  DecimationType::DecimateSkipNaN, DecimationType::DecimateRandom, DecimationType::AllZero, DecimationType::WhiteNoise,
  DecimationType::MostFrequent, DecimationType::MostFrequentNon0, DecimationType::AverageNon0 }

8.1.1 Detailed Description

The entire public API is in this namespace.

The main interface is IZgyReader and IZgyWriter.

8.1.2 Enumeration Type Documentation

8.1.2.1 DecimationType

```
enum OpenZGY::DecimationType  [strong]
```

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

Maps to LodAlgorithm in the implementation layer.

Enumerator

LowPass	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.

8.1.2.2 SampleDataType

`enum OpenZGY::SampleDataType [strong]`

Value type of samples as stored on file. Data may be read and written using this type or 32-bit float.

8.1.2.3 UnitDimension

`enum OpenZGY::UnitDimension [strong]`

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

8.2 OpenZGY::Errors Namespace Reference

Exceptions that can be thrown by OpenZGY.

Classes

- class ZgyAborted
  Exception used to abort the computation.
- class ZgyCorruptedFile
  Exception when the ZGY file became corrupted while writing to it.
- class ZgyEndOfFile
  Exception trying to read past EOF.
- class ZgyError
  Exception base class for all exceptions thrown by OpenZGY.
- class ZgyFormatError
  Exception for 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
  Exception for missing feature.
- class ZgySegmentIsClosed
  Exception used internally to request a retry.
- class ZgyUserError
  Exception that might be caused by the calling application.

8.2.1 Detailed Description

Exceptions that can be thrown by OpenZGY.

8.3 OpenZGY::Formatters Namespace Reference

operator<< for readable output of enums etc.

Functions

- std::ostream & operator<< (std::ostream &os, SampleDataType value)
  Output the string representation of the input enum type.
- std::ostream & operator<< (std::ostream &os, UnitDimension value)
  Output the string representation of the input enum type.
- 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.

8.3.1 Detailed Description

operator<< for readable output of enums etc.

8.4 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.

8.4.1 Detailed Description

Implementation of the abstract interfaces in OpenZGY.

Chapter 9

Class Documentation

9.1 OpenZGY::IOContext Class Reference

Base class for backend specific context.

`#include <iocontext.h>`

Public Member Functions

- virtual std::string toString () const =0

9.1.1 Detailed Description

Base class for backend specific context.

9.1.2 Member Function Documentation

9.1.2.1 toString()

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

Display the context in a human readable format for debugging.

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

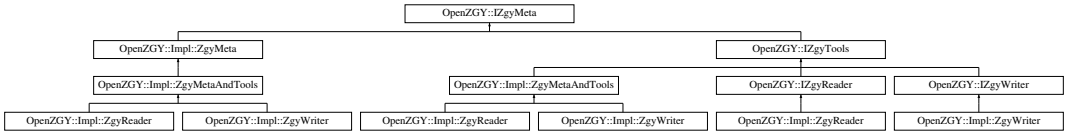
- iocontext.h

9.2 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 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 float32_t zunitfactor () const =0

  Multiply by this factor to convert from storage units to SI units.
- virtual float32_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 meta () const =0
  Dictionary of meta data. NOT IMPLEMENTED.
- virtual int32_t numthreads () const =0
  Number of threads to use. NOT IMPLEMENTED.
- virtual void set_numthreads (int32_t)=0
  Number of threads to use. NOT IMPLEMENTED.
- 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.

9.2.1 Detailed Description

Base class of IZgyReader and IZgyWriter.

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

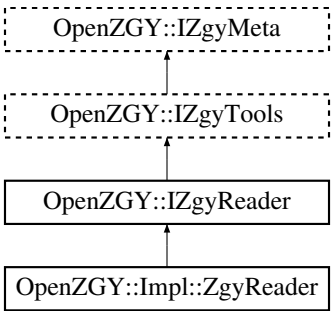
- api.h
- api.cpp

9.3 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

9.3.1 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 scopen inside the ostensibly pure IZgyReader interface. But this reduces the number of classes a library user needs to relate to.

9.3.2 Member Function Documentation

9.3.2.1 close()

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

Close the file and release resources.

The ZgyReader destructor will call close() if not done already, catching and swallowing any exception. Unlike Zgy↩ Writer::close() forgetting to close a file that was only open for read is not a major faux pas. It is still recommended to explicitly close, though.

Implemented in OpenZGY::Impl::ZgyReader.

9.3.2.2 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.

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::ZgyReader.

9.3.2.3 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.

9.3.2.4 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.

9.3.2.5 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).

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.

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.

Implemented in OpenZGY::Impl::ZgyReader.

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

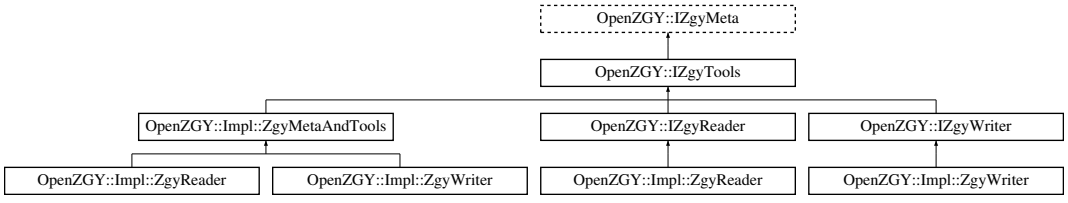
- api.h
- api.cpp

9.4 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

9.4.1 Detailed Description

Base class of IZgyReader and IZgyWriter.

9.4.2 Member Function Documentation

9.4.2.1 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

from	control points in the current coordinate system.
to	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.

9.4.2.2 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

from	control points in the current coordinate system.
to	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:

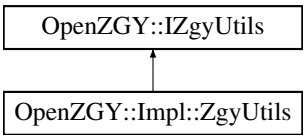
- api.h
- api.cpp

9.5 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.

Static Public Member Functions

- static std::shared_ptr< IZgyUtils > utils (const std::string &prefix, const IOContext *iocontext)

  Create a new concrete instance of IZgyUtils.

9.5.1 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.

9.5.2 Member Function Documentation

9.5.2.1 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.

9.5.2.2 utils()

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

Create a new concrete instance of IZgyUtils.

Parameters

prefix	File name or file name prefix.
iocontext	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 if you have registered a back-end named "xx", both "xx://some/bogus/file.zgy" and just "xx://" will produce an instance that works for your XX backend,

For performance reasons you should consider caching one 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:

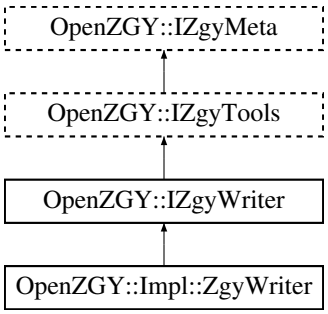
- api.h
- api.cpp

9.6 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) const =0

  Write an arbitrary region.
- virtual void write (const size3i_t &start, const size3i_t &size, const std::int16_t *data) const =0

  Write an arbitrary region with no conversion.
- virtual void write (const size3i_t &start, const size3i_t &size, const std::int8_t *data) const =0

  Write an arbitrary region with no conversion.
- virtual void writeconst (const size3i_t &start, const size3i_t &size, const float *data) const =0

  Write all-constant data.
- virtual void writeconst (const size3i_t &start, const size3i_t &size, const std::int16_t *data) const =0

  Write an arbitrary region with no conversion.
- virtual void writeconst (const size3i_t &start, const size3i_t &size, const std::int8_t *data) const =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 ()=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.

Additional Inherited Members

9.6.1 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 beed 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 scopen inside the ostensibly pure IZgyWriter interface. But this reduces the number of classes a library user needs to relate to.

9.6.2 Member Function Documentation

9.6.2.1 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.

9.6.2.2 finalize()

```
virtual void OpenZGY::IZgyWriter::finalize (
            const std::vector< DecimationType > & decimation = std::vector< Decimation↩
Type >(),
            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

decimation	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.
progress	Function(done, total) called to report progress. If it returns False the computation is aborted. Will be called at least one, even if there is no work.
force	If true, generate the low resolution data even if it appears to not be needed. Use with caution. Especially if writing to the cloud, where data should only be written once.

Implemented in OpenZGY::Impl::ZgyWriter.

9.6.2.3 write() [1/3]

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

Write an arbitrary region.

This will apply conversion float -> storage if needed.

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.

9.6.2.4 write() [2/3]

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

9.6.2.5 write() [3/3]

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

9.6.2.6 writeconst() [1/3]

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

9.6.2.7 writeconst() [2/3]

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

9.6.2.8 writeconst() [3/3]

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

9.7 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.

9.7.1 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.

9.7.2 Constructor & Destructor Documentation

9.7.2.1 ProgressWithDots()

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

Simple progress bar.

Parameters

length	Size of progress bar, default 51 dots.
outfile	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.

9.7.3 Member Function Documentation

9.7.3.1 operator()()

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

Callback invoked to report progress.

Parameters

done	Number of work units done.
total	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

9.8 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

9.8.1 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 (max - min) / (nbins - 1). Bin 0 holds samples with values min_ +/-binwidth/2.

This means that the total range of samples that can be represented in the histogram is actually min-binwidth/2 to max+binwidth/2, which is slightly larger than just min..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 distrubuted 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 distrubuted 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 documentation for this class was generated from the following file:

- api.h

9.9 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.

9.9.1 Detailed Description

Statistics of all sample values on the file.

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

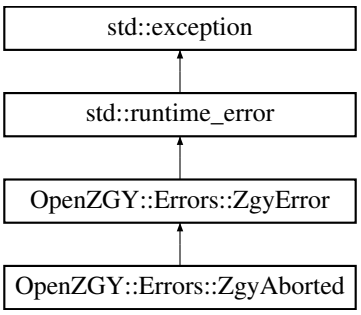
- api.h

9.10 OpenZGY::Errors::ZgyAborted Class Reference

Exception used to abort the computation.

`#include <exception.h>`

Inheritance diagram for OpenZGY::Errors::ZgyAborted:



Public Member Functions

- ZgyAborted (const std::string &arg)

  Exception used to abort the computation.

Additional Inherited Members

9.10.1 Detailed Description

Exception used to abort 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.

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

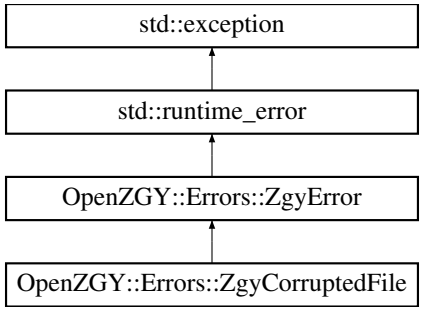
- exception.h

9.11 OpenZGY::Errors::ZgyCorruptedFile Class Reference

Exception when 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)

  Exception when the ZGY file became corrupted while writing to it.

Additional Inherited Members

9.11.1 Detailed Description

Exception when the ZGY file became corrupted while writing to it.

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

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

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

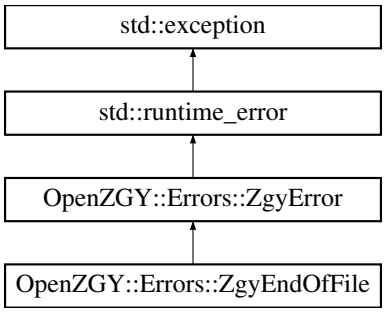
- exception.h

9.12 OpenZGY::Errors::ZgyEndOfFile Class Reference

Exception trying to read past EOF.

`#include <exception.h>`

Inheritance diagram for OpenZGY::Errors::ZgyEndOfFile:



Public Member Functions

- ZgyEndOfFile (const std::string &arg)
  Exception trying to read past EOF.

Additional Inherited Members

9.12.1 Detailed Description

Exception trying to read past EOF.

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

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

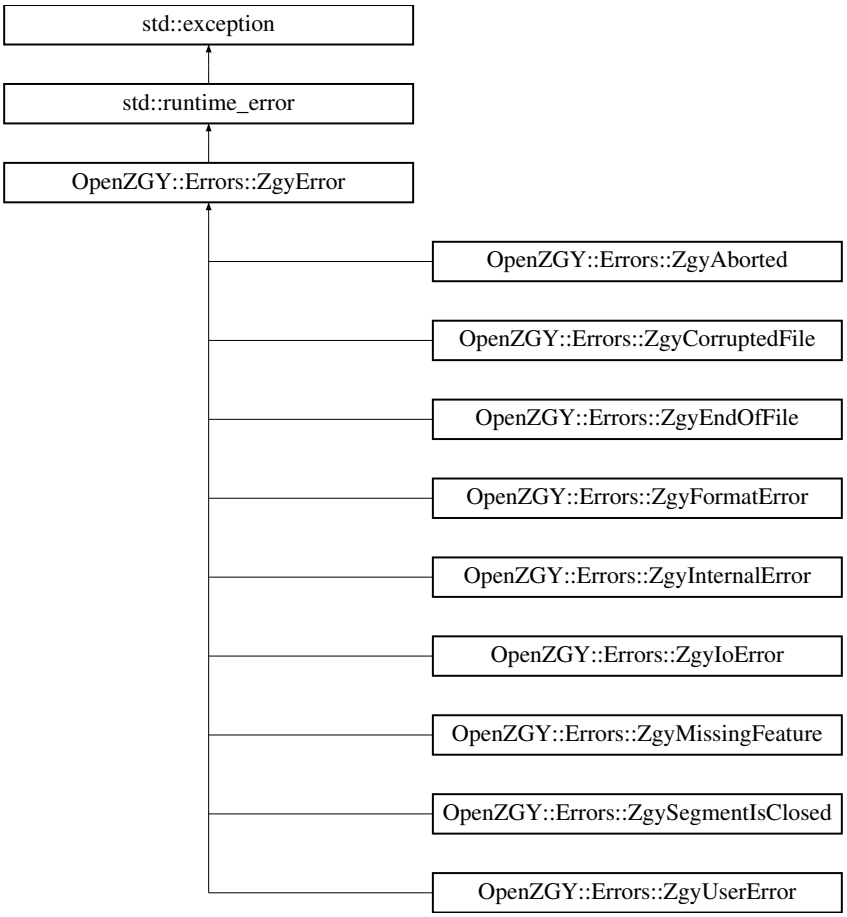
- exception.h

9.13 OpenZGY::Errors::ZgyError Class Reference

Exception 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)

  Exception base class for all exceptions thrown by OpenZGY.

9.13.1 Detailed Description

Exception base class for all exceptions thrown by OpenZGY.

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

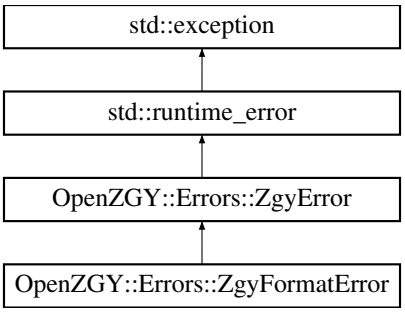
- exception.h

9.14 OpenZGY::Errors::ZgyFormatError Class Reference

Exception for corrupted or unsupported ZGY file.

`#include <exception.h>`

Inheritance diagram for OpenZGY::Errors::ZgyFormatError:



Public Member Functions

- ZgyFormatError (const std::string &arg)

  Exception for corrupted or unsupported ZGY file.

Additional Inherited Members

9.14.1 Detailed Description

Exception for corrupted or unsupported ZGY file.

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

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

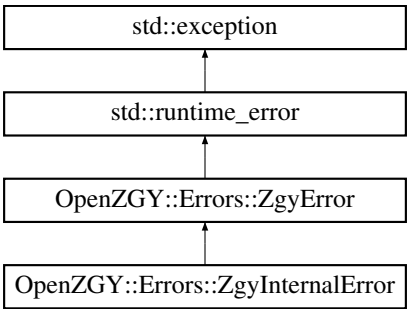
- exception.h

9.15 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

9.15.1 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.

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

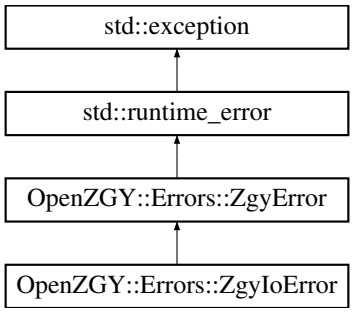
- exception.h

9.16 OpenZGY::Errors::ZgyIoError Class Reference

Exception from the I/O layer.

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

Inheritance diagram for OpenZGY::Errors::ZgyIoError:



Public Member Functions

- ZgyIoError (const std::string &filename, int system_errno)

  Exception from the I/O layer.

Additional Inherited Members

9.16.1 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.

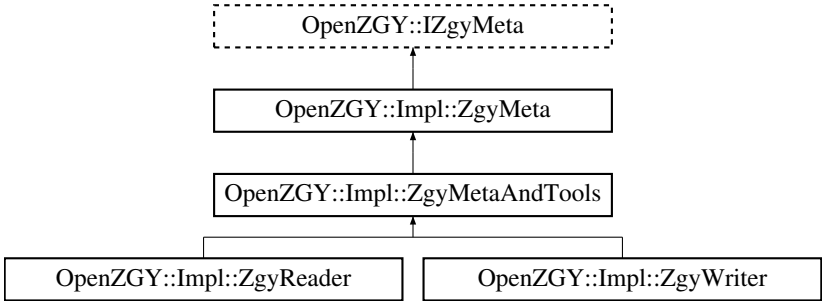
The documentation for this class was generated from the following file:

- exception.h

9.17 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 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 float32_t zunitfactor () const override

  Multiply by this factor to convert from storage units to SI units.
- virtual float32_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 meta () const override

  Dictionary of meta data. NOT IMPLEMENTED.
- virtual int32_t numthreads () const override

  Number of threads to use. NOT IMPLEMENTED.
- virtual void set_numthreads (int32_t) override

  Number of threads to use. NOT IMPLEMENTED.
- 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< InternalZGY::ZgyInternalMeta > _meta

  Handle to the internal metadata layer which this class wraps.

Additional Inherited Members

9.17.1 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 explcitly 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.

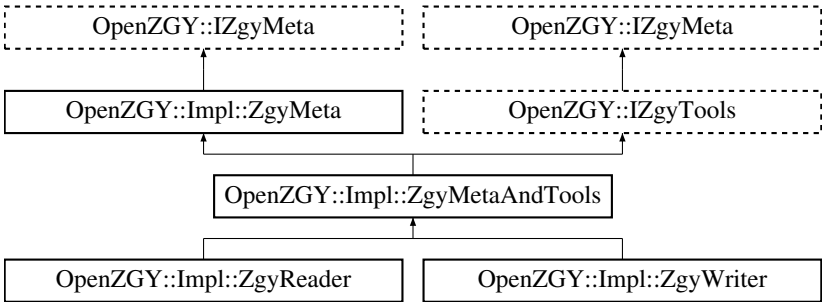
The documentation for this class was generated from the following file:

- api.cpp

9.18 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

9.18.1 Detailed Description

Add coordinate conversion to the concrete ZgyMeta class.

9.18.2 Member Function Documentation

9.18.2.1 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

from	control points in the current coordinate system.
to	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.

9.18.2.2 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

from	control points in the current coordinate system.
to	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:

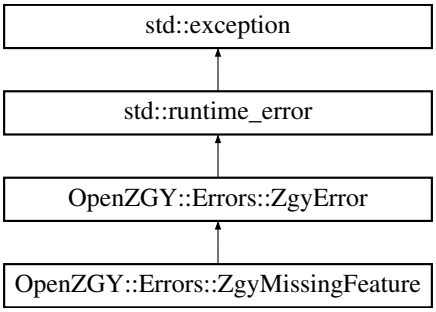
- api.cpp

9.19 OpenZGY::Errors::ZgyMissingFeature Class Reference

Exception for missing feature.

`#include <exception.h>`

Inheritance diagram for OpenZGY::Errors::ZgyMissingFeature:



Public Member Functions

- ZgyMissingFeature (const std::string &arg)

  Exception for missing feature.

Additional Inherited Members

9.19.1 Detailed Description

Exception for missing feature.

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

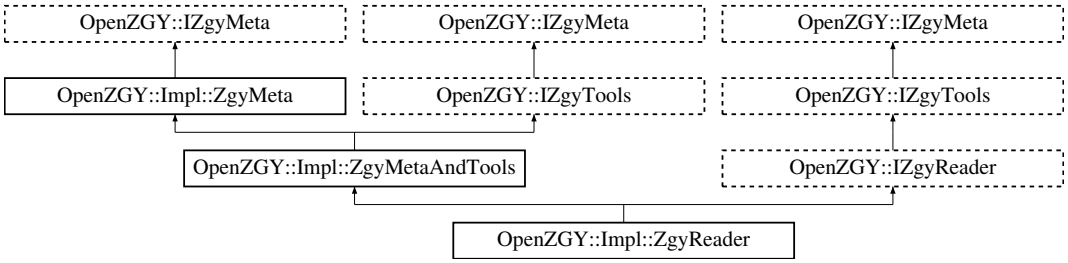
The documentation for this class was generated from the following file:

- exception.h

9.20 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, bool update)

  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

9.20.1 Detailed Description

Concrete implementation of IZgyReader.

9.20.2 Constructor & Destructor Documentation

9.20.2.1 ZgyReader()

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

Open a ZGY file for reading.

9.20.3 Member Function Documentation

9.20.3.1 close()

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

Close the file and release resources.

The ZgyReader destructor will call close() if not done already, catching and swallowing any exception. Unlike ZgyWriter::close() forgetting to close a file that was only open for read is not a major faux pas. It is still recommended to explicitly close, though.

Implements OpenZGY::IZgyReader.

9.20.3.2 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.

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::IZgyReader.

9.20.3.3 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.

9.20.3.4 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.

9.20.3.5 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).

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.

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.

Implements OpenZGY::IZgyReader.

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

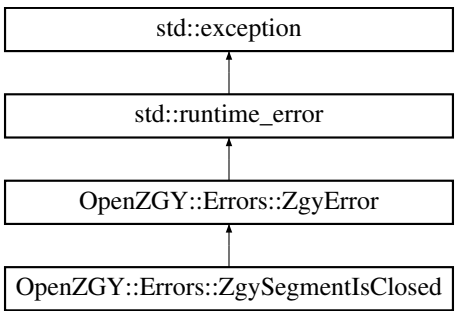
- api.cpp

9.21 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

9.21.1 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.

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

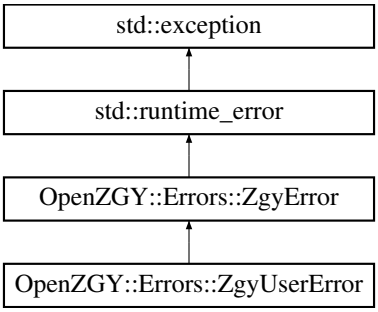
- exception.h

9.22 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

9.22.1 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.

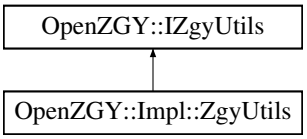
The documentation for this class was generated from the following file:

- exception.h

9.23 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.

Additional Inherited Members

9.23.1 Detailed Description

Concrete implementation of IZgyUtils.

9.23.2 Constructor & Destructor Documentation

9.23.2.1 ZgyUtils()

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

Create a new concrete instance of IZgyUtils.

Parameters

prefix	File name or file name prefix.
iocontext	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 if you have registered a back-end named "xx", both "xx://some/bogus/file.zgy" and just "xx://" will produce an instance that works for your XX backend,

For performance reasons you should consider caching one 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.

9.23.3 Member Function Documentation

9.23.3.1 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.

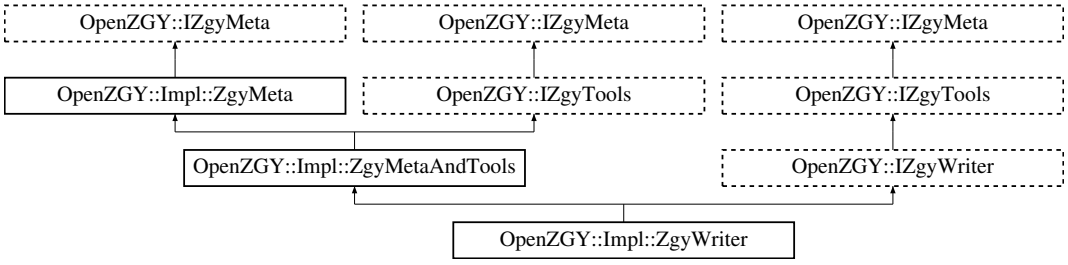
The documentation for this class was generated from the following file:

- api.cpp

9.24 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) const override

  Write an arbitrary region.
- void write (const size3i_t &start, const size3i_t &size, const std::int16_t *data) const override

  Write an arbitrary region with no conversion.
- void write (const size3i_t &start, const size3i_t &size, const std::int8_t *data) const override

  Write an arbitrary region with no conversion.
- void writeconst (const size3i_t &start, const size3i_t &size, const float *data) const override

  Write all-constant data.
- void writeconst (const size3i_t &start, const size3i_t &size, const std::int16_t *data) const override

  Write an arbitrary region with no conversion.
- void writeconst (const size3i_t &start, const size3i_t &size, const std::int8_t *data) const 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 ()

  Flush the file to disk and close it.
- bool errorflag () const override
- void set_errorflag (bool value) override

Additional Inherited Members

9.24.1 Detailed Description

Concrete implementation of IZgyWriter.

9.24.2 Constructor & Destructor Documentation

9.24.2.1 ZgyWriter()

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

Create a ZGY file and open it for writing.

9.24.2.2 ~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.

9.24.3 Member Function Documentation

9.24.3.1 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.

9.24.3.2 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 succeded no operation should manage to corrupt it.

9.24.3.3 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

decimation	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.
progress	Function(done, total) called to report progress. If it returns False the computation is aborted. Will be called at least one, even if there is no work.
force	If true, generate the low resolution data even if it appears to not be needed. Use with caution. Especially if writing to the cloud, where data should only be written once.

Implements OpenZGY::IZgyWriter.

9.24.3.4 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.

9.24.3.5 write() [1/3]

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

Write an arbitrary region.

This will apply conversion float -> storage if needed.

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.

9.24.3.6 write() [2/3]

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

9.24.3.7 write() [3/3]

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

9.24.3.8 writeconst() [1/3]

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

9.24.3.9 writeconst() [2/3]

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

9.24.3.10 writeconst() [3/3]

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

9.25 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)

  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 (const std::string &name, 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 (const std::string &name, 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 & hunit (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.

9.25.1 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});
```

9.25.2 Member Function Documentation

9.25.2.1 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.

9.25.2.2 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.

9.25.2.3 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.

9.25.2.4 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".

9.25.2.5 hunit()

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

Set horizontal unit.

Parameters

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

9.25.2.6 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.

9.25.2.7 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.

9.25.2.8 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.

9.25.2.9 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.

9.25.2.10 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 inadverently shadow something set explicitly.

9.25.2.11 size()

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

Set size of the survey.

Parameters

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

9.25.2.12 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.

9.25.2.13 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.

9.25.2.14 zfp_compressor()

```
ZgyWriterArgs & OpenZGY::ZgyWriterArgs::zfp_compressor (
            const std::string & name,
            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.

9.25.2.15 zfp_lodcompressor()

```
ZgyWriterArgs & OpenZGY::ZgyWriterArgs::zfp_lodcompressor (
            const std::string & name,
            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.

9.25.2.16 zinc()

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

Set increment (distance between samples) in vertical direction.

Vertical annotation is generally safe to have non-integral.

9.25.2.17 zstart()

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

Set first time/depth.

Vertical annotation is generally safe to have non-integral.

9.25.2.18 zunit()

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

Set vertical unit.

Parameters

dimension	time or depth (a.k.a. length).
name	for annotation only.
factor	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 10

File Documentation

10.1 api.cpp File Reference

Implements the pure interfaces of the API.

```
#include "api.h"
#include "exception.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.
- std::ostream & OpenZGY::Formatters::operator<< (std::ostream &os, SampleDataType value)

  Output the string representation of the input enum type.
- std::ostream & OpenZGY::Formatters::operator<< (std::ostream &os, UnitDimension value)

  Output the string representation of the input enum type.
- std::ostream & OpenZGY::Formatters::operator<< (std::ostream &os, DecimationType value)

  Output the string representation of the input enum type.

10.1.1 Detailed Description

Implements the pure interfaces of the API.

10.2 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>
```

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.

Enumerations

- enum OpenZGY::SampleDataType { **unknown** = 1000, **int8** = 1001, **int16** = 1002, **float32** = 1003 }
- enum OpenZGY::UnitDimension { **unknown** = 2000, **time** = 2001, **length** = 2002, **arcangle** = 2003 }
- enum OpenZGY::DecimationType {
  OpenZGY::DecimationType::LowPass = 100, OpenZGY::DecimationType::WeightedAverage, OpenZGY::↩
  DecimationType::Average, OpenZGY::DecimationType::Median,
  OpenZGY::DecimationType::Minimum, OpenZGY::DecimationType::Maximum, OpenZGY::Decimation↩
  Type::MinMax, OpenZGY::DecimationType::Decimate,
  OpenZGY::DecimationType::DecimateSkipNaN, OpenZGY::DecimationType::DecimateRandom, OpenZG↩
  Y::DecimationType::AllZero, OpenZGY::DecimationType::WhiteNoise,
  OpenZGY::DecimationType::MostFrequent, OpenZGY::DecimationType::MostFrequentNon0, OpenZGY::↩
  DecimationType::AverageNon0 }

Functions

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

10.2.1 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 Zgy↩MetaAndTools 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 debuggig harder.

10.3 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)

10.3.1 Detailed Description

Example program using the C++ API.

To build this e.g. on Ububtu 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
```

10.4 exception.h File Reference

Defines exceptions that may be raised by OpenZGY.

```
#include <stdexcept>
```

Classes

- class OpenZGY::Errors::ZgyError
  Exception base class for all exceptions thrown by OpenZGY.
- class OpenZGY::Errors::ZgyFormatError
  Exception for corrupted or unsupported ZGY file.
- class OpenZGY::Errors::ZgyCorruptedFile
  Exception when 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
  Exception trying to read past EOF.
- class OpenZGY::Errors::ZgySegmentIsClosed
  Exception used internally to request a retry.
- class OpenZGY::Errors::ZgyAborted
  Exception used to abort the computation.
- class OpenZGY::Errors::ZgyMissingFeature
  Exception for 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.

10.4.1 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.

10.5 iocontext.h File Reference

Backend specific context.

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

Classes

- class OpenZGY::IOContext

  Base class for backend specific context.

Namespaces

- OpenZGY

  The entire public API is in this namespace.

10.5.1 Detailed Description

Backend specific context.

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

Index