

# Alpha version. Expect changes.

OpenZGY/C++ API and Internals (ALPHA)

Generated by Doxygen 1.8.17

Alpha version. Expect changes.

# Alpha version. Expect changes.

i

<b>1 Main Page</b>	<b>1</b>
<b>2 Example</b>	<b>3</b>
<b>3 Physical layout of ZGY files</b>	<b>5</b>
<b>4 Implementation notes</b>	<b>7</b>
<b>5 Algorithms for decimation</b>	<b>9</b>
<b>6 Migration guide from ZGY-Public / ZGY-Cloud to OpenZGY</b>	<b>11</b>
<b>7 Module Index</b>	<b>13</b>
7.1 Modules . . . . .	13
<b>8 Namespace Index</b>	<b>15</b>
8.1 Namespace List . . . . .	15
<b>9 Hierarchical Index</b>	<b>17</b>
9.1 Class Hierarchy . . . . .	17
<b>10 Class Index</b>	<b>21</b>
10.1 Class List . . . . .	21
<b>11 File Index</b>	<b>25</b>
11.1 File List . . . . .	25
<b>12 Module Documentation</b>	<b>27</b>
12.1 List of exceptions . . . . .	27
12.1.1 Detailed Description . . . . .	27
<b>13 Namespace Documentation</b>	<b>29</b>
13.1 InternalZGY Namespace Reference . . . . .	29
13.1.1 Detailed Description . . . . .	33
13.1.2 Typedef Documentation . . . . .	33
13.1.2.1 compressor_t . . . . .	34
13.1.2.2 index3_t . . . . .	34
13.1.2.3 rawdata_t . . . . .	34
13.1.3 Enumeration Type Documentation . . . . .	34
13.1.3.1 BrickStatus . . . . .	34
13.1.3.2 LodAlgorithm . . . . .	34
13.1.3.3 RawCoordType . . . . .	35
13.1.3.4 RawDataType . . . . .	35
13.1.3.5 RawGridDefinition . . . . .	35
13.1.3.6 RawHorizontalDimension . . . . .	35
13.1.3.7 RawVerticalDimension . . . . .	36
13.1.3.8 UpdateMode . . . . .	36

# Alpha version. Expect changes.

ii

13.1.4 Function Documentation	36
13.1.4.1 align()	36
13.1.4.2 array_to_hex()	37
13.1.4.3 array_to_string()	37
13.1.4.4 byteswapV1Long()	37
13.1.4.5 createGenericLevelOfDetail()	37
13.1.4.6 createGenericLevelOfDetailLoPass()	38
13.1.4.7 createLevelOfDetail()	38
13.1.4.8 createLod()	38
13.1.4.9 createLodMT()	39
13.1.4.10 createLodPart()	39
13.1.4.11 createLodST()	40
13.1.4.12 createLodT()	40
13.1.4.13 IsFiniteT()	40
13.1.4.14 IsNanT()	40
13.1.4.15 ptr_to_array()	41
13.1.4.16 ptr_to_hex()	41
13.1.4.17 ptr_to_string()	41
13.1.4.18 RoundAndClip() [1/2]	41
13.1.4.19 RoundAndClip() [2/2]	41
13.1.4.20 subtiling()	42
13.2 OpenZGY Namespace Reference	42
13.2.1 Detailed Description	43
13.3 OpenZGY::Errors Namespace Reference	43
13.3.1 Detailed Description	43
13.4 OpenZGY::Formatters Namespace Reference	43
13.4.1 Detailed Description	44
13.5 OpenZGY::Impl Namespace Reference	44
13.5.1 Detailed Description	44
<b>14 Class Documentation</b>	<b>45</b>
14.1 InternalZGY::CompressFactoryImpl Class Reference	45
14.1.1 Detailed Description	45
14.1.2 Member Function Documentation	45
14.1.2.1 decompress()	46
14.1.2.2 knownCompressors()	46
14.1.2.3 knownDecompressors()	46
14.1.2.4 registerDecompressor()	46
14.2 InternalZGY::Config Class Reference	46
14.2.1 Detailed Description	47
14.3 InternalZGY::DataBuffer Class Reference	47
14.3.1 Detailed Description	48

14.3.2 Member Function Documentation	50
14.3.2.1 clone()	51
14.3.2.2 copyFrom()	51
14.3.2.3 makeDataBuffer3d()	51
14.3.2.4 scaleToFloat()	52
14.3.2.5 scaleToStorage()	52
14.4 InternalZGY::DataBufferNd< T, NDim > Class Template Reference	52
14.4.1 Member Function Documentation	54
14.4.1.1 clear()	54
14.4.1.2 clone()	54
14.4.1.3 copyFrom()	54
14.4.1.4 fill()	55
14.4.1.5 s_scaleFromFloat()	55
14.4.1.6 s_scaleToFloat()	55
14.4.1.7 scaleToFloat()	56
14.4.1.8 scaleToStorage()	56
14.4.1.9 size3d()	56
14.4.1.10 sizeptr()	56
14.4.1.11 slice()	57
14.4.1.12 slice1()	57
14.4.1.13 stride3d()	57
14.4.1.14 strideptr()	58
14.5 InternalZGY::OrderedCornerPoints::Element Struct Reference	58
14.5.1 Detailed Description	58
14.5.2 Constructor & Destructor Documentation	58
14.5.2.1 Element() [1/3]	58
14.5.2.2 Element() [2/3]	59
14.5.2.3 Element() [3/3]	59
14.5.3 Member Data Documentation	59
14.5.3.1 il	59
14.5.3.2 j	59
14.5.3.3 x	59
14.5.3.4 xl	59
14.5.3.5 y	60
14.6 OpenZGY::Impl::EnumMapper Class Reference	60
14.6.1 Detailed Description	60
14.6.2 Member Function Documentation	60
14.6.2.1 mapDecimationTypeToLodAlgorithm() [1/2]	60
14.6.2.2 mapDecimationTypeToLodAlgorithm() [2/2]	61
14.6.2.3 mapRawDataTypeToSampleDataType()	61
14.6.2.4 mapRawHorizontalDimensionToUnitDimension()	61
14.6.2.5 mapRawVerticalDimensionToUnitDimension()	61

# Alpha version. Expect changes.

iv

14.6.2.6 mapSampleDataTypeToRawDataType()	61
14.6.2.7 mapUnitDimensionToRawHorizontalDimension()	61
14.6.2.8 mapUnitDimensionToRawVerticalDimension()	62
14.7 InternalZGY::Environment Class Reference	62
14.7.1 Detailed Description	62
14.7.2 Member Function Documentation	62
14.7.2.1 getNumericEnv()	62
14.7.2.2 getStringEnv()	63
14.8 InternalZGY::ZgyInternalMeta::ErrorsWillCorruptFile Class Reference	63
14.8.1 Detailed Description	63
14.9 InternalZGY::ZgyInternalBulk::ErrorsWillCorruptFile Class Reference	63
14.9.1 Detailed Description	64
14.10 InternalZGY::FileADT Class Reference	64
14.10.1 Member Function Documentation	65
14.10.1.1 xx_close()	65
14.10.1.2 xx_eof()	65
14.10.1.3 xx_iscloud()	65
14.10.1.4 xx_read()	66
14.10.1.5 xx_readv()	66
14.10.1.6 xx_threadsafe()	66
14.10.1.7 xx_write()	67
14.11 InternalZGY::FileCommon Class Reference	67
14.11.1 Detailed Description	68
14.11.2 Member Function Documentation	68
14.11.2.1 _check_short_read()	68
14.11.2.2 _real_eof()	68
14.12 InternalZGY::FileConfig Class Reference	68
14.13 InternalZGY::FileFactory Class Reference	69
14.13.1 Member Function Documentation	69
14.13.1.1 create()	69
14.14 InternalZGY::FileHeaderAccess Class Reference	70
14.15 InternalZGY::FileHeaderPOD Class Reference	70
14.16 InternalZGY::FileUtilsSeismicStore Class Reference	71
14.17 InternalZGY::GenLodBase Class Reference	71
14.17.1 Detailed Description	72
14.17.2 Constructor & Destructor Documentation	72
14.17.2.1 GenLodBase()	72
14.17.3 Member Function Documentation	73
14.17.3.1 _format_result()	73
14.17.3.2 _prefix()	73
14.17.3.3 _read()	73
14.17.3.4 _report()	73

14.17.3.5 _savestats()	74
14.17.3.6 _write()	74
14.18 InternalZGY::GenLodC Class Reference	74
14.18.1 Constructor & Destructor Documentation	75
14.18.1.1 GenLodC()	75
14.18.2 Member Function Documentation	75
14.18.2.1 _read()	75
14.18.2.2 _savestats()	76
14.18.2.3 _write()	76
14.19 InternalZGY::GenLodImpl Class Reference	76
14.19.1 Detailed Description	77
14.19.2 Constructor & Destructor Documentation	77
14.19.2.1 GenLodImpl()	77
14.19.3 Member Function Documentation	78
14.19.3.1 _accumulate()	78
14.19.3.2 _calculate()	78
14.19.3.3 _decimate()	78
14.19.3.4 _paste1()	79
14.19.3.5 _paste4()	79
14.19.3.6 call()	79
14.19.3.7 suggestHistogramRange()	80
14.20 InternalZGY::GUID Class Reference	80
14.20.1 Detailed Description	81
14.20.2 Member Function Documentation	81
14.20.2.1 copyTo()	81
14.21 InternalZGY::HeaderAccessFactory Class Reference	81
14.22 InternalZGY::HistHeaderAccess Class Reference	82
14.23 InternalZGY::HistHeaderV1Access Class Reference	82
14.24 InternalZGY::HistHeaderV1POD Class Reference	83
14.24.1 Detailed Description	83
14.25 InternalZGY::HistHeaderV2Access Class Reference	83
14.26 InternalZGY::HistHeaderV2POD Class Reference	84
14.26.1 Detailed Description	84
14.27 InternalZGY::HistogramBuilder Class Reference	84
14.27.1 Detailed Description	85
14.27.2 Constructor & Destructor Documentation	85
14.27.2.1 HistogramBuilder() [1/2]	85
14.27.2.2 HistogramBuilder() [2/2]	86
14.27.3 Member Function Documentation	86
14.27.3.1 add()	86
14.27.3.2 gethiststats()	86
14.27.3.3 operator"!=()	86

# Alpha version. Expect changes.

vi

14.27.3.4 operator*=( )	87
14.27.3.5 operator+=( )	87
14.27.3.6 operator-=( )	87
14.27.3.7 operator==( )	87
14.27.3.8 scale( )	87
14.28 InternalZGY::HistogramData Class Reference	88
14.28.1 Detailed Description	88
14.28.2 Constructor & Destructor Documentation	89
14.28.2.1 HistogramData( ) [ 1 / 3 ]	89
14.28.2.2 HistogramData( ) [ 2 / 3 ]	89
14.28.2.3 HistogramData( ) [ 3 / 3 ]	89
14.28.3 Member Function Documentation	89
14.28.3.1 calculateConversionFactors( )	89
14.28.3.2 clear( )	90
14.28.3.3 get( )	90
14.28.3.4 getcount( )	90
14.28.3.5 getLinearTransform( )	90
14.28.3.6 operator!=( )	90
14.28.3.7 operator*=( )	90
14.28.3.8 operator+=( )	91
14.28.3.9 operator-=( )	91
14.28.3.10 operator=( )	91
14.28.3.11 operator==( )	91
14.28.3.12 scale( )	91
14.29 InternalZGY::IFileHeaderAccess Class Reference	92
14.30 InternalZGY::IHeaderAccess Class Reference	92
14.30.1 Detailed Description	93
14.31 InternalZGY::IHistHeaderAccess Class Reference	93
14.32 InternalZGY::IInfoHeaderAccess Class Reference	94
14.33 InternalZGY::IJK Struct Reference	95
14.34 InternalZGY::ILookupTableAccess Class Reference	95
14.35 InternalZGY::SummaryTimer::Impl Class Reference	96
14.36 InternalZGY::ImplicitLinearTransform2d Class Reference	96
14.36.1 Detailed Description	97
14.36.2 Member Typedef Documentation	97
14.36.2.1 this_type	97
14.36.3 Constructor & Destructor Documentation	97
14.36.3.1 ImplicitLinearTransform2d( ) [ 1 / 2 ]	98
14.36.3.2 ImplicitLinearTransform2d( ) [ 2 / 2 ]	98
14.36.4 Member Function Documentation	98
14.36.4.1 Apply( )	98
14.36.4.2 Compare( )	98



14.36.4.3 Identity()	99
14.36.4.4 makeTiePoint()	99
14.36.4.5 operator"!="()	99
14.36.4.6 operator==(())	99
14.36.4.7 Push()	100
14.36.4.8 Rotate()	100
14.36.4.9 Scale()	100
14.36.4.10 Translate()	101
14.37 InternalZGY::ImplicitLinearTransform2dImp Class Reference	101
14.37.1 Detailed Description	101
14.37.2 Member Function Documentation	102
14.37.2.1 Mult()	102
14.38 InternalZGY::InfoHeaderAccess Class Reference	102
14.39 InternalZGY::InfoHeaderV1Access Class Reference	103
14.39.1 Member Function Documentation	104
14.39.1.1 calculate_cache()	104
14.39.1.2 calculate_read()	104
14.39.1.3 calculate_write()	105
14.40 InternalZGY::InfoHeaderV1POD Class Reference	105
14.40.1 Detailed Description	105
14.41 InternalZGY::InfoHeaderV2Access Class Reference	105
14.41.1 Member Function Documentation	107
14.41.1.1 calculate_cache()	107
14.41.1.2 calculate_read()	107
14.41.1.3 calculate_write()	108
14.42 InternalZGY::InfoHeaderV2POD Class Reference	108
14.42.1 Detailed Description	109
14.43 OpenZGY::IOContext Class Reference	109
14.43.1 Detailed Description	109
14.43.2 Constructor & Destructor Documentation	109
14.43.2.1 ~IOContext()	109
14.43.3 Member Function Documentation	109
14.43.3.1 toString()	110
14.44 InternalZGY::OffsetHeaderAccess Class Reference	110
14.45 OpenZGY::IZgyMeta Class Reference	111
14.45.1 Detailed Description	112
14.46 OpenZGY::IZgyReader Class Reference	112
14.46.1 Detailed Description	113
14.46.2 Member Function Documentation	113
14.46.2.1 close()	113
14.46.2.2 read() [1/3]	114
14.46.2.3 read() [2/3]	114

# Alpha version. Expect changes.

viii

14.46.2.4 read() [3/3]	114
14.46.2.5 readconst()	115
14.47 OpenZGY::IZgyTools Class Reference	115
14.47.1 Detailed Description	116
14.47.2 Member Function Documentation	116
14.47.2.1 transform()	116
14.47.2.2 transform1()	117
14.48 OpenZGY::IZgyUtils Class Reference	117
14.48.1 Detailed Description	118
14.48.2 Member Function Documentation	118
14.48.2.1 deletefile()	118
14.48.2.2 utils()	118
14.49 OpenZGY::IZgyWriter Class Reference	119
14.49.1 Detailed Description	120
14.49.2 Member Function Documentation	120
14.49.2.1 close()	120
14.49.2.2 close_incomplete()	121
14.49.2.3 finalize()	121
14.49.2.4 write() [1/3]	121
14.49.2.5 write() [2/3]	122
14.49.2.6 write() [3/3]	122
14.49.2.7 writeconst() [1/3]	123
14.49.2.8 writeconst() [2/3]	123
14.49.2.9 writeconst() [3/3]	123
14.50 InternalZGY::LocalFileLinux Class Reference	124
14.50.1 Member Function Documentation	124
14.50.1.1 _real_eof()	124
14.50.1.2 xx_close()	125
14.50.1.3 xx_eof()	125
14.50.1.4 xx_iscloud()	125
14.50.1.5 xx_read()	125
14.50.1.6 xx_readv()	126
14.50.1.7 xx_threadsafe()	126
14.50.1.8 xx_write()	126
14.51 InternalZGY::LodAllZero< T > Class Template Reference	127
14.51.1 Detailed Description	127
14.52 InternalZGY::LodAverage< T > Class Template Reference	127
14.52.1 Detailed Description	127
14.53 InternalZGY::LodAverage< float > Class Reference	128
14.53.1 Detailed Description	128
14.54 InternalZGY::LodAverageNon0< T > Class Template Reference	128
14.54.1 Detailed Description	128

14.55 InternalZGY::LodAverageNon0< float > Class Reference . . . . .	129
14.55.1 Detailed Description . . . . .	129
14.56 InternalZGY::LodDecimate< T > Class Template Reference . . . . .	129
14.56.1 Detailed Description . . . . .	129
14.57 InternalZGY::LodDecimateSkipNaN< T > Class Template Reference . . . . .	130
14.57.1 Detailed Description . . . . .	130
14.58 InternalZGY::LodDecimateSkipNaN< float > Class Reference . . . . .	130
14.58.1 Detailed Description . . . . .	130
14.59 InternalZGY::LodMaximum< T > Class Template Reference . . . . .	131
14.59.1 Detailed Description . . . . .	131
14.60 InternalZGY::LodMedian< T > Class Template Reference . . . . .	131
14.60.1 Detailed Description . . . . .	131
14.61 InternalZGY::LodMedian< float > Class Reference . . . . .	132
14.61.1 Detailed Description . . . . .	132
14.62 InternalZGY::LodMinimum< T > Class Template Reference . . . . .	132
14.62.1 Detailed Description . . . . .	132
14.63 InternalZGY::LodMinMax< T > Class Template Reference . . . . .	133
14.63.1 Detailed Description . . . . .	133
14.64 InternalZGY::LodMostFrequent< T, SkipNaN, SkipZero > Class Template Reference . . . . .	133
14.64.1 Detailed Description . . . . .	134
14.65 InternalZGY::LodSampling Class Reference . . . . .	134
14.65.1 Detailed Description . . . . .	134
14.65.2 Member Function Documentation . . . . .	134
14.65.2.1 downSample1D() . . . . .	134
14.66 InternalZGY::LodWeightedAverage< T > Class Template Reference . . . . .	135
14.66.1 Detailed Description . . . . .	135
14.67 InternalZGY::Logger Class Reference . . . . .	136
14.67.1 Detailed Description . . . . .	136
14.67.2 Member Function Documentation . . . . .	136
14.67.2.1 getCallback() . . . . .	136
14.67.2.2 getForwarder() . . . . .	137
14.68 InternalZGY::LoggerBase Class Reference . . . . .	137
14.68.1 Detailed Description . . . . .	138
14.68.2 Member Function Documentation . . . . .	138
14.68.2.1 emptyCallback() . . . . .	138
14.68.2.2 logger() [1/2] . . . . .	138
14.68.2.3 logger() [2/2] . . . . .	139
14.68.2.4 standardCallback() . . . . .	139
14.69 InternalZGY::LookupTable Class Reference . . . . .	139
14.69.1 Detailed Description . . . . .	140
14.69.2 Member Function Documentation . . . . .	140
14.69.2.1 calcLookupSize() . . . . .	141

# Alpha version. Expect changes.

x

14.70 InternalZGY::LookupTableV0Access Class Reference	141
14.70.1 Detailed Description	142
14.71 InternalZGY::LookupTable::LutInfo Struct Reference	142
14.71.1 Detailed Description	143
14.72 InternalZGY::LutInfoEx Struct Reference	143
14.72.1 Detailed Description	143
14.73 InternalZGY::NullCompressPlugin Class Reference	144
14.73.1 Detailed Description	144
14.73.2 Member Function Documentation	144
14.73.2.1 compress()	144
14.73.2.2 decompress()	145
14.73.2.3 getCompressor()	146
14.74 InternalZGY::OffsetHeaderAccess Class Reference	147
14.75 InternalZGY::OffsetHeaderV1Access Class Reference	147
14.76 InternalZGY::OffsetHeaderV1POD Class Reference	148
14.76.1 Detailed Description	148
14.77 InternalZGY::OffsetHeaderV2Access Class Reference	148
14.77.1 Member Function Documentation	149
14.77.1.1 calculate()	149
14.78 InternalZGY::OffsetHeaderV2POD Class Reference	150
14.78.1 Detailed Description	150
14.79 InternalZGY::OrderedCornerPoints Class Reference	150
14.79.1 Detailed Description	151
14.79.2 Member Typedef Documentation	151
14.79.2.1 annot_type	151
14.79.2.2 coord_type	151
14.79.3 Member Enumeration Documentation	151
14.79.3.1 anonymous enum	151
14.79.4 Constructor & Destructor Documentation	152
14.79.4.1 OrderedCornerPoints() [1/3]	152
14.79.4.2 OrderedCornerPoints() [2/3]	152
14.79.4.3 OrderedCornerPoints() [3/3]	152
14.79.5 Member Function Documentation	153
14.79.5.1 operator[]()	153
14.80 InternalZGY::PrintingTimer Class Reference	154
14.80.1 Detailed Description	154
14.81 OpenZGY::ProgressWithDots Class Reference	154
14.81.1 Detailed Description	155
14.81.2 Constructor & Destructor Documentation	155
14.81.2.1 ProgressWithDots()	155
14.81.3 Member Function Documentation	156
14.81.3.1 operator()()	156

14.82 InternalZGY::PushEnvironment Class Reference	156
14.82.1 Detailed Description	156
14.82.2 Constructor & Destructor Documentation	157
14.82.2.1 PushEnvironment()	157
14.83 InternalZGY::RawDataTypeDetails Class Reference	157
14.83.1 Detailed Description	157
14.84 InternalZGY::RawDataTypeTraits< T > Struct Template Reference	157
14.84.1 Detailed Description	158
14.85 InternalZGY::RawDataTypeTraits< float > Struct Reference	158
14.86 InternalZGY::RawDataTypeTraits< std::int16_t > Struct Reference	158
14.87 InternalZGY::RawDataTypeTraits< std::int32_t > Struct Reference	158
14.88 InternalZGY::RawDataTypeTraits< std::int8_t > Struct Reference	159
14.89 InternalZGY::RawDataTypeTraits< std::uint16_t > Struct Reference	159
14.90 InternalZGY::RawDataTypeTraits< std::uint32_t > Struct Reference	159
14.91 InternalZGY::RawDataTypeTraits< std::uint8_t > Struct Reference	159
14.92 InternalZGY::RawPrintingTimer Class Reference	160
14.92.1 Detailed Description	160
14.93 InternalZGY::ReadRequest Class Reference	160
14.94 InternalZGY::NullCompressPlugin::Register Class Reference	161
14.94.1 Detailed Description	161
14.94.2 Constructor & Destructor Documentation	161
14.94.2.1 Register()	161
14.94.2.2 ~Register()	162
14.95 OpenZGY::SampleHistogram Class Reference	162
14.95.1 Detailed Description	163
14.96 OpenZGY::SampleStatistics Class Reference	163
14.96.1 Detailed Description	164
14.97 OpenZGY::SeismicStoreIOContext Class Reference	164
14.97.1 Detailed Description	165
14.97.2 Member Function Documentation	165
14.97.2.1 aligned()	165
14.97.2.2 debug_trace()	165
14.97.2.3 legaltag()	165
14.97.2.4 maxhole()	166
14.97.2.5 maxsize()	166
14.97.2.6 sdapikey()	166
14.97.2.7 sdtoken()	167
14.97.2.8 sdtokencb()	167
14.97.2.9 sdurl()	167
14.97.2.10 segsize()	167
14.97.2.11 seismicmeta()	167
14.97.2.12 threads()	168

# Alpha version. Expect changes.

xii

14.97.2.13 toString()	168
14.97.2.14 writeid()	168
14.98 InternalZGY::SimpleTimer Class Reference	168
14.98.1 Detailed Description	169
14.99 InternalZGY::StatisticData Class Reference	169
14.99.1 Detailed Description	170
14.99.2 Constructor & Destructor Documentation	170
14.99.2.1 StatisticData() [1/3]	170
14.99.2.2 StatisticData() [2/3]	170
14.99.2.3 StatisticData() [3/3]	170
14.99.3 Member Function Documentation	170
14.99.3.1 add()	171
14.99.3.2 operator*=(())	171
14.99.3.3 operator+=(())	171
14.99.3.4 operator-=(())	171
14.99.3.5 scale()	171
14.99.3.6 trimRange()	172
14.100 InternalZGY::SummaryPrintingTimer Class Reference	172
14.100.1 Detailed Description	173
14.101 InternalZGY::SummaryTimer Class Reference	173
14.101.1 Detailed Description	174
14.101.2 Member Function Documentation	174
14.101.2.1 add() [1/2]	174
14.101.2.2 add() [2/2]	174
14.101.2.3 reset()	174
14.102 InternalZGY::ImplicitLinearTransform2d::TiePoint Struct Reference	175
14.102.1 Detailed Description	175
14.102.2 Constructor & Destructor Documentation	175
14.102.2.1 TiePoint() [1/2]	175
14.102.2.2 TiePoint() [2/2]	175
14.102.3 Member Data Documentation	176
14.102.3.1 b	176
14.103 InternalZGY::Timer Class Reference	176
14.103.1 Constructor & Destructor Documentation	177
14.103.1.1 Timer()	177
14.103.2 Member Function Documentation	177
14.103.2.1 getValue()	177
14.103.2.2 getValue_s()	178
14.104 InternalZGY::TmpLookupEntry Struct Reference	178
14.105 OpenZGY::Errors::ZgyAborted Class Reference	179
14.105.1 Detailed Description	179
14.106 OpenZGY::Errors::ZgyCorruptedFile Class Reference	179

14.106.1 Detailed Description . . . . .	180
14.107 OpenZGY::Errors::ZgyEndOfFile Class Reference . . . . .	180
14.107.1 Detailed Description . . . . .	181
14.108 OpenZGY::Errors::ZgyError Class Reference . . . . .	181
14.108.1 Detailed Description . . . . .	182
14.109 OpenZGY::Errors::ZgyFormatError Class Reference . . . . .	182
14.109.1 Detailed Description . . . . .	182
14.110 InternalZGY::ZgyInternalBulk Class Reference . . . . .	183
14.110.1 Detailed Description . . . . .	183
14.110.2 Member Function Documentation . . . . .	183
14.110.2.1 readConstantValue() . . . . .	184
14.110.2.2 readToExistingBuffer() . . . . .	184
14.110.2.3 readToNewBuffer() . . . . .	184
14.111 OpenZGY::Errors::ZgyInternalError Class Reference . . . . .	185
14.111.1 Detailed Description . . . . .	185
14.112 InternalZGY::ZgyInternalMeta Class Reference . . . . .	185
14.112.1 Detailed Description . . . . .	186
14.113 InternalZGY::ZgyInternalWriterArgs Class Reference . . . . .	186
14.113.1 Detailed Description . . . . .	187
14.114 OpenZGY::Errors::ZgyIoError Class Reference . . . . .	188
14.114.1 Detailed Description . . . . .	188
14.115 OpenZGY::Impl::ZgyMeta Class Reference . . . . .	188
14.115.1 Detailed Description . . . . .	190
14.116 OpenZGY::Impl::ZgyMetaAndTools Class Reference . . . . .	190
14.116.1 Detailed Description . . . . .	191
14.116.2 Member Function Documentation . . . . .	191
14.116.2.1 transform() . . . . .	191
14.116.2.2 transform1() . . . . .	192
14.117 OpenZGY::Errors::ZgyMissingFeature Class Reference . . . . .	192
14.117.1 Detailed Description . . . . .	193
14.118 OpenZGY::Impl::ZgyReader Class Reference . . . . .	193
14.118.1 Detailed Description . . . . .	194
14.118.2 Constructor & Destructor Documentation . . . . .	194
14.118.2.1 ZgyReader() . . . . .	194
14.118.3 Member Function Documentation . . . . .	194
14.118.3.1 close() . . . . .	194
14.118.3.2 read() [1/3] . . . . .	194
14.118.3.3 read() [2/3] . . . . .	195
14.118.3.4 read() [3/3] . . . . .	195
14.118.3.5 readconst() . . . . .	195
14.119 OpenZGY::Errors::ZgySegmentIsClosed Class Reference . . . . .	196
14.119.1 Detailed Description . . . . .	196

# Alpha version. Expect changes.

xiv

14.120 OpenZGY::Errors::ZgyUserError Class Reference . . . . .	196
14.120.1 Detailed Description . . . . .	197
14.121 OpenZGY::Impl::ZgyUtils Class Reference . . . . .	197
14.121.1 Detailed Description . . . . .	197
14.121.2 Constructor & Destructor Documentation . . . . .	197
14.121.2.1 ZgyUtils() . . . . .	197
14.121.3 Member Function Documentation . . . . .	198
14.121.3.1 deletefile() . . . . .	198
14.122 OpenZGY::Impl::ZgyWriter Class Reference . . . . .	198
14.122.1 Detailed Description . . . . .	199
14.122.2 Constructor & Destructor Documentation . . . . .	199
14.122.2.1 ZgyWriter() . . . . .	199
14.122.2.2 ~ZgyWriter() . . . . .	200
14.122.3 Member Function Documentation . . . . .	200
14.122.3.1 close() . . . . .	200
14.122.3.2 close_incomplete() . . . . .	200
14.122.3.3 errorflag() . . . . .	201
14.122.3.4 finalize() . . . . .	201
14.122.3.5 set_errorflag() . . . . .	201
14.122.3.6 write() [1/3] . . . . .	202
14.122.3.7 write() [2/3] . . . . .	202
14.122.3.8 write() [3/3] . . . . .	202
14.122.3.9 writeconst() [1/3] . . . . .	203
14.122.3.10 writeconst() [2/3] . . . . .	203
14.122.3.11 writeconst() [3/3] . . . . .	203
14.123 OpenZGY::ZgyWriterArgs Class Reference . . . . .	204
14.123.1 Detailed Description . . . . .	205
14.123.2 Member Function Documentation . . . . .	205
14.123.2.1 bricksizes() . . . . .	205
14.123.2.2 compressor() . . . . .	206
14.123.2.3 corners() . . . . .	206
14.123.2.4 datarange() . . . . .	206
14.123.2.5 filename() . . . . .	206
14.123.2.6 hunit() . . . . .	206
14.123.2.7 ilinc() . . . . .	207
14.123.2.8 ilstart() . . . . .	207
14.123.2.9 iocontext() . . . . .	207
14.123.2.10 lodcompressor() . . . . .	207
14.123.2.11 metafrom() . . . . .	208
14.123.2.12 size() . . . . .	208
14.123.2.13 xlinc() . . . . .	208
14.123.2.14 xlstart() . . . . .	208



14.123.2.15 zfp_compressor()	209
14.123.2.16 zfp_lodcompressor()	209
14.123.2.17 zinc()	209
14.123.2.18 zstart()	209
14.123.2.19 zunit()	209
<b>15 File Documentation</b>	<b>211</b>
15.1 api.cpp File Reference	211
15.1.1 Detailed Description	212
15.2 api.h File Reference	212
15.2.1 Detailed Description	214
15.3 example.h File Reference	215
15.3.1 Detailed Description	216
15.4 exception.h File Reference	216
15.4.1 Detailed Description	217
15.5 impl/arrayops.h File Reference	217
15.5.1 Detailed Description	217
15.6 impl/bulk.h File Reference	218
15.6.1 Detailed Description	218
15.7 impl/compress_null.cpp File Reference	218
15.7.1 Detailed Description	219
15.8 impl/compression.cpp File Reference	219
15.8.1 Detailed Description	219
15.9 impl/compression.h File Reference	219
15.9.1 Detailed Description	220
15.10 impl/cornerpoints.h File Reference	220
15.10.1 Detailed Description	220
15.11 impl/databuffer.h File Reference	220
15.11.1 Detailed Description	221
15.12 impl/enum.h File Reference	221
15.12.1 Detailed Description	222
15.13 impl/environment.h File Reference	222
15.13.1 Detailed Description	222
15.14 impl/file.h File Reference	222
15.14.1 Detailed Description	223
15.15 impl/file_local.cpp File Reference	224
15.15.1 Detailed Description	224
15.16 impl/file_sd.h File Reference	224
15.16.1 Detailed Description	225
15.17 impl/genlod.h File Reference	225
15.17.1 Detailed Description	225
15.18 impl/guid.h File Reference	226

# Alpha version. Expect changes.

xvi

15.18.1 Detailed Description . . . . .	226
15.19 impl/histogrambuilder.h File Reference . . . . .	226
15.19.1 Detailed Description . . . . .	227
15.20 impl/histogramdata.h File Reference . . . . .	227
15.20.1 Detailed Description . . . . .	227
15.21 impl/iltf2d.h File Reference . . . . .	227
15.21.1 Detailed Description . . . . .	228
15.22 impl/lodalgo.h File Reference . . . . .	228
15.22.1 Detailed Description . . . . .	229
15.23 impl/lodsampling.h File Reference . . . . .	229
15.23.1 Detailed Description . . . . .	229
15.24 impl/logger.h File Reference . . . . .	229
15.24.1 Detailed Description . . . . .	230
15.25 impl/lookuptable.h File Reference . . . . .	230
15.25.1 Detailed Description . . . . .	231
15.26 impl/meta.h File Reference . . . . .	231
15.26.1 Detailed Description . . . . .	232
15.27 impl/roundandclip.h File Reference . . . . .	233
15.27.1 Detailed Description . . . . .	233
15.28 impl/statisticdata.h File Reference . . . . .	233
15.28.1 Detailed Description . . . . .	234
15.29 impl/structaccess.h File Reference . . . . .	234
15.29.1 Detailed Description . . . . .	235
15.30 impl/subtiling.cpp File Reference . . . . .	235
15.30.1 Detailed Description . . . . .	235
15.31 impl/subtiling.h File Reference . . . . .	236
15.31.1 Detailed Description . . . . .	237
15.32 impl/timer.h File Reference . . . . .	237
15.32.1 Detailed Description . . . . .	237
15.33 impl/transform.h File Reference . . . . .	238
15.33.1 Detailed Description . . . . .	238
15.34 impl/types.h File Reference . . . . .	238
15.34.1 Detailed Description . . . . .	239
15.35 iocontext.h File Reference . . . . .	239
15.35.1 Detailed Description . . . . .	239

<b>Index</b>	<b>241</b>
--------------	------------

# Alpha version. Expect changes.

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

See also the following related pages:

- [Physical layout of ZGY files](#)
- [Implementation notes](#)
- [Algorithms for decimation](#)
- [migration](#)



# Alpha version. Expect changes.

## 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);
    }
}
```

# Alpha version. Expect changes.

Alpha version. Expect changes.

## **Chapter 3**

### **Physical layout of ZGY files**





Alpha version. Expect changes.

## Chapter 4

### Implementation notes



Alpha version. Expect changes.

## Chapter 5

### Algorithms for decimation



# Alpha version. Expect changes.

## Chapter 6

# Migration guide from ZGY-Public / ZGY-Cloud to OpenZGY

### ZGY-Public (C++)

The API is completely different. Since the core API (i.e. without the cloud extensions) is very small I don't think this will be a problem.

### ZGY-Public (Python)

The new API is very similar to the old one. Some stringly typed enums (zunitdim, hunitdim, datatype, sourcetype) have been changed to become real Python enums.

### ZGY-Cloud (C++ and Python)

The old ZGY accessor had a separate ZGY-Cloud module with its own "back door" api for cloud specific operations such as setting the credentials. The ZGY-Cloud module (both the C++ and Python API) have not been published but are used in Petrel and in the ML project.

The following list shows the replacements. Note that the basic create/read/write api is unchanged.

Unless otherwise noted, `zgy_someFunction()` in the old ZGY-Cloud module corresponds to `zgyccloud.someFunction()` in the old Python wrapper.

```
configure(backend, name, sdkey, sdurl)
setToken(backend, token, type, global)
setTokenCb(backend, callback, type) # C++ only
setSegmentSize(size_in_mb)
=> Information to be specified in the iocontext.
deleteFile(name, token)
=> pure python: openzgy.api.ZgyUtils.delete()
   or sdglue.SdUtil(*cred).delete(name)
=> wrapper: openzgycpp.ZgyUtils.delete
=> native: ZgyUtils.deleteFile (TODO-Low test?)
enableRealCache(on, size_in_mb)
setCacheHooks(hooks) # C++ only
getCacheHooks() # C++ only
=> Not supported by OpenZGY.
getTestToken(backend)
=> pass "FILE:carbon.slbapp.com" as the IOContext token.
lastError()
resetError()
```

# Alpha version. Expect changes.

```
=> N/A because all errors are reported synchronously.
listObjects(name, recursive, token)
listBuckets(project) # C++ only
listProject() # C++ only
=> Not supported, and never was officially.
setLogger(callback, level)
getCacheStats(...) # C++ only
=> Debugging and logging is somewhat different.
getLegalTag(name, token) # C++: zgy_{get,set}Tag
setLegalTag(name, token, legaltag)
getMetaData(name, token)
setMetaData(name, token, metadata)
getSeismicMeta(name, token) # C++: zgy_{get,set}SeisMeta
setSeismicMeta(name, token, seismicmeta)
=> TODO-Medium investigate further.
    legaltag, seismicmeta, writeid settable in iocontext
    but cannot(?) be modified later.
    metadata might be missing
getContext(backend, name, global)
setContext(backend, name, value, global)
=> TODO-Medium investigate further.
=> This is a general back door mechanism, used for ...
=> and replaced by ...
zgy_checkAccess(name, token) # C++ only
=> TODO-Low might need to support this.
```

# Alpha version. Expect changes.

## Chapter 7

## Module Index

### 7.1 Modules

Here is a list of all modules:

List of exceptions . . . . .	<a href="#">27</a>
------------------------------	--------------------





# Alpha version. Expect changes.

## Chapter 8

## Namespace Index

### 8.1 Namespace List

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

<a href="#">InternalZGY</a>	
Implementation not visible to clients . . . . .	29
<a href="#">OpenZGY</a>	
The entire public API is in this namespace . . . . .	42
<a href="#">OpenZGY::Errors</a>	
Exceptions that can be thrown by <a href="#">OpenZGY</a> . . . . .	43
<a href="#">OpenZGY::Formatters</a>	
Operator<< for readable output of enums etc . . . . .	43
<a href="#">OpenZGY::Impl</a>	
Implementation of the abstract interfaces in <a href="#">OpenZGY</a> . . . . .	44



## Chapter 9

### Hierarchical Index

#### 9.1 Class Hierarchy

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

InternalZGY::CompressFactoryImpl . . . . .	45
InternalZGY::Config . . . . .	46
InternalZGY::FileConfig . . . . .	68
InternalZGY::DataBuffer . . . . .	47
InternalZGY::DataBufferNd< T, NDim > . . . . .	52
InternalZGY::OrderedCornerPoints::Element . . . . .	58
OpenZGY::Impl::EnumMapper . . . . .	60
InternalZGY::Environment . . . . .	62
InternalZGY::ZgyInternalMeta::ErrorsWillCorruptFile . . . . .	63
InternalZGY::ZgyInternalBulk::ErrorsWillCorruptFile . . . . .	63
std::exception	
std::runtime_error	
OpenZGY::Errors::ZgyError . . . . .	181
OpenZGY::Errors::ZgyAborted . . . . .	179
OpenZGY::Errors::ZgyCorruptedFile . . . . .	179
OpenZGY::Errors::ZgyEndOfFile . . . . .	180
OpenZGY::Errors::ZgyFormatError . . . . .	182
OpenZGY::Errors::ZgyInternalError . . . . .	185
OpenZGY::Errors::ZgyIoError . . . . .	188
OpenZGY::Errors::ZgyMissingFeature . . . . .	192
OpenZGY::Errors::ZgySegmentIsClosed . . . . .	196
OpenZGY::Errors::ZgyUserError . . . . .	196
InternalZGY::FileADT . . . . .	64
InternalZGY::FileCommon . . . . .	67
InternalZGY::LocalFileLinux . . . . .	124
InternalZGY::FileUtilsSeismicStore . . . . .	71
InternalZGY::FileFactory . . . . .	69
InternalZGY::FileHeaderPOD . . . . .	70
InternalZGY::GenLodBase . . . . .	71
InternalZGY::GenLodImpl . . . . .	76
InternalZGY::GenLodC . . . . .	74
InternalZGY::GUID . . . . .	80
InternalZGY::HeaderAccessFactory . . . . .	81
InternalZGY::HistHeaderV1POD . . . . .	83

InternalZGY::HistHeaderV2POD	84
InternalZGY::HistogramBuilder	84
InternalZGY::HistogramData	88
InternalZGY::IHeaderAccess	92
InternalZGY::IFileHeaderAccess	92
InternalZGY::FileHeaderAccess	70
InternalZGY::IHistHeaderAccess	93
InternalZGY::HistHeaderAccess	82
InternalZGY::HistHeaderV1Access	82
InternalZGY::HistHeaderV2Access	83
InternalZGY::IInfoHeaderAccess	94
InternalZGY::InfoHeaderAccess	102
InternalZGY::InfoHeaderV1Access	103
InternalZGY::InfoHeaderV2Access	105
InternalZGY::ILookupTableAccess	95
InternalZGY::LookupTableV0Access	141
InternalZGY::IOffsetHeaderAccess	110
InternalZGY::OffsetHeaderAccess	147
InternalZGY::OffsetHeaderV1Access	147
InternalZGY::OffsetHeaderV2Access	148
InternalZGY::IJK	95
InternalZGY::SummaryTimer::Impl	96
InternalZGY::ImplicitLinearTransform2d	96
InternalZGY::ImplicitLinearTransform2dImp	101
InternalZGY::InfoHeaderV1POD	105
InternalZGY::InfoHeaderV2POD	108
OpenZGY::IOContext	109
OpenZGY::SeismicStoreIOContext	164
OpenZGY::IZgyMeta	111
OpenZGY::Impl::ZgyMeta	188
OpenZGY::Impl::ZgyMetaAndTools	190
OpenZGY::Impl::ZgyReader	193
OpenZGY::Impl::ZgyWriter	198
OpenZGY::IZgyTools	115
OpenZGY::Impl::ZgyMetaAndTools	190
OpenZGY::IZgyReader	112
OpenZGY::Impl::ZgyReader	193
OpenZGY::IZgyWriter	119
OpenZGY::Impl::ZgyWriter	198
OpenZGY::IZgyUtils	117
OpenZGY::Impl::ZgyUtils	197
InternalZGY::LodAllZero< T >	127
InternalZGY::LodAverage< T >	127
InternalZGY::LodAverage< float >	128
InternalZGY::LodAverageNon0< T >	128
InternalZGY::LodAverageNon0< float >	129
InternalZGY::LodDecimate< T >	129
InternalZGY::LodDecimateSkipNaN< T >	130
InternalZGY::LodDecimateSkipNaN< float >	130
InternalZGY::LodMaximum< T >	131
InternalZGY::LodMedian< T >	131
InternalZGY::LodMedian< float >	132
InternalZGY::LodMinimum< T >	132
InternalZGY::LodMinMax< T >	133
InternalZGY::LodMostFrequent< T, SkipNaN, SkipZero >	133
InternalZGY::LodSampling	134

# Alpha version. Expect changes.

InternalZGY::LodWeightedAverage< T > . . . . .	135
InternalZGY::LoggerBase . . . . .	137
InternalZGY::Logger . . . . .	136
InternalZGY::LookupTable . . . . .	139
InternalZGY::LookupTable::LutInfo . . . . .	142
InternalZGY::LutInfoEx . . . . .	143
InternalZGY::NullCompressPlugin . . . . .	144
InternalZGY::OffsetHeaderV1POD . . . . .	148
InternalZGY::OffsetHeaderV2POD . . . . .	150
InternalZGY::OrderedCornerPoints . . . . .	150
OpenZGY::ProgressWithDots . . . . .	154
InternalZGY::PushEnvironment . . . . .	156
InternalZGY::RawDataTypeDetails . . . . .	157
InternalZGY::RawDataTypeTraits< T > . . . . .	157
InternalZGY::RawDataTypeTraits< float > . . . . .	158
InternalZGY::RawDataTypeTraits< std::int16_t > . . . . .	158
InternalZGY::RawDataTypeTraits< std::int32_t > . . . . .	158
InternalZGY::RawDataTypeTraits< std::int8_t > . . . . .	159
InternalZGY::RawDataTypeTraits< std::uint16_t > . . . . .	159
InternalZGY::RawDataTypeTraits< std::uint32_t > . . . . .	159
InternalZGY::RawDataTypeTraits< std::uint8_t > . . . . .	159
InternalZGY::ReadRequest . . . . .	160
InternalZGY::NullCompressPlugin::Register . . . . .	161
OpenZGY::SampleHistogram . . . . .	162
OpenZGY::SampleStatistics . . . . .	163
InternalZGY::StatisticData . . . . .	169
InternalZGY::SummaryTimer . . . . .	173
InternalZGY::SummaryPrintingTimer . . . . .	172
InternalZGY::ImplicitLinearTransform2d::TiePoint . . . . .	175
InternalZGY::Timer . . . . .	176
InternalZGY::PrintingTimer . . . . .	154
InternalZGY::RawPrintingTimer . . . . .	160
InternalZGY::SimpleTimer . . . . .	168
InternalZGY::TmpLookupEntry . . . . .	178
InternalZGY::ZgyInternalBulk . . . . .	183
InternalZGY::ZgyInternalMeta . . . . .	185
InternalZGY::ZgyInternalWriterArgs . . . . .	186
OpenZGY::ZgyWriterArgs . . . . .	204



# Alpha version. Expect changes.

## Chapter 10

## Class Index

### 10.1 Class List

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

<a href="#">InternalZGY::CompressFactoryImpl</a>	
Registry of known compress and decompress algorithms	45
<a href="#">InternalZGY::Config</a>	46
<a href="#">InternalZGY::DataBuffer</a>	
Each <a href="#">DataBuffer</a> instance represents some in memory data	47
<a href="#">InternalZGY::DataBufferNd&lt; T, NDim &gt;</a>	52
<a href="#">InternalZGY::OrderedCornerPoints::Element</a>	58
<a href="#">OpenZGY::Impl::EnumMapper</a>	
Convert between internal and external data types	60
<a href="#">InternalZGY::Environment</a>	62
<a href="#">InternalZGY::ZgyInternalMeta::ErrorsWillCorruptFile</a>	63
<a href="#">InternalZGY::ZgyInternalBulk::ErrorsWillCorruptFile</a>	63
<a href="#">InternalZGY::FileADT</a>	64
<a href="#">InternalZGY::FileCommon</a>	
Implementation of some methods that might be shared	67
<a href="#">InternalZGY::FileConfig</a>	68
<a href="#">InternalZGY::FileFactory</a>	69
<a href="#">InternalZGY::FileHeaderAccess</a>	70
<a href="#">InternalZGY::FileHeaderPOD</a>	70
<a href="#">InternalZGY::FileUtilsSeismicStore</a>	71
<a href="#">InternalZGY::GenLodBase</a>	71
<a href="#">InternalZGY::GenLodC</a>	74
<a href="#">InternalZGY::GenLodImpl</a>	76
<a href="#">InternalZGY::GUID</a>	
Simplified GUID handling. Only big endian, random number guids	80
<a href="#">InternalZGY::HeaderAccessFactory</a>	81
<a href="#">InternalZGY::HistHeaderAccess</a>	82
<a href="#">InternalZGY::HistHeaderV1Access</a>	82
<a href="#">InternalZGY::HistHeaderV1POD</a>	
Physical layout of Histogram Header version 1	83
<a href="#">InternalZGY::HistHeaderV2Access</a>	83
<a href="#">InternalZGY::HistHeaderV2POD</a>	
Physical layout of Histogram Header version 2 and 3	84
<a href="#">InternalZGY::HistogramBuilder</a>	
Collect statistics and histogram for bulk data	84

<a href="#">InternalZGY::HistogramData</a>	
A histogram for a data set	88
<a href="#">InternalZGY::IFileHeaderAccess</a>	92
<a href="#">InternalZGY::IHeaderAccess</a>	92
<a href="#">InternalZGY::IHistHeaderAccess</a>	93
<a href="#">InternalZGY::IInfoHeaderAccess</a>	94
<a href="#">InternalZGY::IJK</a>	95
<a href="#">InternalZGY::ILookupTableAccess</a>	95
<a href="#">InternalZGY::SummaryTimer::Impl</a>	96
<a href="#">InternalZGY::ImplicitLinearTransform2d</a>	96
<a href="#">InternalZGY::ImplicitLinearTransform2dImp</a>	101
<a href="#">InternalZGY::InfoHeaderAccess</a>	102
<a href="#">InternalZGY::InfoHeaderV1Access</a>	103
<a href="#">InternalZGY::InfoHeaderV1POD</a>	
Physical layout of Info Header version 1	105
<a href="#">InternalZGY::InfoHeaderV2Access</a>	105
<a href="#">InternalZGY::InfoHeaderV2POD</a>	
Physical layout of Info Header version 2 and 3	108
<a href="#">OpenZGY::IOContext</a>	
Base class for backend specific context	109
<a href="#">InternalZGY::IOffsetHeaderAccess</a>	110
<a href="#">OpenZGY::IZgyMeta</a>	
Base class of <a href="#">IZgyReader</a> and <a href="#">IZgyWriter</a>	111
<a href="#">OpenZGY::IZgyReader</a>	
Main API for reading ZGY files	112
<a href="#">OpenZGY::IZgyTools</a>	
Base class of <a href="#">IZgyReader</a> and <a href="#">IZgyWriter</a>	115
<a href="#">OpenZGY::IZgyUtils</a>	
Operations other than read and write	117
<a href="#">OpenZGY::IZgyWriter</a>	
Main API for creating ZGY files	119
<a href="#">InternalZGY::LocalFileLinux</a>	124
<a href="#">InternalZGY::LodAllZero&lt; T &gt;</a>	
Set the low resolution data to all zeros	127
<a href="#">InternalZGY::LodAverage&lt; T &gt;</a>	
Arithmetic average of 8 neighboring integer samples	127
<a href="#">InternalZGY::LodAverage&lt; float &gt;</a>	
Arithmetic average of 8 neighboring float samples, ignoring NaN	128
<a href="#">InternalZGY::LodAverageNon0&lt; T &gt;</a>	
Arithmetic average, excluding zero, of integer data	128
<a href="#">InternalZGY::LodAverageNon0&lt; float &gt;</a>	
Arithmetic average, excluding NaN and zero, of float data	129
<a href="#">InternalZGY::LodDecimate&lt; T &gt;</a>	
Use just one of the 8 input values	129
<a href="#">InternalZGY::LodDecimateSkipNaN&lt; T &gt;</a>	
Use just one of the 8 integral input values	130
<a href="#">InternalZGY::LodDecimateSkipNaN&lt; float &gt;</a>	
Use just one of the 8 float input values, looking for one not NaN	130
<a href="#">InternalZGY::LodMaximum&lt; T &gt;</a>	
Maximum of 8 neighboring samples, excluding NaN	131
<a href="#">InternalZGY::LodMedian&lt; T &gt;</a>	
Median of 8 neighboring integer samples	131
<a href="#">InternalZGY::LodMedian&lt; float &gt;</a>	
Median of 8 neighboring float samples, ignoring NaN	132
<a href="#">InternalZGY::LodMinimum&lt; T &gt;</a>	
Minimum of 8 neighboring samples, excluding NaN	132
<a href="#">InternalZGY::LodMinMax&lt; T &gt;</a>	
Weird algorithm, probably not useful	133



# Alpha version. Expect changes.

<a href="#">InternalZGY::LodMostFrequent&lt; T, SkipNaN, SkipZero &gt;</a>	
Most frequent value . . . . .	133
<a href="#">InternalZGY::LodSampling</a>	
Static methods for downsampling. Used by lodalgo.cpp only . . . . .	134
<a href="#">InternalZGY::LodWeightedAverage&lt; T &gt;</a>	
Weighted arithmetic average of 8 neighboring samples . . . . .	135
<a href="#">InternalZGY::Logger</a> . . . . .	136
<a href="#">InternalZGY::LoggerBase</a> . . . . .	137
<a href="#">InternalZGY::LookupTable</a>	
Static methods to assist working with lookup tables . . . . .	139
<a href="#">InternalZGY::LookupTableV0Access</a> . . . . .	141
<a href="#">InternalZGY::LookupTable::LutInfo</a>	
Decoded contents of the lookup table for one brick or tile . . . . .	142
<a href="#">InternalZGY::LutInfoEx</a> . . . . .	143
<a href="#">InternalZGY::NullCompressPlugin</a>	
Example compression plug-in that always fails to compress . . . . .	144
<a href="#">InternalZGY::OffsetHeaderAccess</a> . . . . .	147
<a href="#">InternalZGY::OffsetHeaderV1Access</a> . . . . .	147
<a href="#">InternalZGY::OffsetHeaderV1POD</a>	
Physical layout of Offset Header version 1 . . . . .	148
<a href="#">InternalZGY::OffsetHeaderV2Access</a> . . . . .	148
<a href="#">InternalZGY::OffsetHeaderV2POD</a>	
Physical layout of Offset Header version 2 and 3 (empty) . . . . .	150
<a href="#">InternalZGY::OrderedCornerPoints</a> . . . . .	150
<a href="#">InternalZGY::PrintingTimer</a>	
Timer that prints its result when going out of scope . . . . .	154
<a href="#">OpenZGY::ProgressWithDots</a>	
Simple progress bar . . . . .	154
<a href="#">InternalZGY::PushEnvironment</a> . . . . .	156
<a href="#">InternalZGY::RawDataTypeDetails</a> . . . . .	157
<a href="#">InternalZGY::RawDataTypeTraits&lt; T &gt;</a> . . . . .	157
<a href="#">InternalZGY::RawDataTypeTraits&lt; float &gt;</a> . . . . .	158
<a href="#">InternalZGY::RawDataTypeTraits&lt; std::int16_t &gt;</a> . . . . .	158
<a href="#">InternalZGY::RawDataTypeTraits&lt; std::int32_t &gt;</a> . . . . .	158
<a href="#">InternalZGY::RawDataTypeTraits&lt; std::int8_t &gt;</a> . . . . .	159
<a href="#">InternalZGY::RawDataTypeTraits&lt; std::uint16_t &gt;</a> . . . . .	159
<a href="#">InternalZGY::RawDataTypeTraits&lt; std::uint32_t &gt;</a> . . . . .	159
<a href="#">InternalZGY::RawDataTypeTraits&lt; std::uint8_t &gt;</a> . . . . .	159
<a href="#">InternalZGY::RawPrintingTimer</a>	
Timer that prints its result when going out of scope . . . . .	160
<a href="#">InternalZGY::ReadRequest</a> . . . . .	160
<a href="#">InternalZGY::NullCompressPlugin::Register</a>	
Register the compress and decompress functions in the factory . . . . .	161
<a href="#">OpenZGY::SampleHistogram</a>	
Histogram of all sample values on the file . . . . .	162
<a href="#">OpenZGY::SampleStatistics</a>	
Statistics of all sample values on the file . . . . .	163
<a href="#">OpenZGY::SeismicStoreIOContext</a>	
Credentials and configuration for Seismic Store . . . . .	164
<a href="#">InternalZGY::SimpleTimer</a>	
Timer that knows where to store the result . . . . .	168
<a href="#">InternalZGY::StatisticData</a>	
Holds the result of computing statistics . . . . .	169
<a href="#">InternalZGY::SummaryPrintingTimer</a>	
SummaryTimer that prints its result when going out of scope . . . . .	172
<a href="#">InternalZGY::SummaryTimer</a>	
Hold the timing results from zero or more <a href="#">Timer</a> instances . . . . .	173
<a href="#">InternalZGY::ImplicitLinearTransform2d::TiePoint</a> . . . . .	175

<a href="#">InternalZGY::Timer</a>	176
<a href="#">InternalZGY::TmpLookupEntry</a>	178
<a href="#">OpenZGY::Errors::ZgyAborted</a>	
User aborted the computation	179
<a href="#">OpenZGY::Errors::ZgyCorruptedFile</a>	
The ZGY file became corrupted while writing to it	179
<a href="#">OpenZGY::Errors::ZgyEndOfFile</a>	
Trying to read past EOF	180
<a href="#">OpenZGY::Errors::ZgyError</a>	
Base class for all exceptions thrown by OpenZGY	181
<a href="#">OpenZGY::Errors::ZgyFormatError</a>	
Corrupted or unsupported ZGY file	182
<a href="#">InternalZGY::ZgyInternalBulk</a>	183
<a href="#">OpenZGY::Errors::ZgyInternalError</a>	
Exception that might be caused by a bug in OpenZGY	185
<a href="#">InternalZGY::ZgyInternalMeta</a>	185
<a href="#">InternalZGY::ZgyInternalWriterArgs</a>	
Internal counterpart to <a href="#">OpenZGY::ZgyWriterArgs</a>	186
<a href="#">OpenZGY::Errors::ZgyIoError</a>	
Exception from the I/O layer	188
<a href="#">OpenZGY::Impl::ZgyMeta</a>	
High level API for reading and writing ZGY files	188
<a href="#">OpenZGY::Impl::ZgyMetaAndTools</a>	
Add coordinate conversion to the concrete <a href="#">ZgyMeta</a> class	190
<a href="#">OpenZGY::Errors::ZgyMissingFeature</a>	
Missing feature	192
<a href="#">OpenZGY::Impl::ZgyReader</a>	
Concrete implementation of <a href="#">IZgyReader</a>	193
<a href="#">OpenZGY::Errors::ZgySegmentIsClosed</a>	
Exception used internally to request a retry	196
<a href="#">OpenZGY::Errors::ZgyUserError</a>	
Exception that might be caused by the calling application	196
<a href="#">OpenZGY::Impl::ZgyUtils</a>	
Concrete implementation of <a href="#">IZgyUtils</a>	197
<a href="#">OpenZGY::Impl::ZgyWriter</a>	
Concrete implementation of <a href="#">IZgyWriter</a>	198
<a href="#">OpenZGY::ZgyWriterArgs</a>	
Argument package for creating a ZGY file	204

# Alpha version. Expect changes.

## Chapter 11

## File Index

### 11.1 File List

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

<a href="#">/home/build/oz/doc/<b>doxygen.h</b></a>	??
<a href="#">api.cpp</a>	
Implements the pure interfaces of the API	211
<a href="#">api.h</a>	
IZgyReader, IZgyWriter, and other user visible classes	212
<b>declspec.h</b>	??
<a href="#">example.h</a>	
Example program using the C++ API	215
<a href="#">exception.h</a>	
Defines exceptions that may be raised by OpenZGY	216
<a href="#">iocontext.h</a>	
Backend specific context	239
<a href="#">impl/arrayops.h</a>	217
<a href="#">impl/bulk.h</a>	
Bulk data read/write	218
<a href="#">impl/compress_null.cpp</a>	
Example code for the compression plug-in mechanism	218
<a href="#">impl/compression.cpp</a>	
Compression factory	219
<a href="#">impl/compression.h</a>	
Factory for handling compression	219
<a href="#">impl/cornerpoints.h</a>	
Deprecated. See <a href="#">transform.h</a>	220
<a href="#">impl/databuffer.h</a>	
Each DataBuffer instance represents some in memory data	220
<a href="#">impl/enum.h</a>	
Enums and type aliases not visible to the public API	221
<a href="#">impl/environment.h</a>	
Reading and writing environment variables	222
<a href="#">impl/file.h</a>	
Low level I/O, abstract layer	222
<a href="#">impl/file_local.cpp</a>	
Low level I/O, regular files	224
<a href="#">impl/file_sd.h</a>	
Low level I/O utilities, Seismic Store	224

impl/ <a href="#">genlod.h</a>	
Generate low resolution bricks	225
impl/ <a href="#">guid.h</a>	
Simplified GUID handling. Only big endian, random number guides	226
impl/ <a href="#">histogrambuilder.h</a>	
Collect statistics and histogram for bulk data	226
impl/ <a href="#">histogramdata.h</a>	
Class <a href="#">InternalZGY::HistogramData</a>	227
impl/ <a href="#">iltf2d.h</a>	
Deprecated. See <a href="#">transform.h</a>	227
impl/ <a href="#">lodalgo.h</a>	
Decimation algorithms to output low resolution bricks	228
impl/ <a href="#">lodsampling.h</a>	
Static methods for downsampling. Used by lodalgo.cpp only	229
impl/ <a href="#">logger.h</a>	
Logging framework	229
impl/ <a href="#">lookuptable.h</a>	
Static methods to assist working with lookup tables	230
impl/ <a href="#">meta.h</a>	
Meta data read/write	231
impl/ <a href="#">roundandclip.h</a>	
Conversion between scalar types	233
impl/ <a href="#">statisticdata.h</a>	
Class <a href="#">InternalZGY::StatisticData</a>	233
impl/ <a href="#">structaccess.h</a>	
Low level tools for data conversion	234
impl/ <a href="#">subtiling.cpp</a>	
Handle 8x8 subtiling	235
impl/ <a href="#">subtiling.h</a>	
Handle 8x8 subtiling	236
impl/ <a href="#">timer.h</a>	
Easy to use performance measurement	237
impl/ <a href="#">transform.h</a>	
General coordinate conversion based on 3 control points	238
impl/ <a href="#">types.h</a>	
Types and enums only used internally	238

## Chapter 12

## Module Documentation

### 12.1 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.*

#### 12.1.1 Detailed Description



# Alpha version. Expect changes.

## Chapter 13

## Namespace Documentation

### 13.1 InternalZGY Namespace Reference

Implementation not visible to clients.

#### Classes

- class [CompressFactoryImpl](#)

*Registry of known compress and decompress algorithms.*

- class [Config](#)
- class [DataBuffer](#)

*Each [DataBuffer](#) instance represents some in memory data.*

- class [DataBufferNd](#)
- class [Environment](#)
- class [FileADT](#)
- class [FileCommon](#)

*Implementation of some methods that might be shared.*

- class [FileConfig](#)
- class [FileFactory](#)
- class [FileHeaderAccess](#)
- class [FileHeaderPOD](#)
- class [FileUtilsSeismicStore](#)
- class [GenLodBase](#)
- class [GenLodC](#)
- class [GenLodImpl](#)
- class [GUID](#)

*Simplified GUID handling. Only big endian, random number guids.*

- class [HeaderAccessFactory](#)
- class [HistHeaderAccess](#)
- class [HistHeaderV1Access](#)
- class [HistHeaderV1POD](#)

*Physical layout of Histogram Header version 1.*

- class [HistHeaderV2Access](#)
- class [HistHeaderV2POD](#)

*Physical layout of Histogram Header version 2 and 3.*

- class [HistogramBuilder](#)

*Collect statistics and histogram for bulk data.*

- class [HistogramData](#)

*A histogram for a data set.*

- class [IFileHeaderAccess](#)
- class [IHeaderAccess](#)
- class [IHistHeaderAccess](#)
- class [IInfoHeaderAccess](#)
- struct [IJK](#)
- class [ILookupTableAccess](#)
- class [ImplicitLinearTransform2d](#)
- class [ImplicitLinearTransform2dImp](#)
- class [InfoHeaderAccess](#)
- class [InfoHeaderV1Access](#)
- class [InfoHeaderV1POD](#)

*Physical layout of Info Header version 1.*

- class [InfoHeaderV2Access](#)
- class [InfoHeaderV2POD](#)

*Physical layout of Info Header version 2 and 3.*

- class [IOffsetHeaderAccess](#)
- class [LocalFileLinux](#)
- class [LodAllZero](#)

*Set the low resolution data to all zeros.*

- class [LodAverage](#)

*Arithmetic average of 8 neighboring integer samples.*

- class [LodAverage< float >](#)

*Arithmetic average of 8 neighboring float samples, ignoring NaN.*

- class [LodAverageNon0](#)

*Arithmetic average, excluding zero, of integer data.*

- class [LodAverageNon0< float >](#)

*Arithmetic average, excluding NaN and zero, of float data.*

- class [LodDecimate](#)

*Use just one of the 8 input values.*

- class [LodDecimateSkipNaN](#)

*Use just one of the 8 integral input values.*

- class [LodDecimateSkipNaN< float >](#)

*Use just one of the 8 float input values, looking for one not NaN.*

- class [LodMaximum](#)

*Maximum of 8 neighboring samples, excluding NaN.*

- class [LodMedian](#)

*Median of 8 neighboring integer samples.*

- class [LodMedian< float >](#)

*Median of 8 neighboring float samples, ignoring NaN.*

- class [LodMinimum](#)

*Minimum of 8 neighboring samples, excluding NaN.*

- class [LodMinMax](#)

*Weird algorithm, probably not useful.*

- class [LodMostFrequent](#)

*Most frequent value.*

- class [LodSampling](#)

*Static methods for downsampling. Used by lodalgo.cpp only.*

- class [LodWeightedAverage](#)

*Weighted arithmetic average of 8 neighboring samples.*



- class [Logger](#)
- class [LoggerBase](#)
- class [LookupTable](#)
  - Static methods to assist working with lookup tables.*
- class [LookupTableV0Access](#)
- struct [LutInfoEx](#)
- class [NullCompressPlugin](#)
  - Example compression plug-in that always fails to compress.*
- class [OffsetHeaderAccess](#)
- class [OffsetHeaderV1Access](#)
- class [OffsetHeaderV1POD](#)
  - Physical layout of Offset Header version 1.*
- class [OffsetHeaderV2Access](#)
- class [OffsetHeaderV2POD](#)
  - Physical layout of Offset Header version 2 and 3 (empty).*
- class [OrderedCornerPoints](#)
- class [PrintingTimer](#)
  - Timer that prints its result when going out of scope.*
- class [PushEnvironment](#)
- class [RawDataTypeDetails](#)
- struct [RawDataTypeTraits](#)
- struct [RawDataTypeTraits< float >](#)
- struct [RawDataTypeTraits< std::int16\\_t >](#)
- struct [RawDataTypeTraits< std::int32\\_t >](#)
- struct [RawDataTypeTraits< std::int8\\_t >](#)
- struct [RawDataTypeTraits< std::uint16\\_t >](#)
- struct [RawDataTypeTraits< std::uint32\\_t >](#)
- struct [RawDataTypeTraits< std::uint8\\_t >](#)
- class [RawPrintingTimer](#)
  - Timer that prints its result when going out of scope.*
- class [ReadRequest](#)
- class [SimpleTimer](#)
  - Timer that knows where to store the result.*
- class [StatisticData](#)
  - Holds the result of computing statistics.*
- class [SummaryPrintingTimer](#)
  - SummaryTimer that prints its result when going out of scope.*
- class [SummaryTimer](#)
  - Hold the timing results from zero or more [Timer](#) instances.*
- class [Timer](#)
- struct [TmpLookupEntry](#)
- class [ZgyInternalBulk](#)
- class [ZgyInternalMeta](#)
- class [ZgyInternalWriterArgs](#)
  - Internal counterpart to [OpenZGY::ZgyWriterArgs](#).*

### Typedefs

- typedef std::array< std::int64\_t, 3 > [index3\\_t](#)
  - type equivalent to [std::int64\\_t\[3\]](#)*
- typedef std::pair< std::shared\_ptr< const void >, std::int64\_t > [rawdata\\_t](#)
  - Shared data plus size. No other information.*
- typedef std::function< [rawdata\\_t](#)(const [rawdata\\_t](#) &, const [index3\\_t](#) &)> [compressor\\_t](#)
  - Function for compressing a brick.*
- typedef std::vector< [ReadRequest](#) > **ReadList**
- typedef std::vector< [ReadList](#) > **ReadDoubleList**

## Enumerations

- enum [RawDataType](#) {  
SignedInt8 = 0, UnsignedInt8 = 1, SignedInt16 = 2, UnsignedInt16 = 3,  
SignedInt32 = 4, UnsignedInt32 = 5, Float32 = 6, lbmFloat32 = 7 }
- enum [RawCoordType](#) {  
Unknown = 0, Meters = 1, Feet = 2, ArcSec = 3,  
ArcDeg = 4, ArcDegMinSec = 5 }
- enum [RawHorizontalDimension](#) { Unknown = 0, Length = 1, ArcAngle = 2 }
- enum [RawVerticalDimension](#) { Unknown = 0, Depth = 1, SeismicTWT = 2, SeismicOWT = 3 }
- enum [RawGridDefinition](#) { Unknown = 0, Parametric = 1, ThreePoint = 2, FourPoint = 3 }
- enum [BrickStatus](#) { Missing = 0, Constant = 1, Normal = 2, Compressed = 3 }
- enum [UpdateMode](#) { Never = 0, Constant = 1, Always = 4, Pedantic = 5 }
- enum [OpenMode](#) { Closed = 0, ReadOnly, ReadWrite, Truncate }
- enum [UsageHint](#) {  
Unknown = 0x00, TextFile = 0x01, Header = 0x10, Data = 0x20,  
Compressed = 0x40, Mixed = 0x40 }
- enum [LodAlgorithm](#) {  
[LodAlgorithm::LowPass](#) = 0, [LodAlgorithm::WeightedAverage](#), [LodAlgorithm::Average](#), [LodAlgorithm::Median](#),  
[LodAlgorithm::Minimum](#), [LodAlgorithm::Maximum](#), [LodAlgorithm::MinMax](#), [LodAlgorithm::Decimate](#),  
[LodAlgorithm::DecimateSkipNaN](#), [LodAlgorithm::DecimateRandom](#), [LodAlgorithm::AllZero](#), [LodAlgorithm::WhiteNoise](#),  
[LodAlgorithm::MostFrequent](#), [LodAlgorithm::MostFrequentNon0](#), [LodAlgorithm::AverageNon0](#) }

## Functions

- template<typename T, typename F >  
void [createGenericLevelOfDetail](#) (T \*dst, const std::array< std::int64\_t, 3 > &dsizes, const std::array< std::int64\_t, 3 > &dstrides, const T \*src, const std::array< std::int64\_t, 3 > &ssizes, const std::array< std::int64\_t, 3 > &sstrides, F function)
- template<typename T >  
void [createGenericLevelOfDetailLoPass](#) (T \*dst, const std::array< std::int64\_t, 3 > &dsizes, const std::array< std::int64\_t, 3 > &dstrides, const T \*src, const std::array< std::int64\_t, 3 > &ssizes, const std::array< std::int64\_t, 3 > &sstrides)
- Lowpass decimation vertically, simple decimation horizontally.*
- template<typename T >  
void [createLevelOfDetail](#) (T \*dst, const std::array< std::int64\_t, 3 > &dsizes, const std::array< std::int64\_t, 3 > &dstrides, const T \*src, const std::array< std::int64\_t, 3 > &ssizes, const std::array< std::int64\_t, 3 > &sstrides, [LodAlgorithm](#) algorithm, const std::int64\_t \*hist, int bins, double minHist, double maxHist)
- template<typename T >  
void [createLodT](#) (const std::shared\_ptr< [DataBuffer](#) > &result, const std::shared\_ptr< const [DataBuffer](#) > &input, [LodAlgorithm](#) algorithm, const std::int64\_t \*hist, std::int32\_t bincount, double histogram\_min, double histogram\_max)
- void [createLodST](#) (const std::shared\_ptr< [DataBuffer](#) > &result, const std::shared\_ptr< const [DataBuffer](#) > &input, [LodAlgorithm](#) algorithm, const std::int64\_t \*hist, std::int32\_t bincount, double histogram\_min, double histogram\_max)
- Main entry point for low resolution compute.*
- void [createLodPart](#) (const std::shared\_ptr< [DataBuffer](#) > &result, const std::shared\_ptr< const [DataBuffer](#) > &input, [LodAlgorithm](#) algorithm, const std::int64\_t \*hist, std::int32\_t bincount, double histogram\_min, double histogram\_max, int slice\_dim, std::int64\_t slice\_beg, std::int64\_t slice\_count)
- void [createLodMT](#) (const std::shared\_ptr< [DataBuffer](#) > &result, const std::shared\_ptr< const [DataBuffer](#) > &input, [LodAlgorithm](#) algorithm, const std::int64\_t \*hist, std::int32\_t bincount, double histogram\_min, double histogram\_max)
- Main entry point for low resolution compute.*
- void [createLod](#) (const std::shared\_ptr< [DataBuffer](#) > &result, const std::shared\_ptr< const [DataBuffer](#) > &input, [LodAlgorithm](#) algorithm, const std::int64\_t \*hist, std::int32\_t bincount, double histogram\_min, double histogram\_max)

*Main entry point for low resolution compute.*

- OPENZGY\_API bool **newlogger** (int priority, const std::string &str)
- OPENZGY\_API bool **newlogger** (int priority, const std::ios &ss)
- template<typename T >  
bool **IsFiniteT** (T)
- template<> bool **IsFiniteT**< float > (float value)
- template<> bool **IsFiniteT**< double > (double value)
- template<> bool **IsFiniteT**< long double > (long double value)
- template<typename T >  
bool **IsNanT** (T)
- template<> bool **IsNanT**< float > (float value)
- template<> bool **IsNanT**< double > (double value)
- template<> bool **IsNanT**< long double > (long double value)
- template<typename T >  
T **RoundAndClip** (double value)
- template<typename T >  
T **RoundAndClip** (double value, T nan)
- template<typename T, std::size\_t N>  
std::array< T, N > **ptr\_to\_array** (const T \*in)
- template<typename T >  
T **align** (const T &in)
- template<typename T, std::size\_t N>  
std::string **array\_to\_string** (const std::array< T, N > &a)
- template<typename T, std::size\_t N>  
std::string **array\_to\_hex** (const std::array< T, N > &a)
- template<typename T, std::size\_t N>  
std::string **ptr\_to\_string** (const T \*a)
- template<typename T, std::size\_t N>  
std::string **ptr\_to\_hex** (const T \*a)
- OPENZGY\_TEST\_API void **byteswapV1Long** (std::int64\_t \*ptr, size\_t n=1)
- OPENZGY\_TEST\_API void **byteswapV1Long** (std::uint64\_t \*ptr, size\_t n=1)
- template<typename T, typename U, int N>  
std::array< T, N > **array\_cast** (const std::array< U, N > &in)
- void **subtiling** (const std::array< std::int64\_t, 3 > &bricksizes, std::int64\_t itemsize, void \*dst, const void \*src, bool remove)
- bool **generalTransform** (double AX0, double AY0, double AX1, double AY1, double AX2, double AY2, double BX0, double BY0, double BX1, double BY1, double BX2, double BY2, double \*X, double \*Y, std::size\_t length)

*General coordinate conversion based on 3 control points.*

### 13.1.1 Detailed Description

Implementation not visible to clients.

•

### 13.1.2 Typedef Documentation

### 13.1.2.1 compressor\_t

```
typedef std::function<rawdata_t(const rawdata_t&, const index3_t&)> InternalZGY::compressor_t
```

Function for compressing a brick.

The 3d extent of the data to be compressed is passed as a hint. The total size in the rawdata\_t parameter is in bytes, while the size hint is in elements.

See also decompressor\_t and compfactory\_t. Currently those are only used inside class [CompressFactoryImpl](#), so I might as well keep them private.

### 13.1.2.2 index3\_t

```
typedef std::array<std::int64_t, 3> InternalZGY::index3_t
```

type equivalent to std::int64\_t[3]

The implementation is a std::array.

Note: I have considered crating a new type instead of just an alias, as this would make it easier to add arithmetic and output formatters. But I really want to avoid the dependencies this would cause. See [arrayops.h](#) for some operations that work even without a separate type.

### 13.1.2.3 rawdata\_t

```
typedef std::pair<std::shared_ptr<const void>, std::int64_t> InternalZGY::rawdata_t
```

Shared data plus size. No other information.

Used to describe both inputs and outputs for the compress and decompress algorithms. Both algorithms might also need the 3d extent of the input (compress) or expected output (decompress). That information will be provided separately. The compression plug-in might choose to ignore the hints.

## 13.1.3 Enumeration Type Documentation

### 13.1.3.1 BrickStatus

```
enum InternalZGY::BrickStatus [strong]
```

Brick status as used in the internal API only.

### 13.1.3.2 LodAlgorithm

```
enum InternalZGY::LodAlgorithm [strong]
```

Possible algorithms to generate LOD bricks. This is work in progress, and not all these algorithms will get implemented. The ones that are not needed will eventually get trimmed from this enum.

### 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. NOT IMPLEMENTED.
AllZero	Just fill the LOD brick with zeroes.
WhiteNoise	Fill with white noise. NOT IMPLEMENTED.
MostFrequent	The value that occurs most frequently.
MostFrequentNon0	The non-zero value that occurs most frequently.
AverageNon0	Average value, but treat 0 as NaN.

#### 13.1.3.3 RawCoordType

```
enum InternalZGY::RawCoordType [strong]
```

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

#### 13.1.3.4 RawDataType

```
enum InternalZGY::RawDataType [strong]
```

Sample data type as stored on the file. DO NOT CHANGE except possibly adding more codes at end. In the public API this maps to SampleDataType. Source: BrickedFileVersion.cpp, MetaDataValue.h This enum is used for all versions. Note that the existing public ZGY library only recognizes SignedInt8, SignedInt16, and Float32.

#### 13.1.3.5 RawGridDefinition

```
enum InternalZGY::RawGridDefinition [strong]
```

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

#### 13.1.3.6 RawHorizontalDimension

```
enum InternalZGY::RawHorizontalDimension [strong]
```

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

### 13.1.3.7 RawVerticalDimension

```
enum InternalZGY::RawVerticalDimension [strong]
```

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

### 13.1.3.8 UpdateMode

```
enum InternalZGY::UpdateMode [strong]
```

WORK IN PROGRESS, potential configurable behavior.

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

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

Kinds of leakage:

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

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

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

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

## 13.1.4 Function Documentation

### 13.1.4.1 align()

```
template<typename T >  
T InternalZGY::align (  
    const T & in )
```

Return the input scalar in such a way that &in is allowed to be misaligned with respect to T. Note that I am not sure this is good enough. A simple assignment instead of the memcpy will trigger undefined behavior. A decent compiler should optimize away the memcpy but also make sure that it doesn't use e.g. not try to use e.g. SSE instructions that require alignment. Worry: By defining the input argument as T&, will the compiler already at that point make up its mind that in must be aligned? <https://pzemtsov.github.io/2016/11/06/bug-story-alignment-on-x86.html>

### 13.1.4.2 array\_to\_hex()

```
template<typename T , std::size_t N>
std::string InternalZGY::array_to_hex (
    const std::array< T, N > & a )
```

For debugging, format a `std::array` of a numeric type into a hex string.

### 13.1.4.3 array\_to\_string()

```
template<typename T , std::size_t N>
std::string InternalZGY::array_to_string (
    const std::array< T, N > & a )
```

For debugging, works as `std::to_string` but takes care to also output `std::int8_t` and `std::uint8_t` as numbers, even though those types are probably typedef'd to signed/unsigned char.

### 13.1.4.4 byteswapV1Long()

```
void InternalZGY::byteswapV1Long (
    std::int64_t * ptr,
    size_t n = 1 )
```

Convert a 64-bit integer stored on disk as `vcs_uint64` to the in-memory representation or back. Class `vcs_uint64` consisted of two `uint32_type` members representing the high and low (in that order) 32-bits of a 64-bit unsigned integer. The two halves were stored little-endian on disk.

This method includes conversion between big and little endian if needed, byte swapping each of the 32-bit halves separately. Caveat, that functionality has not been properly tested.

The format was used by:

- VCS (compressed ZGY) version 1 - 13
- VBS (uncompressed ZGY) version 1
- VCO (compressed object file) version 1

Currently only VBS 1 is a concern for this code. There are 64 bit ints in the alpha- and brick lookup tables and in the offset headers.

### 13.1.4.5 createGenericLevelOfDetail()

```
template<typename T , typename F >
void InternalZGY::createGenericLevelOfDetail (
    T * dst,
    const std::array< std::int64_t, 3 > & dsize,
    const std::array< std::int64_t, 3 > & dstride,
    const T * src,
    const std::array< std::int64_t, 3 > & ssize,
    const std::array< std::int64_t, 3 > & sstride,
    F function )
```

Generic LOD calculation that can be used for all algorithms where the resulting sample depends only on the 8 surrounding samples in the source brick. I.e. currently everything except the lowpass algorithm that needs a window of size 10. The actual algorithm to be used is passed as a functor. If any calculation needs to be done up front, that can be handled by the functor's constructor.

### 13.1.4.6 createGenericLevelOfDetailLoPass()

```
template<typename T >
void InternalZGY::createGenericLevelOfDetailLoPass (
    T * dst,
    const std::array< std::int64_t, 3 > & dsize,
    const std::array< std::int64_t, 3 > & dstride,
    const T * src,
    const std::array< std::int64_t, 3 > & ssize,
    const std::array< std::int64_t, 3 > & sstride )
```

Lowpass decimation vertically, simple decimation horizontally.

Unlike the other algorithms this one needs full traces as input. Or at least prefers full traces, to avoid brick artifacts vertically.

### 13.1.4.7 createLevelOfDetail()

```
template<typename T >
void InternalZGY::createLevelOfDetail (
    T * dst,
    const std::array< std::int64_t, 3 > & dsize,
    const std::array< std::int64_t, 3 > & dstride,
    const T * src,
    const std::array< std::int64_t, 3 > & ssize,
    const std::array< std::int64_t, 3 > & sstride,
    LodAlgorithm algorithm,
    const std::int64_t * hist,
    int bins,
    double minHist,
    double maxHist )
```

Create a single LOD brick based on the data from one LOD level below. The valuetype of the source and target must be the same, but can be almost any scalar type.

Histogram information is passed in for use by some of the algorithms. The histogram should be in "storage" units. That only affects its limits though. TODO-Worry make sure these are set correctly. The old accessor assumed that for integral data the limits were simply the entire range.

When doing lowpass filtering of integral data, the actual calculation is done using doubles. We need to clip the result to the range of the data value type to prevent overflows, since the lowpass filter can give a result slightly outside the input range.

When doing lowpass filtering of float data there will be no clipping. Some samples might end up slightly outside the actual value range of the input data. Note behavior change: The old accessor did some clipping to avoid that case. Which just added to the confusion because the range it clipped to might be inaccurate.

### 13.1.4.8 createLod()

```
void OPENZGY_TEST_API InternalZGY::createLod (
    const std::shared_ptr< DataBuffer > & result,
    const std::shared_ptr< const DataBuffer > & input,
    LodAlgorithm algorithm,
    const std::int64_t * hist,
    std::int32_t bincount,
    double histogram_min,
    double histogram_max )
```

Main entry point for low resolution compute.

Create a single low resolution brick 1/8th the size of the input. Decides whether to enable multi-threading or not.



### 13.1.4.9 createLodMT()

```
void OPENZGY_TEST_API InternalZGY::createLodMT (
    const std::shared_ptr< DataBuffer > & result,
    const std::shared_ptr< const DataBuffer > & input,
    LodAlgorithm algorithm,
    const std::int64_t * hist,
    std::int32_t bincount,
    double histogram_min,
    double histogram_max )
```

Main entry point for low resolution compute.

Create a single low resolution brick 1/8th the size of the input. This is the multi-threaded version.

Caveat: If parallelizing is also done at a higher level then consider `omp_set_max_active_levels()`, especially if the higher level might only be able to use 2 or 4 threads.

One argument for parallelizing on a higher level instead is that we might then be able to read and compute at the same time. N/A for local file access because buffer cache prefetch can give us that automatically. I think.

TODO-Low fall back to single threaded if the input is small (e.g. less than 16 slices or less than  $64^3$  samples) or if the caller asks for cheaper algorithms. Low priority because those cases are (a) rare and (b) should run fast anyway, even if single thread *might* give a slight improvement.

Using a simple "copy" app between two local ssd files I measured the time for lod compute as 4% of finalize time or 2% of total time with 8 threads. Compared to 23% / 13% for a single thread. So the MT case had a 10% overall speedup on this test.

Switching to the much cheaper "Decimate" algorithm (just pick one of every 8 samples) then even in the single threaded case the algorithm only accounts for < 1% so there is not much to gain.

Algorithm(s) used for Onnia	Time
Decimate	8 sec
Average	37 sec
LowPass	72 sec
WeightedAverage	732 sec
Decimate,WeightedAverage	102 sec
LowPass,Decimate	64 sec
LowPass,WeightedAverage	159 sec

### 13.1.4.10 createLodPart()

```
void InternalZGY::createLodPart (
    const std::shared_ptr< DataBuffer > & result,
    const std::shared_ptr< const DataBuffer > & input,
    LodAlgorithm algorithm,
    const std::int64_t * hist,
    std::int32_t bincount,
    double histogram_min,
    double histogram_max,
```

```
int slice_dim,  
std::int64_t slice_beg,  
std::int64_t slice_count )
```

Run the low resolution compute on just a part of the provided data buffer. The part will be one of more slices in the slowest changing direction. If passing the last 3 parameters as "entire survey" then this is equivalent to calling [createLodST\(\)](#) directly.

#### 13.1.4.11 createLodST()

```
void OPENZGY_TEST_API InternalZGY::createLodST (  
    const std::shared_ptr< DataBuffer > & result,  
    const std::shared_ptr< const DataBuffer > & input,  
    LodAlgorithm algorithm,  
    const std::int64_t * hist,  
    std::int32_t bincount,  
    double histogram_min,  
    double histogram_max )
```

Main entry point for low resolution compute.

Create a single low resolution brick 1/8th the size of the input. This is the single threaded version.

#### 13.1.4.12 createLodT()

```
template<typename T >  
void InternalZGY::createLodT (  
    const std::shared_ptr< DataBuffer > & result,  
    const std::shared_ptr< const DataBuffer > & input,  
    LodAlgorithm algorithm,  
    const std::int64_t * hist,  
    std::int32_t bincount,  
    double histogram_min,  
    double histogram_max )
```

New version with "modernized" types. Not sure if it belongs in this file or higher up.

#### 13.1.4.13 IsFiniteT()

```
template<typename T >  
bool InternalZGY::IsFiniteT (  
    T )
```

IsFiniteT uses template magic to avoid having to test integral types.

#### 13.1.4.14 IsNanT()

```
template<typename T >  
bool InternalZGY::IsNanT (  
    T )
```

IsNanT uses template magic to avoid having to test integral types.

### 13.1.4.15 ptr\_to\_array()

```
template<typename T , std::size_t N>
std::array<T, N> InternalZGY::ptr_to_array (
    const T * in )
```

Convert a raw C pointer, possibly misaligned, to a safer std::array. Callers will need to specify both template parameters. In theory the compiler should be able to deduce T, but I haven't found a way to do this while still explicitly providing N.

### 13.1.4.16 ptr\_to\_hex()

```
template<typename T , std::size_t N>
std::string InternalZGY::ptr_to_hex (
    const T * a )
```

For debugging, format a raw pointer to a numeric type into a hex string. Caller needs to explicitly provide both template parameters.

### 13.1.4.17 ptr\_to\_string()

```
template<typename T , std::size_t N>
std::string InternalZGY::ptr_to_string (
    const T * a )
```

For debugging, format a raw pointer to a numeric type into a string. Caller needs to explicitly provide both template parameters.

### 13.1.4.18 RoundAndClip() [1/2]

```
template<typename T >
T InternalZGY::RoundAndClip (
    double value )
```

Cast a double to another type. If the target type is floating point, this is a simple cast. If the target type is integral the result is rounded to nearest and clipped to the target's value range. If you supply a second argument, NaN in the input is replaced with this value. plus or minus infinite are still clipped to the target range. If you do not supply a second argument, NaN input will result in an undefined return value. It probably ends up being 0x80000000 for signed/unsigned int and 0 for narrowed integral types, but the caller should not depend on this.

### 13.1.4.19 RoundAndClip() [2/2]

```
template<typename T >
T InternalZGY::RoundAndClip (
    double value,
    T nan )
```

Cast a double to another type, including substituting NaN with a value supplied by the caller if and only if the target is integral.

### 13.1.4.20 subtiling()

```
void InternalZGY::subtiling (
    const std::array< std::int64_t, 3 > & bricksizes,
    std::int64_t itemsize,
    void * dst,
    const void * src,
    bool remove )
```

Convert a single brick from subtiled to standard (if remove=true) or back.

## 13.2 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.*

### 13.2.1 Detailed Description

The entire public API is in this namespace.

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

## 13.3 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.*

### 13.3.1 Detailed Description

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

## 13.4 OpenZGY::Formatters Namespace Reference

operator<< for readable output of enums etc.

## Functions

- `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.*
- `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.*

### 13.4.1 Detailed Description

`operator<<` for readable output of enums etc.

## 13.5 OpenZGY::Impl Namespace Reference

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

## Classes

- class [EnumMapper](#)  
*convert between internal and external data types.*
- 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](#).*

### 13.5.1 Detailed Description

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

# Alpha version. Expect changes.

## Chapter 14

## Class Documentation

### 14.1 InternalZGY::CompressFactoryImpl Class Reference

Registry of known compress and decompress algorithms.

#### Public Types

- typedef std::function< [compressor\\_t](#)(const std::vector< std::string > &)> **compfactory\_t**
- typedef std::function< [rawdata\\_t](#)(const [rawdata\\_t](#) &, [BrickStatus](#), const [index3\\_t](#) &)> **decompressor\_t**

#### Static Public Member Functions

- static void [registerCompressor](#) (const std::string &name, const compfactory\_t &fn)  
*Register a function that will be called to create a compressor function.*
- static void [registerDecompressor](#) (const std::string &name, const decompressor\_t &fn)  
*Register a function that can decompress one or more types of compressed data.*
- static std::vector< std::string > [knownCompressors](#) ()
- static std::vector< std::string > [knownDecompressors](#) ()
- static [compressor\\_t](#) **getCompressor** (const std::string &name, const std::vector< std::string > &args)
- static [rawdata\\_t](#) **decompress** (const [rawdata\\_t](#) &cdata, [BrickStatus](#) status, const [index3\\_t](#) &shape)

#### 14.1.1 Detailed Description

Registry of known compress and decompress algorithms.

The compression and decompression algorithms are completely separate but we might as well handle both in the same class.

#### 14.1.2 Member Function Documentation

## 14.1.2.1 decompress()

```
rawdata_t InternalZGY::CompressFactoryImpl::decompress (
    const rawdata_t & cdata,
    BrickStatus status,
    const index3_t & shape ) [static]
```

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

## 14.1.2.2 knownCompressors()

```
std::vector< std::string > InternalZGY::CompressFactoryImpl::knownCompressors ( ) [static]
```

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

## 14.1.2.3 knownDecompressors()

```
std::vector< std::string > InternalZGY::CompressFactoryImpl::knownDecompressors ( ) [static]
```

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

## 14.1.2.4 registerDecompressor()

```
void InternalZGY::CompressFactoryImpl::registerDecompressor (
    const std::string & name,
    const decompressor_t & fn ) [static]
```

Register a function that can decompress one or more types of compressed data.

Pass an empty function in order to remove the registration.

The registered functions will be called in reverse order of registration until one of them indicates that it has decompressed the data. The supplied name is only for information, and technically doesn't even need to be unique. But it really should match the name of the compressor.

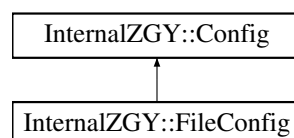
The documentation for this class was generated from the following files:

- [impl/compression.h](#)
- [impl/compression.cpp](#)

## 14.2 InternalZGY::Config Class Reference

```
#include <file.h>
```

Inheritance diagram for `InternalZGY::Config`:





### Public Member Functions

- virtual std::string **dump** () const =0

### Static Protected Member Functions

- static std::string **\_get\_string\_env** (const std::string &value\_from\_cf, const std::string &cfname, const std::string &envname, const std::string &dflt)
- static std::int64\_t **\_get\_numeric\_env** (std::int64\_t value\_from\_cf, const std::string &cfname, const std::string &envname, std::int64\_t dflt, std::int64\_t min\_value, std::int64\_t max\_value)
- static std::string **\_redact** (const std::string &)

### 14.2.1 Detailed Description

A concrete [Config](#) instance is specific to the back-end that uses it. It is normally populated from an IOContext and is not directly visible to application code. The jury is still out regarding whether class IOContext should be backend specific or just a stringly typed dictionary.

TODO-High: For now just remove [Config](#) completely and use the application visible IOContext. I can re-introduce {File,SD}[Config](#) later as a trivial mapping of the corresponding IOContext just to improve isolation.

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

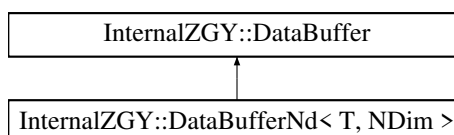
- impl/[file.h](#)
- impl/file.cpp

## 14.3 InternalZGY::DataBuffer Class Reference

Each [DataBuffer](#) instance represents some in memory data.

```
#include <databuffer.h>
```

Inheritance diagram for InternalZGY::DataBuffer:



### Public Types

- enum **flags\_t** { **C\_ORDERING** =1, **CONTIGUOUS** =2, **POSITIVE\_STRIDE** =4, **DEGENERATE** =8 }

## Public Member Functions

- **DataBuffer** (const [DataBuffer](#) &)=delete
- [DataBuffer](#) & **operator=** (const [DataBuffer](#) &)=delete
- virtual std::string **toString** () const =0
- virtual std::shared\_ptr< void > **voidData** ()=0
- virtual std::shared\_ptr< const void > **voidData** () const =0
- virtual bool **contiguous** () const =0
- virtual std::int64\_t **allocsize** () const =0
- virtual std::int64\_t **totalsize** () const =0
- virtual std::int64\_t **itemsiz** () const =0
- virtual const std::int64\_t \* **sizeptr** () const =0
- virtual const std::int64\_t \* **strideptr** () const =0
- virtual std::array< std::int64\_t, 3 > **size3d** () const =0
- virtual std::array< std::int64\_t, 3 > **stride3d** () const =0
- virtual bool **ownsdata** () const =0
- virtual bool **isScalar** () const =0
- virtual bool **isAllSame** (const std::int64\_t \*used\_in) const =0
- virtual double **scalarAsDouble** () const =0
- virtual [RawDataType](#) **datatype** () const =0
- virtual void **fill** (double value)=0
- virtual void **clear** ()=0
- virtual std::pair< double, double > **range** () const =0
- virtual void **copyFrom** (const [DataBuffer](#) \*src, const std::int64\_t \*srcorig, const std::int64\_t \*dstorig, const std::int64\_t \*cpyorig, const std::int64\_t \*cpysize)=0
- virtual std::shared\_ptr< [DataBuffer](#) > **clone** () const =0
- virtual std::shared\_ptr< [DataBuffer](#) > **scaleToFloat** (const std::array< double, 2 > &)=0
- virtual std::shared\_ptr< [DataBuffer](#) > **scaleToStorage** (const std::array< double, 2 > &, [RawDataType](#))=0
- virtual std::uint32\_t **layout** () const =0
- virtual bool **is\_cstride** () const =0
- virtual void **check\_cstride** () const =0
- virtual std::shared\_ptr< [DataBuffer](#) > **slice1** (int dim, std::int64\_t start, std::int64\_t size) const =0

## Static Public Member Functions

- static std::shared\_ptr< [DataBuffer](#) > **makeDataBuffer3d** (void \*raw, std::int64\_t nbytes, const std::array< std::int64\_t, 3 > &size, [RawDataType](#) dtype)

### 14.3.1 Detailed Description

Each [DataBuffer](#) instance represents some in memory data.

Warning, creeping features. The class starter out as a trivial pointer + 3d size struct but is growing dangerously close to emulating a Python `numpy.ndarray`.

This data type represents an unsafe buffer pointer, plus:

- (template parameter) value type
- (template parameter) number of dimensions
- Size in each dimension and implied total buffer size.
- Stride in each dimension to convert 1d/Nd.

# Alpha version. Expect changes.

- Option to flag the entire buffer as constant-value and store just the constant. The buffer pointer is then null. Note, I could also allocate a single T and set buffer to point to that. But this is just asking for buffer overrun bugs.

There is no knowledge of where in the survey this data belongs.

There is no support for views covering just part of a buffer, and strides must all be positive. It is possible to support those scenarios but let's not put in bells and whistles before we see they are needed.

[DataBuffer](#) is a non-templated base class to allow passing buffers around without needing to have the calling code templated as well.

[copyFrom\(\)](#) is somewhat unsafe. It can verify (using `dynamic_cast`) that "src" is of the same type as "this". But `srcorig` etc. are passed as dumb pointers so the code cannot know whether they are of the correct size. Maybe I am being too general here; maybe the number of dimensions should always be 3?

The actual bulk data is stored as a `std::shared_ptr<T>` which means it can be passed around independantly of the [DataBuffer](#) instance. The `voidData()` method returns the actual smart pointer as a `std::shared_ptr<void>` which can be downcast to the correct type. the `data()` method in the templated leaf type returns the raw pointer. It is possible to also have methods returning `void*` and `std::shared_ptr<T>` but don't add those unless actually used.

Constructors that expect an external buffer are required to provide it as a smart pointer. The [DataBuffer](#) will share ownership with the caller. There is nothing that prevents the callers from passing a `shared_ptr` with a no-op deleter, effectively removing reference counting. But that will of course void their warranty. And make the callers 100% responsible for the data being valid long enough.

Unsafe external buffers are currently used for `read()` and `write()` requests from the application code and for delivery from the file back end. Reference counted external buffers are used when the buffer was built from decompressed data.

Each of those scenarios should be ok since no buffers should be held onto after the functions return. TODO-Worry: Could async read requests get delivered after the call to `read()` got aborted via an exception? TODO-Worry: Might somebody decide to hold on to a buffer in order to implement delayed write? In 99% of cases the application's data buffer needs to be copied into a properly reference counted buffer anyway. Maybe make that 100% by not allowing short cuts when the application writes exactly one brick. Another problem is if the writer decides to copy out brick at a time into a reusable one brick buffer. This could also lead to the user's buffer being held on to longer. Maybe make sure all bricks are copied up front before any writing starts. This might also simplify parallelized compression.

As an alternative to the above, I have considered a scheme that makes less use of smart pointers so I can reduce the overhead they cause. Beware of premature optimization though. And more things that could go wrong with buffers freed early. The changes would be:

- Inside [DataBuffer](#) there will be an unsafe "`T *_data`" pointer instead of a smart pointer.
- If the [DataBuffer](#) owns its storage it will need code in the destructor to free it. No reference counting though.
- Callers that want to use an external buffer has no choice but to pass an unsafe pointer. Which means the caller is always responsible for keeping the data alive long enough. Note that I suspect this will happen anyway (see "voiding the warranty", above). So this might not be a major point.
- It will still be possible to obtain a smart pointer to the raw data if the [DataBuffer](#) instance itself is reference counted. Use the aliasing constructor of `std::shared_ptr`. Use `enable_shared_from_this`, or (preferably) change the signature of `voidData()` to provide the instance to alias. See `ZgyInternalBulk::_writeOneNormal`←Brick where the code needs a smart pointer in preparation to queue up a write.
- Beware that the `DataBuffer::_data` pointer must not be deleted (e.g. due to a `realloc`) before the instance itself goes out of scope. That rule applies in any case, but without reference counting we are talking about accessing garbage data instead of just stale data.

- Copy/assign would need a deep copy instead of aliasing the data buffer. However, shallow copying (e.g. to create a slice view) is not supported yet. So this is a future worry.

Other changes I have considered, orthogonal to the above:

- Move all data members to the parent. `_data` will be declared as `std::shared_ptr<void>`, and `data()` will need to do a cast. Which is safe with respect to alignment as long as the cast is to the concrete type that the buffer initially had. Several of the methods in the [DataBuffer](#) base class can now be made non virtual if desired.
- Have the destructor and/or `clear()` check the reference count of `_data` and do a deep copy if there are external references and if `_data` uses fake refcounts with an empty destructor. Unfortunately the latter is difficult to check for. Also, this fix only helps for asynchronous writes, not reads.
- TODO-Low: Remove the template on number of dimensions. This would make the API simpler. See e.g. `sizeptr()` vs. `size3d()`. Current usage in [OpenZGY](#) is always `NDim=3`. If the Alpha tile feature is resurrected then there will be a need for 2d arrays, but can probably be handled as 2d arrays with `nk=1`. Not quite as elegant as the current template but might still be a good idea overall.

TODO-Low decide on support for strided data.

It is easy enough for this class to allow arbitrary strides. Even strides that don't make sense. The problem is that code accessing the data would like to make assumptions about the stride so the code becomes simpler and also simpler to test. The current situation is unclear. Some code is flexible but it might not be possible to use that flexibility due to other code making assumptions.

- Fixed order `InlineCrosslineSlice` with no padding. Stride is simply `(nj*nk, nk, 1)`.
- Fixed order `InlineCrosslineSlice` with padding. As above, but the `nj` and `nk` numbers might be larger to account for unused space at the end in the `j` and `k` direction. Unused space in the slowest varying direction will not be detected. And is not usually relevant to the consumer.
- As above but allow applying an initial offset. This might be done by adjusting the data pointer, so there is no extra calculation to be done. Caller may still assume the data starts at `(0,0,0)`.
- As above but any of the 6 possible orderings may be chosen.
- No restrictions. Strides may be negative, or they might not make any sense.

The more support there is for handling strides, the more opportunities exist to use views on a buffer instead of copying them.

## 14.3.2 Member Function Documentation

### 14.3.2.1 clone()

```
virtual std::shared_ptr<DataBuffer> InternalZGY::DataBuffer::clone ( ) const [pure virtual]
```

Deep copy of a [DataBuffer](#)

Implemented in [InternalZGY::DataBufferNd< T, NDim >](#).

### 14.3.2.2 copyFrom()

```
virtual void InternalZGY::DataBuffer::copyFrom (
    const DataBuffer * src,
    const std::int64_t * srcorig,
    const std::int64_t * dstorig,
    const std::int64_t * cpyorig,
    const std::int64_t * cpysize ) [pure virtual]
```

Corresponds to `openzgy.impl._partialCopy()`. To make templates work better the C++ version is an instance method with 'this' as destination.

Implemented in [InternalZGY::DataBufferNd< T, NDim >](#).

### 14.3.2.3 makeDataBuffer3d()

```
std::shared_ptr< DataBuffer > InternalZGY::DataBuffer::makeDataBuffer3d (
    void * raw,
    std::int64_t nbytes,
    const std::array< std::int64_t, 3 > & size,
    RawDataType dtype ) [static]
```

Convert voidptr + nbytes + size3d to a [DataBuffer](#).

To avoid too much copy/paste, this method is able to create several different buffer types by calling different [DataBufferNd](#) constructors.

If voidptr is null this creates an uninitialized 3d [DataBuffer](#) with the given size and type and `isScalar() == false`. nbytes should be 0.

If voidptr is not null then this creates a scalar [DataBuffer](#) if nbytes indicates there is room for just one sample. If there are enough bytes it creates a regular [DataBuffer](#) pointing to the voidptr that was passed in.

The valid values for nbytes are:

0 - If and only if voidptr is null. `sizeof(T) - (T*)voidptr` contains the scalar to use. `sizeof(double) - (double*)voidptr` contains the scalar to use, `size[0]*size[1]*size[2]*sizeof(T) - (T*)voidptr` is used as-is.

Any other value will raise an exception.

Technically the scheme is ambiguous if `sizeof(T)` is 8 bytes and size indicates less than 8 bytes. Cannot know for sure whether the buffer points to a double or to 8/4/2/1 element of T. This should never happen in practice. If worried, remove the "double" feature or introduce a new `is_double` flag.

The scheme also cannot distinguish between a scalar size 1x1x1 and a regular buffer size 1x1x1. But those two are practically equivalent.

Caveat: If the data pointer is not correctly aligned to the type of data then the returned [DataBuffer](#) will also have a misaligned pointer.

Caveat: Since the user can pass in a void\* buffer, this will not be reference counted and could go out of scope before the databuffer does. This might change. To mitigate this somehow, if the code knows the buffer is no longer valid it can call `databuffer.clear()`.

### 14.3.2.4 scaleToFloat()

```
virtual std::shared_ptr<DataBuffer> InternalZGY::DataBuffer::scaleToFloat (
    const std::array< double, 2 > & ) [pure virtual]
```

Convert DataBufferNd<T,N> to DataBufferNd<float,N>, hiding leaf types.

Implemented in [InternalZGY::DataBufferNd< T, NDim >](#).

### 14.3.2.5 scaleToStorage()

```
virtual std::shared_ptr<DataBuffer> InternalZGY::DataBuffer::scaleToStorage (
    const std::array< double, 2 > & ,
    RawDataType ) [pure virtual]
```

Convert DataBufferNd<float,N> to DataBufferNd<T,N>, hiding leaf types.

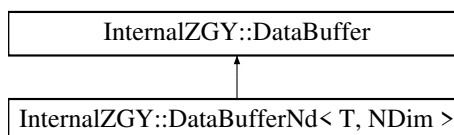
Implemented in [InternalZGY::DataBufferNd< T, NDim >](#).

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

- [impl/databuffer.h](#)
- [impl/databuffer.cpp](#)

## 14.4 InternalZGY::DataBufferNd< T, NDim > Class Template Reference

Inheritance diagram for InternalZGY::DataBufferNd< T, NDim >:



### Public Types

- enum { **ndim** = NDim }
- typedef T **value\_type**
- typedef [DataBufferNd< T, NDim >](#) **self\_type**
- typedef std::array< std::int64\_t, NDim > **ndsize\_t**

### Public Member Functions

- **DataBufferNd** (const [self\\_type](#) &)=delete
- [self\\_type](#) & **operator=** (const [self\\_type](#) &)=delete
- **DataBufferNd** (T scalar, const std::array< std::int64\_t, NDim > &size)
- **DataBufferNd** (const std::array< std::int64\_t, NDim > &size)
- **DataBufferNd** (const std::array< std::int64\_t, NDim > &size, const std::array< std::int64\_t, NDim > &stride)
- **DataBufferNd** (const std::shared\_ptr< T > &data, const std::array< std::int64\_t, NDim > &size)
- **DataBufferNd** (const std::shared\_ptr< T > &data, const std::array< std::int64\_t, NDim > &size, const std::array< std::int64\_t, NDim > &stride)
- T \* **data** ()
- const T \* **data** () const
- T **scalarValue** () const
- virtual bool **ownsdata** () const
- bool **isScalar** () const override
- bool **isAllSame** (const std::int64\_t \*used\_in) const override
- double **scalarAsDouble** () const override
- [RawDataType](#) **datatype** () const override
- void **fill** (double value) override
  - Set all samples to the same value.*
- void **clear** () override
- std::pair< double, double > **range** () const override
- std::uint32\_t **layout** () const override
- bool **is\_cstride** () const override
- void **check\_cstride** () const override
- std::string **toString** () const override
- std::shared\_ptr< void > **voidData** () override
- std::shared\_ptr< const void > **voidData** () const override
- bool **contiguous** () const override
- std::int64\_t **allocsize** () const override
- std::int64\_t **totalsize** () const override
- std::int64\_t **itemsz** () const override
- const std::int64\_t \* **sizeptr** () const override
- const std::int64\_t \* **strideptr** () const override
- std::array< std::int64\_t, 3 > **size3d** () const override
- std::array< std::int64\_t, 3 > **stride3d** () const override
- const ndsize\_t & **safesize** () const
- const ndsize\_t & **safestr** () const
- std::shared\_ptr< [DataBuffer](#) > **clone** () const
- std::shared\_ptr< [DataBuffer](#) > **scaleToFloat** (const std::array< double, 2 > &) override
- std::shared\_ptr< [DataBuffer](#) > **scaleToStorage** (const std::array< double, 2 > &, [RawDataType](#)) override
- std::shared\_ptr< [DataBuffer](#) > **slice1** (int dim, std::int64\_t start, std::int64\_t size) const override
  - Make a shallow view over the buffer, showing only part of the input.*
- std::shared\_ptr< [self\\_type](#) > **slice** (const ndsize\_t &neworig, const ndsize\_t &newsize) const
  - Make a shallow view over the buffer, showing only part of the input.*
- void **copyFrom** (const [DataBuffer](#) \*src, const std::int64\_t \*srcorig, const std::int64\_t \*dstorig, const std::int64\_t \*cpyorig, const std::int64\_t \*cpysize)

### Static Public Member Functions

- static void **copySubset** (const ndsize\_t &srcorig, const [self\\_type](#) &src, const ndsize\_t &dstorig, [self\\_type](#) &dst, const ndsize\_t &cpyorig, const ndsize\_t &cpysize)
- static void **copySubset** (const ndsize\_t &srcorig, const [self\\_type](#) &src, const ndsize\_t &dstorig, [self\\_type](#) &dst)
- static std::shared\_ptr< [DataBuffer](#) > **s\_scaleToFloat** (const [DataBuffer](#) \*, const std::array< double, 2 > &)
- static std::shared\_ptr< [DataBuffer](#) > **s\_scaleFromFloat** (const [DataBuffer](#) \*, const std::array< double, 2 > &)

## 14.4.1 Member Function Documentation

### 14.4.1.1 clear()

```
template<typename T , int NDim>
void InternalZGY::DataBufferNd< T, NDim >::clear  [override], [virtual]
```

Make the buffer empty. Size and stride are unchanged but the data will be nullptr. This will cause an access violation if trying to read or write data. The reason this is useful is that `_data` might point to something that is not reference counted by our smart pointer. The pointer's deleter is then a no-op. `clear()` should be called if we can no longer trust that the pointer is valid. A reproducible null pointer exception is way better than random crashes.

Implements [InternalZGY::DataBuffer](#).

### 14.4.1.2 clone()

```
template<typename T , int NDim>
std::shared_ptr< DataBuffer > InternalZGY::DataBufferNd< T, NDim >::clone  [virtual]
```

Make a deep copy of the buffer. Works both for scalars and normal buffers. TODO-Low: consider making the buffer contiguous.

Implements [InternalZGY::DataBuffer](#).

### 14.4.1.3 copyFrom()

```
template<typename T , int NDim>
void InternalZGY::DataBufferNd< T, NDim >::copyFrom (
    const DataBuffer * src,
    const std::int64_t * srcorig,
    const std::int64_t * dstorig,
    const std::int64_t * cpyorig,
    const std::int64_t * cpysize ) [virtual]
```

Corresponds to `openzgy.impl._partialCopy()`. To make templates work better the C++ version is an instance method with 'this' as destination.

Implements [InternalZGY::DataBuffer](#).



### 14.4.1.4 fill()

```
template<typename T , int NDim>
void InternalZGY::DataBufferNd< T, NDim >::fill (
    double value ) [override], [virtual]
```

Set all samples to the same value.

The value is passed as a float and will be cast to the correct type. There is NO check for overflow.

Some optimizing is attempted because gprof claims that (a) calling this function amounts to 50% of the read overhead, and (b) memset, if possible, would reduce that time significantly. I suspect that both of those numbers are false. An ad-hoc [Timer](#) reports 10% in both cases. But just in case I am wrong I have added the optimization.

Using memset is possible if the value to be set is zero (the common case for float cubes) and if the sample size is 1 (all int8 cubes). Int16 cubes are less likely to benefit.

Implements [InternalZGY::DataBuffer](#).

### 14.4.1.5 s\_scaleFromFloat()

```
template<typename T , int NDim>
std::shared_ptr< DataBuffer > InternalZGY::DataBufferNd< T, NDim >::s_scaleFromFloat (
    const DataBuffer * in,
    const std::array< double, 2 > & factors ) [static]
```

See [s\\_scaleToFloat\(\)](#) for details.

### 14.4.1.6 s\_scaleToFloat()

```
template<typename T , int NDim>
std::shared_ptr< DataBuffer > InternalZGY::DataBufferNd< T, NDim >::s_scaleToFloat (
    const DataBuffer * in,
    const std::array< double, 2 > & factors ) [static]
```

Copy the input [DataBuffer](#) into a newly allocated buffer of float. The input is assumed to have been read from a ZGY file, so this function will also apply the storage to float transform.

If the buffer is already floating point then it needs neither type conversion nor scaling and the function will return an empty pointer. Note that it does *not* return its input argument, because the input is a plain pointer and I don't want to mess with `std::enable_shared_from_this`.

Note that both `s_scaleToFloat` and `s_scaleFromFloat` are private static methods that are templated on the integral type. This means that `s_scaleToFloat` is templated on the input type and `s_scaleFromFloat` is templated on the desired target type. The public interface is in the virtual [scaleToFloat\(\)](#) and [scaleToStorage\(\)](#) methods.

If the buffer is non contiguous the contents of the padding area is unspecified. It might be garbage or values converted from the input. This violates the principle of least surprise but is likely harmless.

TODO-Low: Should I try to combine the type conversion and scaling done here with the functionality of `copySubset`? This might allow skipping one temporary buffer but I don't know how easy it will be for the bulk access to make use of it.

## 14.4.1.7 scaleToFloat()

```
template<typename T , int NDim>
std::shared_ptr< DataBuffer > InternalZGY::DataBufferNd< T, NDim >::scaleToFloat (
    const std::array< double, 2 > & ) [override], [virtual]
```

Convert [DataBufferNd](#)<T,N> to [DataBufferNd](#)<float,N>, hiding leaf types.

Implements [InternalZGY::DataBuffer](#).

## 14.4.1.8 scaleToStorage()

```
template<typename T , int NDim>
std::shared_ptr< DataBuffer > InternalZGY::DataBufferNd< T, NDim >::scaleToStorage (
    const std::array< double, 2 > & ,
    RawDataType ) [override], [virtual]
```

Convert [DataBufferNd](#)<float,N> to [DataBufferNd](#)<T,N>, hiding leaf types.

Implements [InternalZGY::DataBuffer](#).

## 14.4.1.9 size3d()

```
template<typename T , int NDim>
std::array< std::int64_t, 3 > InternalZGY::DataBufferNd< T, NDim >::size3d [override], [virtual]
```

Return the size of the buffer, assuming it is 3d. Which it almost always is. As a code smell this suggests that [DataBuffer](#) maybe shouldn't have been templated on NDim in the first place. Or have NDim as a regular variable. If fewer than 3 dimensions exist then the first index or indices are treated as size=1, stride=0. If more then 3 dimensions only the last 3 will be returned. Which is probably not very useful.

Implements [InternalZGY::DataBuffer](#).

## 14.4.1.10 sizeptr()

```
template<typename T , int NDim>
const std::int64_t * InternalZGY::DataBufferNd< T, NDim >::sizeptr [override], [virtual]
```

Return the size of the buffer, as a pointer to the first dimension. This is unsafe since the compiler cannot check the length. This function is meant for internal use in this file. Used by CopySubset and CopySize. TODO-Low consider making those friends so that this method can be declared private or inlined.

Implements [InternalZGY::DataBuffer](#).

### 14.4.1.11 slice()

```
template<typename T , int NDim>
std::shared_ptr< DataBufferNd< T, NDim > > InternalZGY::DataBufferNd< T, NDim >::slice (
    const ndsize_t & neworig,
    const ndsize_t & newsize ) const
```

Make a shallow view over the buffer, showing only part of the input.

If the input is contiguous and the slicing is done only on the slowest varying dimension then the result will be contiguous as well.

TODO-Worry: Much of the code using class [DataBuffer](#) assumes contiguous buffers without stating this explicitly.

### 14.4.1.12 slice1()

```
template<typename T , int NDim>
std::shared_ptr< DataBuffer > InternalZGY::DataBufferNd< T, NDim >::slice1 (
    int dim,
    std::int64_t start,
    std::int64_t size ) const [override], [virtual]
```

Make a shallow view over the buffer, showing only part of the input.

If the input is contiguous and the slicing is done only on the slowest varying dimension then the result will be contiguous as well.

TODO-Worry: Much of the code using class [DataBuffer](#) assumes contiguous buffers without stating this explicitly.

The [slice1\(\)](#) function signature allows just one dimension to change at a time. This corresponds to how the method is used in [OpenZGY](#). It also makes it possible to declare the method virtual in class [DataBuffer](#). Yet another place I regret templating on NDim.

Implements [InternalZGY::DataBuffer](#).

### 14.4.1.13 stride3d()

```
template<typename T , int NDim>
std::array< std::int64_t, 3 > InternalZGY::DataBufferNd< T, NDim >::stride3d [override],
[virtual]
```

Return the layout of the buffer, assuming it is 3d. See [size3d\(\)](#).

Implements [InternalZGY::DataBuffer](#).

#### 14.4.1.14 strideptr()

```
template<typename T , int NDim>
const std::int64_t * InternalZGY::DataBufferNd< T, NDim >::strideptr [override], [virtual]
```

Return the layout of the buffer, assuming it is 3d. See [sizeptr\(\)](#).

Implements [InternalZGY::DataBuffer](#).

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

- [impl/databuffer.h](#)
- [impl/databuffer.cpp](#)

## 14.5 InternalZGY::OrderedCornerPoints::Element Struct Reference

```
#include <cornerpoints.h>
```

### Public Member Functions

- [Element](#) ()
- [Element](#) (index\_type \_i, index\_type \_j)
- [Element](#) (index\_type \_i, index\_type \_j, [annot\\_type](#) \_il, [annot\\_type](#) \_xl, [coord\\_type](#) \_x, [coord\\_type](#) \_y)

### Public Attributes

- [index\\_type](#) i
- [index\\_type](#) j
- [annot\\_type](#) il
- [annot\\_type](#) xl
- [coord\\_type](#) x
- [coord\\_type](#) y

#### 14.5.1 Detailed Description

Map projection coordinate datatype. Represents one specific corner.

#### 14.5.2 Constructor & Destructor Documentation

##### 14.5.2.1 Element() [1/3]

```
InternalZGY::OrderedCornerPoints::Element::Element ( )
```

Map projection Y (Northing) coordinate. Default constructor, initializes all members to zero.

### 14.5.2.2 Element() [2/3]

```
InternalZGY::OrderedCornerPoints::Element::Element (
    index_type _i,
    index_type _j )
```

Constructor. Initializes i and j from args, remaining members are set to zero.

### 14.5.2.3 Element() [3/3]

```
InternalZGY::OrderedCornerPoints::Element::Element (
    index_type _i,
    index_type _j,
    annot_type _il,
    annot_type _xl,
    coord_type _x,
    coord_type _y )
```

Constructor. Initializes all members from args.

## 14.5.3 Member Data Documentation

### 14.5.3.1 il

`annot_type` InternalZGY::OrderedCornerPoints::Element::il

Bulk-data index along second dimension.

### 14.5.3.2 j

`index_type` InternalZGY::OrderedCornerPoints::Element::j

Bulk-data index along first dimension.

### 14.5.3.3 x

`coord_type` InternalZGY::OrderedCornerPoints::Element::x

Crossline annotation index.

### 14.5.3.4 xl

`annot_type` InternalZGY::OrderedCornerPoints::Element::xl

Inline annotation index.

## 14.5.3.5 y

`coord_type` `InternalZGY::OrderedCornerPoints::Element::y`

Map projection X (Easting) coordinate.

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

- impl/[cornerpoints.h](#)
- impl/[cornerpoints.cpp](#)

## 14.6 OpenZGY::Impl::EnumMapper Class Reference

convert between internal and external data types.

### Static Public Member Functions

- static `SampleDataType` [mapRawDataTypeToSampleDataType](#) (`InternalZGY::RawDataType`)
- static `UnitDimension` [mapRawHorizontalDimensionToUnitDimension](#) (`InternalZGY::RawHorizontalDimension`)
- static `UnitDimension` [mapRawVerticalDimensionToUnitDimension](#) (`InternalZGY::RawVerticalDimension`)
- static `InternalZGY::RawDataType` [mapSampleDataTypeToRawDataType](#) (`SampleDataType`)
- static `InternalZGY::RawHorizontalDimension` [mapUnitDimensionToRawHorizontalDimension](#) (`UnitDimension`)
- static `InternalZGY::RawVerticalDimension` [mapUnitDimensionToRawVerticalDimension](#) (`UnitDimension`)
- static `InternalZGY::ZgyInternalWriterArgs` [mapWriterArgs](#) (const `ZgyWriterArgs` &)
- static `InternalZGY::LodAlgorithm` [mapDecimationTypeToLodAlgorithm](#) (`DecimationType` value)
- static `std::vector< InternalZGY::LodAlgorithm >` [mapDecimationTypeToLodAlgorithm](#) (const `std::vector< DecimationType >` &values)

### 14.6.1 Detailed Description

convert between internal and external data types.

### 14.6.2 Member Function Documentation

#### 14.6.2.1 [mapDecimationTypeToLodAlgorithm\(\)](#) [1/2]

```
std::vector< InternalZGY::LodAlgorithm > OpenZGY::Impl::EnumMapper::mapDecimationTypeToLodAlgorithm (
    const std::vector< DecimationType > & values ) [static]
```

See [mapRawDataTypeToSampleType\(\)](#) for comments.

### 14.6.2.2 mapDecimationTypeToLodAlgorithm() [2/2]

```
InternalZGY::LodAlgorithm OpenZGY::Impl::EnumMapper::mapDecimationTypeToLodAlgorithm (
    DecimationType value ) [static]
```

See mapRawDataTypeToSampleType() for comments.

### 14.6.2.3 mapRawDataTypeToSampleDataType()

```
SampleDataType OpenZGY::Impl::EnumMapper::mapRawDataTypeToSampleDataType (
    InternalZGY::RawDataType value ) [static]
```

Map between enums used in the public API and the internal ones that might change without notice and might be used to define the actual numbers written to file.

As a general rule, if an invalid enum tag is encountered while converting from public to internal then this will throw an exception because it would be a user error. When converting from internal to public the error is probably bad data encountered on the file. The mapping function might just return "unknown", leaving to the caller to decide whether this should be silently ignored.

### 14.6.2.4 mapRawHorizontalDimensionToUnitDimension()

```
UnitDimension OpenZGY::Impl::EnumMapper::mapRawHorizontalDimensionToUnitDimension (
    InternalZGY::RawHorizontalDimension value ) [static]
```

See mapRawDataTypeToSampleType() for comments.

### 14.6.2.5 mapRawVerticalDimensionToUnitDimension()

```
UnitDimension OpenZGY::Impl::EnumMapper::mapRawVerticalDimensionToUnitDimension (
    InternalZGY::RawVerticalDimension value ) [static]
```

See mapRawDataTypeToSampleType() for comments.

The distinction between TWT and OWT, which is stored on the file, is lost. To my knowledge there is no code that recognizes OWD in ZGY files anyway, so it is better to not expose this to the API.

### 14.6.2.6 mapSampleDataTypeToRawDataType()

```
InternalZGY::RawDataType OpenZGY::Impl::EnumMapper::mapSampleDataTypeToRawDataType (
    SampleDataType value ) [static]
```

See mapRawDataTypeToSampleType() for comments.

### 14.6.2.7 mapUnitDimensionToRawHorizontalDimension()

```
InternalZGY::RawHorizontalDimension OpenZGY::Impl::EnumMapper::mapUnitDimensionToRawHorizontal←
Dimension (
    UnitDimension value ) [static]
```

See mapRawDataTypeToSampleType() for comments.

An explicit "unknown" is allowed, but passing e.g. "time" in this context (which is supposedly horizontal) will raise an exception.

## 14.6.2.8 mapUnitDimensionToRawVerticalDimension()

```
InternalZGY::RawVerticalDimension OpenZGY::Impl::EnumMapper::mapUnitDimensionToRawVertical↵  
Dimension (   
    UnitDimension value )    [static]
```

See mapRawDataTypeToSampleType() for comments.

An explicit "unknown" is allowed, but passing e.g. "arcangle" in this context (which is supposedly vertical) will raise an exception.

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

- [api.cpp](#)

## 14.7 InternalZGY::Environment Class Reference

```
#include <environment.h>
```

### Static Public Member Functions

- static int [getNumericEnv](#) (const char \*name, int dflt=0)  
*Get value as a number.*
- static std::string [getStringEnv](#) (const char \*name, const char \*dflt=0)  
*Get value as a string.*
- static void [putNumericEnv](#) (const char \*name, int value)  
*Convert number to string and store in the environment.*
- static void [putStringEnv](#) (const char \*name, const char \*value)  
*Store the supplied string in the environment.*

### 14.7.1 Detailed Description

Tiny helper class to access the environment variables of the process. Basically this is just a wrapper around getenv, but there are some Linux/Windows differences and it is a bad idea to duplicate that special handling too many places.

### 14.7.2 Member Function Documentation

#### 14.7.2.1 getNumericEnv()

```
int InternalZGY::Environment::getNumericEnv (   
    const char * name,   
    int dflt = 0 )    [static]
```

Get value as a number.

Get an environment variable as a number. If the environment variable is missing or empty then the supplied default value is returned instead. If no default value was given then 0 is returned. Trailing garbage is ignored, so if the variable doesn't start with a digit the function silently returns 0. I.e. NOT the provided default value.



### 14.7.2.2 getStringEnv()

```
std::string InternalZGY::Environment::getStringEnv (
    const char * name,
    const char * dflt = 0 ) [static]
```

Get value as a string.

Get an environment variable as a std::string. If the environment variable is empty or missing then the provided default value is returned. Or the empty string if there is no default. Attempting to distinguish between empty and missing variable is not portable. So this function won't let you. An environment variable with an empty name is always considered unset.

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

- impl/[environment.h](#)
- impl/environment.cpp

## 14.8 InternalZGY::ZgyInternalMeta::ErrorsWillCorruptFile Class Reference

### Public Member Functions

- **ErrorsWillCorruptFile** ([ZgyInternalMeta](#) \*owner)
- void **disarm** ()

### 14.8.1 Detailed Description

Duplicated between impl/bulk.cpp and impl/meta.cpp but sets different flags. See [ZgyInternalMeta::ErrorsWillCorruptFile](#) for details, and ZgyWriter::errorflag().

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

- impl/meta.cpp

## 14.9 InternalZGY::ZgyInternalBulk::ErrorsWillCorruptFile Class Reference

### Public Member Functions

- **ErrorsWillCorruptFile** ([ZgyInternalBulk](#) \*owner)
- void **disarm** ()

## 14.9.1 Detailed Description

Duplicated between impl/bulk.cpp and impl/meta.cpp but sets different flags.

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

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

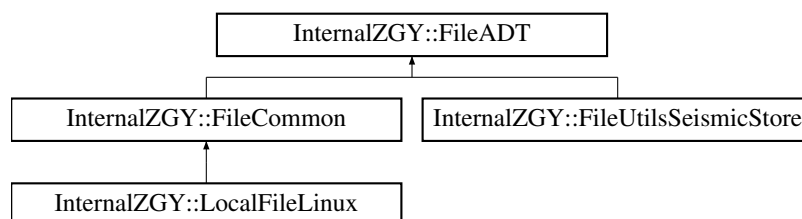
C++ note: Unlike Python it isn't trivial (or portable) to check whether a destructor is being called due to leaving scope normally or due to unwinding an exception. So in the C++ version the code should explicitly call `disarm()` at the end of the critical section.

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

- impl/bulk.cpp

## 14.10 InternalZGY::FileADT Class Reference

Inheritance diagram for InternalZGY::FileADT:



### Public Member Functions

- virtual void `xx_read` (void \*data, std::int64\_t offset, std::int64\_t size, UsageHint usagehint=UsageHint::Unknown)=0
- virtual void `xx_readv` (const ReadList &requests, bool parallel\_ok=false, bool immutable\_ok=false, bool transient\_ok=false, UsageHint usagehint=UsageHint::Unknown)=0
- virtual void `xx_write` (const void \*data, std::int64\_t offset, std::int64\_t size, UsageHint usagehint=UsageHint::Unknown)=0
- virtual void `xx_close` ()=0
- virtual std::int64\_t `xx_eof` () const =0
- virtual bool `xx_threadsafe` () const =0
- virtual bool `xx_iscloud` () const =0

### Static Public Member Functions

- static std::shared\_ptr< FileADT > **factory** (const std::string &filename, OpenMode mode, const OpenZGY::IOContext \*iocontext)

### Static Protected Member Functions

- static std::string [\\_nice](#) (std::int64\_t n)  
*Human readable number.*
- static void [\\_validate\\_read](#) (void \*data, std::int64\_t offset, std::int64\_t size, std::int64\_t eof, OpenMode mode)
- static void [\\_validate\\_write](#) (const void \*data, std::int64\_t offset, std::int64\_t size, OpenMode mode)
- static void [\\_validate\\_readv](#) (const ReadList &requests, std::int64\_t eof, OpenMode mode)

### 14.10.1 Member Function Documentation

#### 14.10.1.1 [xx\\_close\(\)](#)

```
void InternalZGY::FileADT::xx_close ( ) [pure virtual]
```

Close the file. This should always be done explicitly instead of assuming the destructor will handle it. If left to the destructor then any exceptions will be swallowed.

Implemented in [InternalZGY::LocalFileLinux](#).

#### 14.10.1.2 [xx\\_eof\(\)](#)

```
virtual std::int64_t InternalZGY::FileADT::xx_eof ( ) const [pure virtual]
```

Return the current end-of-file, i.e. the file size.

Implemented in [InternalZGY::LocalFileLinux](#).

#### 14.10.1.3 [xx\\_iscloud\(\)](#)

```
virtual bool InternalZGY::FileADT::xx_iscloud ( ) const [pure virtual]
```

Return true if the file is on the cloud. This might trigger some optimizations.

Implemented in [InternalZGY::LocalFileLinux](#).

## 14.10.1.4 `xx_read()`

```
virtual void InternalZGY::FileADT::xx_read (
    void * data,
    std::int64_t offset,
    std::int64_t size,
    UsageHint usagehint = UsageHint::Unknown ) [pure virtual]
```

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

Implemented in [InternalZGY::LocalFileLinux](#).

## 14.10.1.5 `xx_readv()`

```
virtual void InternalZGY::FileADT::xx_readv (
    const ReadList & requests,
    bool parallel_ok = false,
    bool immutable_ok = false,
    bool transient_ok = false,
    UsageHint usagehint = UsageHint::Unknown ) [pure virtual]
```

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

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

The delivery functor is called as `fn(void* data, std::int64_t size)`

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

Implemented in [InternalZGY::LocalFileLinux](#).

## 14.10.1.6 `xx_threadsafe()`

```
virtual bool InternalZGY::FileADT::xx_threadsafe ( ) const [pure virtual]
```

Return true if multiple reads can be in progress at the same time.

Implemented in [InternalZGY::LocalFileLinux](#).

### 14.10.1.7 xx\_write()

```
virtual void InternalZGY::FileADT::xx_write (
    const void * data,
    std::int64_t offset,
    std::int64_t size,
    UsageHint usagehint = UsageHint::Unknown ) [pure virtual]
```

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

Implemented in [InternalZGY::LocalFileLinux](#).

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

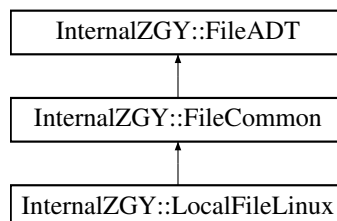
- [impl/file.h](#)
- [impl/file.cpp](#)

## 14.11 InternalZGY::FileCommon Class Reference

Implementation of some methods that might be shared.

```
#include <file.h>
```

Inheritance diagram for InternalZGY::FileCommon:



### Public Member Functions

- **FileCommon** (const std::string &filename, OpenMode mode, const [OpenZGY::IOContext](#) \*iocontext)
- virtual std::int64\_t [\\_real\\_eof](#) () const  
*Get the current file size for error reporting.*
- virtual void [\\_check\\_short\\_read](#) (std::int64\_t offset, std::int64\_t size, std::int64\_t got) const  
*Throw a descriptive error if there was something wrong with the read.*

### Protected Attributes

- [FileConfig](#) **\_config**
- OpenMode **\_mode**
- std::string **\_name**
- std::int64\_t **\_eof**
- std::shared\_ptr< [SummaryTimer](#) > **\_rtimer**
- std::shared\_ptr< [SummaryTimer](#) > **\_wtimer**

## Additional Inherited Members

### 14.11.1 Detailed Description

Implementation of some methods that might be shared.

Using this class is optional. Concrete classes can inherit directly from [FileADT](#) if they want to.

### 14.11.2 Member Function Documentation

#### 14.11.2.1 `_check_short_read()`

```
void InternalZGY::FileCommon::_check_short_read (
    std::int64_t offset,
    std::int64_t size,
    std::int64_t got ) const [virtual]
```

Throw a descriptive error if there was something wrong with the read.

Currently works for local files only. TODO-Low fix?

#### 14.11.2.2 `_real_eof()`

```
std::int64_t InternalZGY::FileCommon::_real_eof ( ) const [virtual]
```

Get the current file size for error reporting.

The default implementation in the base class just assumes that the `xx_eof()` that is (probably) maintained internally is correct.

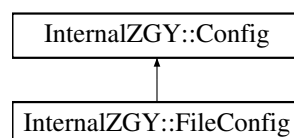
Reimplemented in [InternalZGY::LocalFileLinux](#).

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

- [impl/file.h](#)
- [impl/file.cpp](#)

## 14.12 InternalZGY::FileConfig Class Reference

Inheritance diagram for InternalZGY::FileConfig:



## Public Member Functions

- **FileConfig** (const [OpenZGY::IOContext](#) \*context)
- virtual std::string **dump** () const override

## Additional Inherited Members

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

- [impl/file.h](#)
- [impl/file.cpp](#)

## 14.13 InternalZGY::FileFactory Class Reference

### Public Types

- typedef std::function< std::shared\_ptr< [FileADT](#) >const std::string &, OpenMode, const [OpenZGY::IOContext](#) \*)> **factory\_t**

### Public Member Functions

- std::shared\_ptr< [FileADT](#) > **create** (const std::string &filename, OpenMode mode, const [OpenZGY::IOContext](#) \*iocontext)
- void **add\_factory** (const factory\_t &factory)

### Static Public Member Functions

- static [FileFactory](#) & **instance** ()

### 14.13.1 Member Function Documentation

#### 14.13.1.1 create()

```
std::shared_ptr< FileADT > InternalZGY::FileFactory::create (
    const std::string & filename,
    OpenMode mode,
    const OpenZGY::IOContext * iocontext )
```

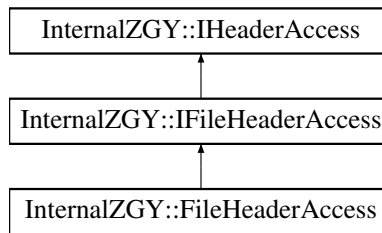
Try the registered factories in the order of registration until one is found that can handle this file. Caveat: If registration is done using static initializers then that order is undefined. So when a factory decides whether to handle a file or not it should not make assumptions about where it is in the list.

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

- [impl/file.h](#)
- [impl/file.cpp](#)

## 14.14 InternalZGY::FileHeaderAccess Class Reference

Inheritance diagram for InternalZGY::FileHeaderAccess:



### Public Member Functions

- virtual podbytes\_t **podbytes** () const override
- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset)
- virtual void **byteswap** ()
- virtual void **dump** (std::ostream &out, const std::string &prefix="") override
- virtual std::array< std::uint8\_t, 4 > **magic** () const override
- virtual std::uint32\_t **version** () const override

### Public Attributes

- [FileHeaderPOD](#) \_pod

### Additional Inherited Members

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

- impl/meta.cpp

## 14.15 InternalZGY::FileHeaderPOD Class Reference

### Public Attributes

- std::uint8\_t \_**magic** [4]
- std::uint32\_t \_**version**

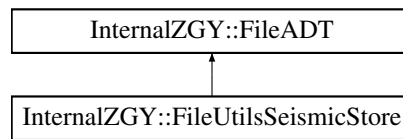
The documentation for this class was generated from the following file:

- impl/meta.cpp



## 14.16 InternalZGY::FileUtilsSeismicStore Class Reference

Inheritance diagram for InternalZGY::FileUtilsSeismicStore:



### Public Member Functions

- virtual void **deleteFile** (const std::string &name, bool missing\_ok) const =0

### Additional Inherited Members

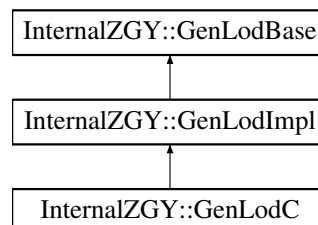
The documentation for this class was generated from the following file:

- impl/[file\\_sd.h](#)

## 14.17 InternalZGY::GenLodBase Class Reference

```
#include <genlod.h>
```

Inheritance diagram for InternalZGY::GenLodBase:



### Public Member Functions

- [GenLodBase](#) (const [index3\\_t](#) &size, const [index3\\_t](#) &bricksizes, [RawDataType](#) dtype, const std::array< double, 2 > &histogram\_range, std::int32\_t nlods\_in, const std::vector< [LodAlgorithm](#) > &decimation, const std::shared\_ptr< [HistogramData](#) > &histogram, double defaultvalue, const std::function< bool(std::int64\_t, std::int64\_t)> &progress, bool verbose)

### Protected Member Functions

- virtual std::shared\_ptr< [DataBuffer](#) > [\\_read](#) (std::int32\_t lod, const [index3\\_t](#) &pos, const [index3\\_t](#) &size)
- virtual void [\\_write](#) (std::int32\_t lod, const [index3\\_t](#) &pos, const std::shared\_ptr< const [DataBuffer](#) > &data)
- virtual void [\\_savestats](#) ()
- void [\\_report](#) (const [DataBuffer](#) \*data)
- std::string [\\_prefix](#) (std::int32\_t lod)

## Static Protected Member Functions

- static std::string [\\_format\\_result](#) (const std::shared\_ptr< [DataBuffer](#) > &data)

## Protected Attributes

- std::int32\_t [\\_nlods](#)
- std::int64\_t [\\_total](#)
- std::int64\_t [\\_done](#)
- [index3\\_t](#) [\\_surveysize](#)
- [index3\\_t](#) [\\_bricksize](#)
- [RawDataType](#) [\\_dtype](#)
- std::array< double, 2 > [\\_histogram\\_range](#)
- std::vector< [LodAlgorithm](#) > [\\_decimation](#)
- std::shared\_ptr< [HistogramData](#) > [\\_wa\\_histogram](#)
- double [\\_wa\\_defaultstorage](#)
- std::function< bool(std::int64\_t, std::int64\_t)> [\\_progress](#)
- bool [\\_verbose](#)

### 14.17.1 Detailed Description

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

### 14.17.2 Constructor & Destructor Documentation

#### 14.17.2.1 GenLodBase()

```
InternalZGY::GenLodBase::GenLodBase (
    const index3\_t & size,
    const index3\_t & bricksize,
    RawDataType dtype,
    const std::array< double, 2 > & histogram_range,
    std::int32_t nlods_in,
    const std::vector< LodAlgorithm > & decimation,
    const std::shared_ptr< HistogramData > & histogram,
    double defaultstorage,
    const std::function< bool(std::int64_t, std::int64_t)> & progress,
    bool verbose )
```

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

Note that the [WeightedAverage](#) algorithm requires a histogram to work. If no histogram was provided then the current contents of the accumulated histogram will be used. This is unfortunate and might cause brick artifacts. Especially in the first few bricks that are generated. With a non-recursive algorithm (plan C) and with only lod2 and above uses weighted average then this is unproblematic. Because in that case we will be done with the histogram

once we need it. TODO-Low consider doing an initial pass with a statistical sampling of the lod0 data, only for use with weighted average. There will be a minor issue with some values appearing to have zero frequency, but this should not cause any trouble. (assume "1").

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

### 14.17.3 Member Function Documentation

#### 14.17.3.1 \_format\_result()

```
std::string InternalZGY::GenLodBase::_format_result (
    const std::shared_ptr< DataBuffer > & data ) [static], [protected]
```

For debugging and logging only.

#### 14.17.3.2 \_prefix()

```
std::string InternalZGY::GenLodBase::_prefix (
    std::int32_t lod ) [protected]
```

For debugging and logging only.

#### 14.17.3.3 \_read()

```
std::shared_ptr< DataBuffer > InternalZGY::GenLodBase::_read (
    std::int32_t lod,
    const index3\_t & pos,
    const index3\_t & size ) [protected], [virtual]
```

This is a stub that must be redefined except for low level unit tests. Read a block from the ZGY file (plans B and C) or the application (plan D). The lod parameter will always be 0 for plans C and D. Returns a ScalarBuffer if all constant, else a 3D numpy array.

Reimplemented in [InternalZGY::GenLodC](#).

#### 14.17.3.4 \_report()

```
void InternalZGY::GenLodBase::_report (
    const DataBuffer * data ) [protected]
```

Invoke the user's progress callback if any. Keep track of how many bricks we have processed. Both reads and writes increment the same counter. For plan C the reads will cover all blocks in lod 0 and the writes will cover all blocks in lod > 0. For plan D all blocks are written which means the computation of \_total done in `init` might need to change.

### 14.17.3.5 `_savestats()`

```
void InternalZGY::GenLodBase::_savestats ( ) [protected], [virtual]
```

This is a stub that must be redefined except for low level unit tests. Finalize and write the computed statistics and histogram to the file.

Reimplemented in [InternalZGY::GenLodC](#).

### 14.17.3.6 `_write()`

```
void InternalZGY::GenLodBase::_write (
    std::int32_t lod,
    const index3_t & pos,
    const std::shared_ptr< const DataBuffer > & data ) [protected], [virtual]
```

This is a stub that must be redefined except for low level unit tests. Write a block to the ZGY file. Silently ignore writes of data that is known to have been read directly from the file. For plans B and C this means ignoring all writes to lod 0.

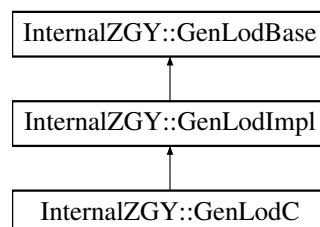
Reimplemented in [InternalZGY::GenLodC](#).

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

- [impl/genlod.h](#)
- [impl/genlod.cpp](#)

## 14.18 InternalZGY::GenLodC Class Reference

Inheritance diagram for InternalZGY::GenLodC:



### Public Member Functions

- [GenLodC](#) (const std::shared\_ptr< [ZgyInternalBulk](#) > &accessor, const std::shared\_ptr< [ZgyInternalMeta](#) > &meta, const [compressor\\_t](#) &lodcompressor, const std::vector< [LodAlgorithm](#) > &decimation, const std::function< bool(std::int64\_t, std::int64\_t)> &progress, bool verbose)

## Protected Member Functions

- `std::shared_ptr< DataBuffer > _read` (`std::int32_t lod`, `const index3\_t &pos`, `const index3\_t &size`) override
- `void _write` (`std::int32_t lod`, `const index3\_t &pos`, `const std::shared_ptr< const DataBuffer > &data`) override
- `void _savestats` () override

## Additional Inherited Members

### 14.18.1 Constructor & Destructor Documentation

#### 14.18.1.1 GenLodC()

```
InternalZGY::GenLodC::GenLodC (
    const std::shared_ptr< ZgyInternalBulk > & accessor,
    const std::shared_ptr< ZgyInternalMeta > & meta,
    const compressor\_t & lodcompressor,
    const std::vector< LodAlgorithm > & decimation,
    const std::function< bool(std::int64_t, std::int64_t)> & progress,
    bool verbose )
```

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

### 14.18.2 Member Function Documentation

#### 14.18.2.1 \_read()

```
std::shared_ptr< DataBuffer > InternalZGY::GenLodC::_read (
    std::int32_t lod,
    const index3\_t & pos,
    const index3\_t & size ) [override], [protected], [virtual]
```

See base class for details.

Reimplemented from [InternalZGY::GenLodBase](#).

### 14.18.2.2 `_savestats()`

```
void InternalZGY::GenLodC::_savestats ( ) [override], [protected], [virtual]
```

This is a stub that must be redefined except for low level unit tests. Finalize and write the computed statistics and histogram to the file.

Reimplemented from [InternalZGY::GenLodBase](#).

### 14.18.2.3 `_write()`

```
void InternalZGY::GenLodC::_write (
    std::int32_t lod,
    const index3_t & pos,
    const std::shared_ptr< const DataBuffer > & data ) [override], [protected],
[virtual]
```

See base class for details.

Reimplemented from [InternalZGY::GenLodBase](#).

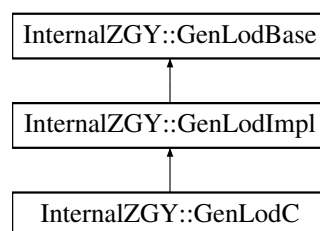
The documentation for this class was generated from the following files:

- [impl/genlod.h](#)
- [impl/genlod.cpp](#)

## 14.19 InternalZGY::GenLodImpl Class Reference

```
#include <genlod.h>
```

Inheritance diagram for InternalZGY::GenLodImpl:



### Public Member Functions

- [GenLodImpl](#) (const [index3\\_t](#) &size, const [index3\\_t](#) &bricksizes, [RawDataType](#) dtype, const std::array< double, 2 > &histogram\_range, std::int32\_t nlods\_in, const std::vector< [LodAlgorithm](#) > &decimation, const std::shared\_ptr< [HistogramData](#) > &histogram, double defaultvalue, const std::function< bool(std::int64\_t, std::int64\_t)> &progress, bool verbose)
- std::tuple< std::shared\_ptr< [StatisticData](#) >, std::shared\_ptr< [HistogramData](#) > > [call](#) ()

## Protected Member Functions

- `template<typename T >`  
`void _accumulateT (const std::shared_ptr< const DataBuffer > &data_in)`
- `void _accumulate (const std::shared_ptr< const DataBuffer > &data)`
- `std::shared_ptr< DataBuffer > _calculate (const index3\_t &readpos_in, std::int32_t readlod)`
- `std::shared_ptr< DataBuffer > _decimate (const std::shared_ptr< const DataBuffer > &data, std::int64_t lod)`
- `std::shared_ptr< DataBuffer > _paste1 (const std::shared_ptr< DataBuffer > &result, const std::shared_ptr< const DataBuffer > &more, std::int64_t ioff, std::int64_t joff)`
- `std::shared_ptr< const DataBuffer > _paste4 (const std::shared_ptr< const DataBuffer > &d00, const std::shared_ptr< const DataBuffer > &d01, const std::shared_ptr< const DataBuffer > &d10, const std::shared_ptr< const DataBuffer > &d11)`

## Static Protected Member Functions

- `static std::array< double, 2 > suggestHistogramRange (const std::array< double, 2 > &writtenrange, RawDataType dtype)`

## Additional Inherited Members

### 14.19.1 Detailed Description

Abstract class for generating low resolution bricks, histogram, and statistics. The inherited methods for I/O are still stubs.

### 14.19.2 Constructor & Destructor Documentation

#### 14.19.2.1 GenLodImpl()

```
InternalZGY::GenLodImpl::GenLodImpl (
    const index3\_t & size,
    const index3\_t & bricksize,
    RawDataType dtype,
    const std::array< double, 2 > & histogram_range,
    std::int32_t nlods_in,
    const std::vector< LodAlgorithm > & decimation,
    const std::shared_ptr< HistogramData > & histogram,
    double defaultstorage,
    const std::function< bool(std::int64_t, std::int64_t)> & progress,
    bool verbose )
```

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

## 14.19.3 Member Function Documentation

### 14.19.3.1 `_accumulate()`

```
void InternalZGY::GenLodImpl::_accumulate (
    const std::shared_ptr< const DataBuffer > & data ) [protected]
```

Keep a running tally of statistics and histogram.

### 14.19.3.2 `_calculate()`

```
std::shared_ptr< DataBuffer > InternalZGY::GenLodImpl::_calculate (
    const index3\_t & readpos_in,
    std::int32_t readlod ) [protected]
```

Read data from the specified (readpos, readlod) and store it back. The function will itself decide how much to read. But with several constraints. Always read full traces. Size in i and j needs to be  $2 * bs * 2^N$  where bs is the file's brick size in that dimension, Clipped to the survey boundaries. This might give an empty result.

TODO-Performance: Allow the application to configure how much memory we are allowed to use. Increase the block size accordingly. Larger bricks might help the bulk layer to become more efficient.

When readlod is 0 and the data was read from the ZGY file then the writing part is skipped. Since the data is obviously there already.

In addition to reading and writing at the readlod level, the method will compute a single decimated buffer at readlod+1 and return it. As with the read/write the buffer might be smaller at the survey edge. Note that the caller is responsible for storing the decimated data.

Full resolution data (lod 0) will be read from file (plan C) or the application (plan D). Low resolution is computed by a recursive call to this function (plans C and D) or by reading the file (plan B). Note that currently only plan C is fully implemented.

For plans B and C a single call needs to be made to read the brick (there is by definition just one) at the highest level of detail. This will end up computing all possible low resolution bricks and storing them. For plan B the caller must iterate.

The function is also responsible for collecting statistics and histogram data. Note that some of the decimation algorithms use the histogram of the entire file. Ideally the histogram of the entire file should be available before decimation starts but that is impractical. At least make sure the histogram update is done early enough and the decimation late enough that the chunk of data being decimated has already been added to the histogram.

### 14.19.3.3 `_decimate()`

```
std::shared_ptr< DataBuffer > InternalZGY::GenLodImpl::_decimate (
    const std::shared_ptr< const DataBuffer > & data,
    std::int64_t lod ) [protected]
```

Return a decimated version of the input buffer with half the size (rounded up) in each dimension. In total the result will be  $\sim 1/8$  the size of the input.

Lod refers to the level being generated. Must be  $\geq 1$ .



### 14.19.3.4 `_paste1()`

```
std::shared_ptr< DataBuffer > InternalZGY::GenLodImpl::_paste1 (
    const std::shared_ptr< DataBuffer > & result,
    const std::shared_ptr< const DataBuffer > & more,
    std::int64_t ioff,
    std::int64_t joff ) [protected]
```

See [\\_paste4\(\)](#) for details.

### 14.19.3.5 `_paste4()`

```
std::shared_ptr< const DataBuffer > InternalZGY::GenLodImpl::_paste4 (
    const std::shared_ptr< const DataBuffer > & d00,
    const std::shared_ptr< const DataBuffer > & d01,
    const std::shared_ptr< const DataBuffer > & d10,
    const std::shared_ptr< const DataBuffer > & d11 ) [protected]
```

Combine 4 buffers into one. Input buffers may be None (do not paste) or ScalarBuffer (paste a constant value). If all not-None buffers are just scalars then the return from this function will also be a scalar. d01 adds more data in the J direction, so it starts at i=0, j>0 in the target. Similarly d10 adds more in the J direction. And d11 in the diagonal.

Performance note: It is in theory possible to avoid some buffer copies, this one in particular, by passing our 4 or 8 buffers to the decimation algorithm instead of a combined buffer. The algorithms remain just as efficient. BUT if sizes can vary or bricks can be missing or number of bricks differs in level 1 because we read directly from the file then things can get really complicated really fast.

### 14.19.3.6 `call()`

```
std::tuple< std::shared_ptr< StatisticData >, std::shared_ptr< HistogramData > > InternalZGY::GenLodImpl::call ( )
```

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

TODO-Performance: If the bulk layer is made thread safe for writing it is possible to do parallel processing at this high level. E.g. do this for just one level: Split into 4 sub-tasks that each execute in one thread. Special handling will be needed for the single highest-level brick. All 4 threads need to be joined before that one can be done. With one level here and with 4 threads used in [\\_calculate\(\)](#) this means we will be reading with 16 threads. But there are serious caveats:

- Using 4 threads here means there is just 1/4th of the memory available to each thread, which means requests may become smaller which means we might not see any benefit at all.
- The added complexity both here and in making the bulk write thread safe might not be worth the trouble.

### 14.19.3.7 suggestHistogramRange()

```
std::array< double, 2 > InternalZGY::GenLodImpl::suggestHistogramRange (
    const std::array< double, 2 > & writtenrange,
    RawDataType dtype ) [static], [protected]
```

Choose the histogram range to use.

The range of all sample values seen until now, i.e. everything written, is passed in. The possibility to overwrite data or (future) append to existing file makes this not accurate but still probably good enough.

If writtenrange is invalid this means that no finite samples have been written. Choose an arbitrary range in that case instead of ending up with a lot of obscure corner cases. But, TODO-Low the range needs to include defaultvalue as there might be unwritten bricks.

TODO-Low: If data is being appended, we will still re-compute the entire histogram. Include the current value range stored on file in case the update only consists of smaller values. Problem is, the first write might not have bothered to finalize and thus did not save this information. I probably need to "finalize" with an empty histogram. Also bear in mind that the stored range will be in user and not storage values.

Widen the histogram range for integral types to cover the entire possible range, not just the sample range written. For int8/uint8 this is a no-brainer because having less than one possible sample value in each bin inevitably leads to some empty bins even for a completely smooth distribution of the input. For int16/uint16 it is still workable but it had been better to use a compromise: Use a narrower range or narrowed using an integer factor. Or sidestep the issue by internally using a 64k histogram that will get trimmed down to 256 entries later.

TODO-Low should the suggested range for float data include defaultvalue? Which for floats is always zero? Technically the code should keep track of whether all bricks have been explicitly written and if not include zero. It doesn't. This "bug" is probably in the "extreme nitpicking" category.

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

- impl/genlod.h
- impl/genlod.cpp

## 14.20 InternalZGY::GUID Class Reference

Simplified GUID handling. Only big endian, random number guids.

```
#include <guid.h>
```

### Public Types

- typedef std::array< std::uint8\_t, 16 > **guid\_bytes\_t**

### Public Member Functions

- **GUID** (const guid\_bytes\_t &in)
- **GUID** (nullptr\_t)
- std::string **toString** () const
- void **copyTo** (std::uint8\_t \*ptr, std::int64\_t len)

### 14.20.1 Detailed Description

Simplified GUID handling. Only big endian, random number guides.

The old ZGY accessor relies on an external dependency to an "uuid" package which tends to change between linux distros. This has caused a number of headaches. So, don't do that.

### 14.20.2 Member Function Documentation

#### 14.20.2.1 copyTo()

```
void InternalZGY::GUID::copyTo (
    std::uint8_t * ptr,
    std::int64_t len )
```

Copy the raw bytes of the guid to the specified pointer. The length is redundant. It should always be 16 but is included because it is then more obvious at the calling site what is going on.

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

- [impl/guid.h](#)
- [impl/guid.cpp](#)

## 14.21 InternalZGY::HeaderAccessFactory Class Reference

### Static Public Member Functions

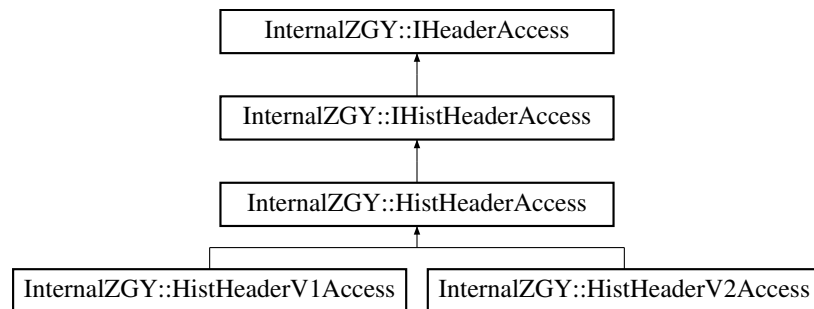
- static std::shared\_ptr< [IFileHeaderAccess](#) > **createFileHeader** ()
- static std::shared\_ptr< [IOffsetHeaderAccess](#) > **createOffsetHeader** (std::uint32\_t)
- static std::shared\_ptr< [IInfoHeaderAccess](#) > **createInfoHeader** (std::uint32\_t)
- static std::shared\_ptr< [IHistHeaderAccess](#) > **createHistHeader** (std::uint32\_t)
- static std::shared\_ptr< [ILookupTableAccess](#) > **createAlphaLookup** (std::uint32\_t)
- static std::shared\_ptr< [ILookupTableAccess](#) > **createBrickLookup** (std::uint32\_t)

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

- [impl/meta.h](#)
- [impl/meta.cpp](#)

## 14.22 InternalZGY::HistHeaderAccess Class Reference

Inheritance diagram for InternalZGY::HistHeaderAccess:



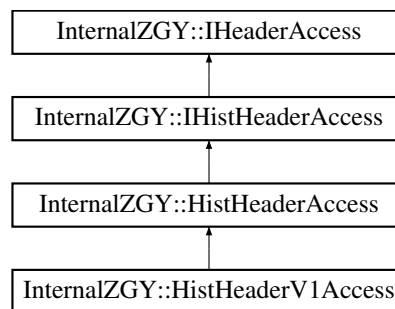
### Additional Inherited Members

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

- `impl/meta.cpp`

## 14.23 InternalZGY::HistHeaderV1Access Class Reference

Inheritance diagram for InternalZGY::HistHeaderV1Access:



### Public Member Functions

- virtual `podbytes_t` **podbytes** () const override
- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset, std::int64\_t size) override
- virtual void **byteswap** () override
- virtual void **calculate** ()
- virtual void **sethisto** (double minvalue, double maxvalue, const std::int64\_t \*bins, std::int64\_t bincount)
- virtual `std::int64_t` **bincount** () const
- virtual `std::int64_t` **samplecount** () const
- virtual double **minvalue** () const
- virtual double **maxvalue** () const
- virtual const `std::int64_t *` **bins** () const

### Public Attributes

- [HistHeaderV1POD\\_pod](#)

### Additional Inherited Members

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

- impl/meta.cpp

## 14.24 InternalZGY::HistHeaderV1POD Class Reference

Physical layout of Histogram Header version 1.

### Public Attributes

- float **\_max**
- float **\_min**
- std::uint32\_t **\_bin** [256]

### 14.24.1 Detailed Description

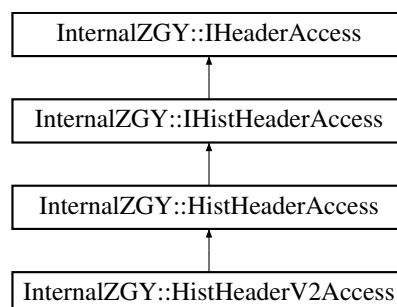
Physical layout of Histogram Header version 1.

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

- impl/meta.cpp

## 14.25 InternalZGY::HistHeaderV2Access Class Reference

Inheritance diagram for InternalZGY::HistHeaderV2Access:



## Public Member Functions

- virtual podbytes\_t **podbytes** () const override
- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset, std::int64\_t size) override
- virtual void **byteswap** () override
- virtual void **calculate** ()
- virtual std::int64\_t **bincount** () const
- virtual std::int64\_t **samplecount** () const
- virtual double **minvalue** () const
- virtual double **maxvalue** () const
- virtual const std::int64\_t \* **bins** () const
- virtual void **sethisto** (double minvalue, double maxvalue, const std::int64\_t \*bins, std::int64\_t bincount)

## Public Attributes

- [HistHeaderV2POD](#) \_pod

## Additional Inherited Members

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

- impl/meta.cpp

## 14.26 InternalZGY::HistHeaderV2POD Class Reference

Physical layout of Histogram Header version 2 and 3.

### Public Attributes

- std::int64\_t \_cnt
- float \_min
- float \_max
- std::int64\_t \_bin [256]

### 14.26.1 Detailed Description

Physical layout of Histogram Header version 2 and 3.

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

- impl/meta.cpp

## 14.27 InternalZGY::HistogramBuilder Class Reference

Collect statistics and histogram for bulk data.

```
#include <histogrambuilder.h>
```

## Public Types

- typedef int **size\_type**
- typedef StatisticData::count\_type **count\_type**

## Public Member Functions

- [HistogramBuilder](#) (size\_type \_nbins, double \_min, double \_max)
- [HistogramBuilder](#) (const count\_type \*bins, int nbins, double min, double max, count\_type scnt, double ssum, double sssq, double smin, double smax)
- [HistogramBuilder](#) & **operator+=** (const [HistogramBuilder](#) &other)
- [HistogramBuilder](#) & **operator-=** (const [HistogramBuilder](#) &other)
- [HistogramBuilder](#) & **operator\*=** (count\_type factor)
- bool **operator==** (const [HistogramBuilder](#) &other) const
- bool **operator!=** (const [HistogramBuilder](#) &other) const
- [StatisticData](#) **gethiststats** () const
- template<typename It >  
void **add** (It begin, It end)
- void **scale** (double oldmin, double oldmax, double newmin, double newmax)

### 14.27.1 Detailed Description

Collect statistics and histogram for bulk data.

Note the following caveat with histogram data. The histogram is described by number of bins and a pair of min/max values. It is not obvious whether those values describe the center value of the first and last bin, or whether they are the open ended range representing the whole histogram. I.e. the first value (inclusive) of the first bin and the last value (exclusive) of the last bin. [HistogramBuilder](#) uses the former definition.

### 14.27.2 Constructor & Destructor Documentation

#### 14.27.2.1 HistogramBuilder() [1/2]

```
InternalZGY::HistogramBuilder::HistogramBuilder (  
    size_type nbins,  
    double min,  
    double max )
```

Create a new [HistogramBuilder](#).

## 14.27.2.2 HistogramBuilder() [2/2]

```
InternalZGY::HistogramBuilder::HistogramBuilder (
    const count_type * bins,
    int nbins,
    double min,
    double max,
    count_type scnt,
    double ssum,
    double sssq,
    double smin,
    double smax )
```

Copy constructor from a histogram passed as discrete information. Useful for converting from some other histogram type. Always create a fixed-range histogram with the same range and number of bins as the source. If this is not what you need, convert it using [operator+=\(\)](#). The above, below, and infinite counts are left at zero.

## 14.27.3 Member Function Documentation

### 14.27.3.1 add()

```
template<typename It >
void InternalZGY::HistogramBuilder::add (
    It begin,
    It end )
```

Add samples from an iterator to a histogram. If the caller has already calculated statistics from the iterator, those can be passed it so this function doesn't need to do it again.

### 14.27.3.2 gethiststats()

```
StatisticData InternalZGY::HistogramBuilder::gethiststats ( ) const
```

Statistics calculated from bins

Calculate statistics directly from the histogram. See the overloaded [StatisticData](#) constructor for details. Here we don't know whether the histogram was built from 8-bit data, so we play it safe and assume it wasn't.

### 14.27.3.3 operator"!="()

```
bool InternalZGY::HistogramBuilder::operator!= (
    const HistogramBuilder & other ) const
```

See [operator==](#) for a description.



### 14.27.3.4 operator\*=( )

```
HistogramBuilder & InternalZGY::HistogramBuilder::operator*= (
    count_type factor )
```

Multiply [HistogramBuilder](#) with a constant N, equivalent to creating a new instance and adding the old one to it N times. N can be negative.

### 14.27.3.5 operator+=( )

```
HistogramBuilder & InternalZGY::HistogramBuilder::operator+= (
    const HistogramBuilder & other )
```

Add the samples found in another histogram, just as if the samples had been added one at a time using `Add()`.

### 14.27.3.6 operator-= ( )

```
HistogramBuilder & InternalZGY::HistogramBuilder::operator-= (
    const HistogramBuilder & other )
```

Subtract the samples found in another histogram, more or less undoing the effect of `Add`. The min/max range in the statistics might be left showing a too wide range.

### 14.27.3.7 operator==( )

```
bool InternalZGY::HistogramBuilder::operator== (
    const HistogramBuilder & other ) const
```

Two histograms are considered equal if they return the same bin count for any input value. In practice there is some slop, as only the center of each bin of either histogram is checked. The statistics information need not match. Nor is there any check that the histograms have the same range or even the same number of bins.

### 14.27.3.8 scale( )

```
void InternalZGY::HistogramBuilder::scale (
    double oldmin,
    double oldmax,
    double newmin,
    double newmax )
```

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

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

- [impl/histogrambuilder.h](#)
- [impl/histogrambuilder.cpp](#)

## 14.28 InternalZGY::HistogramData Class Reference

A histogram for a data set.

```
#include <histogramdata.h>
```

### Public Types

- typedef std::int32\_t **size\_type**
- typedef std::int64\_t **count\_type**

### Public Member Functions

- [HistogramData](#) (size\_type \_nbins, double \_min, double \_max)
- [HistogramData](#) (const count\_type \*bins, int nbins, double min, double max)
- [HistogramData](#) (const [HistogramData](#) &other)
- [HistogramData](#) & [operator=](#) (const [HistogramData](#) &other)
- [HistogramData](#) & [operator+=](#) (const [HistogramData](#) &other)
- [HistogramData](#) & [operator-=](#) (const [HistogramData](#) &other)
- [HistogramData](#) & [operator\\*=](#) (count\_type factor)
- bool [operator==](#) (const [HistogramData](#) &other) const
- bool [operator!=](#) (const [HistogramData](#) &other) const
- count\_type [get](#) (double value) const
- count\_type [getcount](#) () const
- void [scale](#) (double oldmin, double oldmax, double newmin, double newmax)
- void [clear](#) ()
- void [calculateConversionFactors](#) (double \*A, double \*B) const

### Static Public Member Functions

- static void [getLinearTransform](#) (double \*offset, double \*[scale](#), double oldmin, double oldmax, double newmin, double newmax)

### 14.28.1 Detailed Description

A histogram for a data set.

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  $\text{min}_+ \pm \text{binwidth}/2$ .

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.

## 14.28.2 Constructor & Destructor Documentation

### 14.28.2.1 HistogramData() [1/3]

```
InternalZGY::HistogramData::HistogramData (
    size_type nbins,
    double min,
    double max )
```

Create a new [HistogramData](#).

### 14.28.2.2 HistogramData() [2/3]

```
InternalZGY::HistogramData::HistogramData (
    const count_type * bins,
    int nbins,
    double min,
    double max )
```

Copy constructor from a histogram passed as discrete information. Useful for converting from some other histogram type. Always create a fixed-range histogram with the same range and number of bins as the source. If this is not what you need, convert it using [operator+=\(\)](#).

### 14.28.2.3 HistogramData() [3/3]

```
InternalZGY::HistogramData::HistogramData (
    const HistogramData & other )
```

Standard copy constructor. Needs to be explicit because of the allocated data for bins.

## 14.28.3 Member Function Documentation

### 14.28.3.1 calculateConversionFactors()

```
void InternalZGY::HistogramData::calculateConversionFactors (
    double * A,
    double * B ) const
```

Get the linear transform for mapping from application values to bin numbers. The resulting bin number is meant to be rounded to nearest integral value. If the histogram range is not initialized yet (can happen if histogram is empty) then a transform is returned that will map all values to a non-existent bin.

## 14.28.3.2 clear()

```
void InternalZGY::HistogramData::clear ( )
```

Erase all values from the histogram. The range remains unchanged.

## 14.28.3.3 get()

```
HistogramData::count_type InternalZGY::HistogramData::get (
    double value ) const
```

Get sample count for this value

Given a sample value, return how many times this sample was found. This is almost the inverse of AddOne, but the function does no scaling of the value.

## 14.28.3.4 getcount()

```
HistogramData::count_type InternalZGY::HistogramData::getcount ( ) const
```

Get sample count for all values

Return the total number of samples added to any bin. Currently this is returned by summing all the bins, if this is a performance problem that it is possible to maintain a separate count member. Called from compare().

## 14.28.3.5 getLinearTransform()

```
void InternalZGY::HistogramData::getLinearTransform (
    double * offset,
    double * scale,
    double oldmin,
    double oldmax,
    double newmin,
    double newmax ) [static]
```

Get the linear transform (offset, scale) that will map values from the range oldmin..oldmax to the range newmin..newmax. If either range is empty, the identity transform will be returned.

## 14.28.3.6 operator"!="()

```
bool InternalZGY::HistogramData::operator!= (
    const HistogramData & other ) const
```

See operator== for a description.

## 14.28.3.7 operator\*=( )

```
HistogramData & InternalZGY::HistogramData::operator*= (
    count_type factor )
```

Multiply [HistogramBuilder](#) with a constant N, equivalent to creating a new instance and adding the old one to it N times. N can be negative.

### 14.28.3.8 operator+=( )

```
HistogramData & InternalZGY::HistogramData::operator+= (
    const HistogramData & other )
```

Add the samples found in another histogram, just as if the samples had been added one at a time using Add().

### 14.28.3.9 operator-=( )

```
HistogramData & InternalZGY::HistogramData::operator-= (
    const HistogramData & other )
```

Subtract the samples found in another histogram, more or less undoing the effect of add. The min/max range in the statistics might be left showing a too wide range.

### 14.28.3.10 operator=( )

```
HistogramData & InternalZGY::HistogramData::operator= (
    const HistogramData & other )
```

Standard assignment operator. Needs to be explicit because of the allocated data for bins.

### 14.28.3.11 operator==( )

```
bool InternalZGY::HistogramData::operator== (
    const HistogramData & other ) const
```

Two histograms are considered equal if they return the same bin count for any input value. In practice there is some slop, as only the center of each bin of either histogram is checked. The statistics information need not match. Nor is there any check that the histograms have the same range or even the same number of bins.

### 14.28.3.12 scale( )

```
void InternalZGY::HistogramData::scale (
    double oldmin,
    double oldmax,
    double newmin,
    double newmax )
```

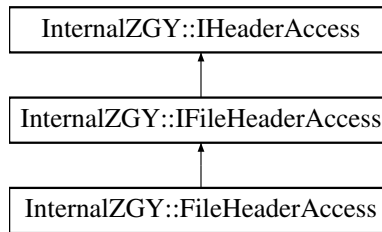
Calculate the linear transform needed to convert from one range (typically the natural data range of the integral storage type) to the data range that the application wants to see. Then update the histogram and associated statistics so they look like the transform had been done on every single data point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

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

- impl/histogramdata.h
- impl/histogramdata.cpp

## 14.29 InternalZGY::IFileHeaderAccess Class Reference

Inheritance diagram for InternalZGY::IFileHeaderAccess:



### Public Member Functions

- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset)=0
- virtual void **byteswap** ()=0
- virtual std::array< std::uint8\_t, 4 > **magic** () const =0
- virtual std::uint32\_t **version** () const =0

### Additional Inherited Members

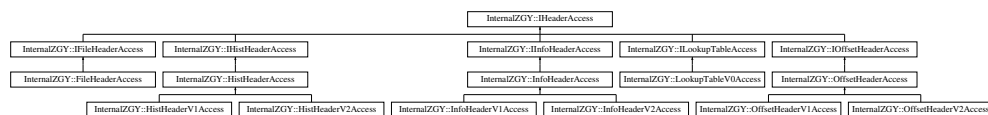
The documentation for this class was generated from the following file:

- impl/[meta.h](#)

## 14.30 InternalZGY::IHeaderAccess Class Reference

```
#include <meta.h>
```

Inheritance diagram for InternalZGY::IHeaderAccess:



### Public Types

- typedef std::vector< std::uint8\_t > **podbytes\_t**

### Public Member Functions

- virtual podbytes\_t **podbytes** () const =0
- virtual void **dump** (std::ostream &out, const std::string &prefix="")=0

## 14.30.1 Detailed Description

Currently I am having the access function for `foo[N]` return `std::array` instead of a raw pointer. This is inefficient but simplifies the case where the result is a derived value. I might change my mind about this.

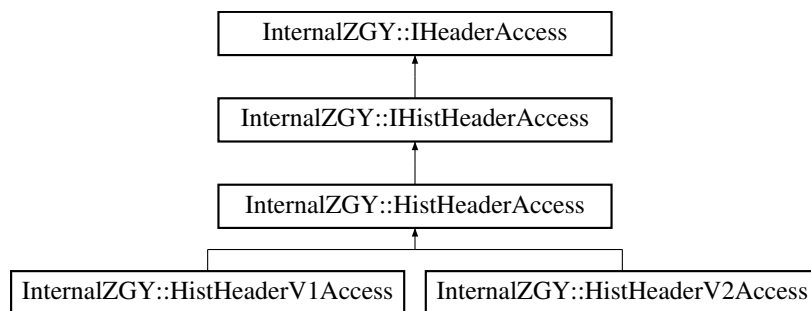
The `ptr_to_array` method allows writing `return ptr_to_array<float,3>(_foo);` instead of `return std::array<float,3>{_foo[0], _foo[1], _foo[2]};` and also tries to handle the case when `_foo` is misaligned.

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

- [impl/meta.h](#)

## 14.31 InternalZGY::IHistHeaderAccess Class Reference

Inheritance diagram for InternalZGY::IHistHeaderAccess:



## Public Member Functions

- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset, std::int64\_t size)=0
- virtual void **byteswap** ()=0
- virtual void **calculate** ()=0
- virtual std::int64\_t **bincount** () const =0
- virtual std::int64\_t **samplecount** () const =0
- virtual double **minvalue** () const =0
- virtual double **maxvalue** () const =0
- virtual const std::int64\_t \* **bins** () const =0
- virtual void **sethisto** (double minvalue, double maxvalue, const std::int64\_t \*bins, std::int64\_t bincount)=0

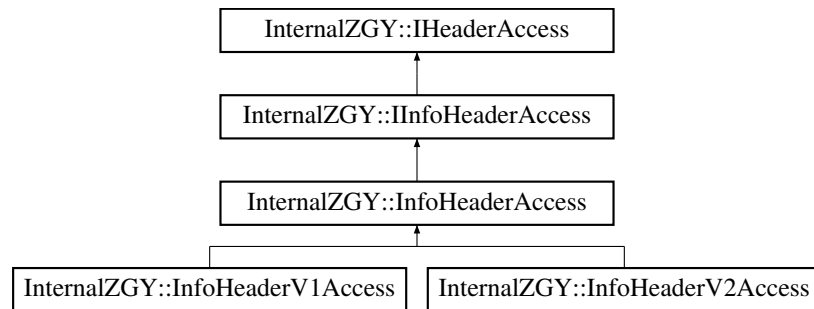
## Additional Inherited Members

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

- [impl/meta.h](#)

## 14.32 InternalZGY::IInfoHeaderAccess Class Reference

Inheritance diagram for InternalZGY::IInfoHeaderAccess:



### Public Member Functions

- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset, std::int64\_t size)=0
- virtual void **byteswap** ()=0
- virtual void **calculate\_cache** ()=0
- virtual void **calculate\_read** (const podbytes\_t &slbuf, const std::shared\_ptr< [IHistHeaderAccess](#) > &hh)=0
- virtual podbytes\_t **calculate\_write** ()=0
- virtual std::array< std::int64\_t, 3 > **bricksize** () const =0
- virtual std::array< float, 2 > **codingrange** () const =0
- virtual std::array< std::uint8\_t, 16 > **dataid** () const =0
- virtual std::array< std::uint8\_t, 16 > **verid** () const =0
- virtual std::array< std::uint8\_t, 16 > **previd** () const =0
- virtual std::string **srcname** () const =0
- virtual std::string **srcdesc** () const =0
- virtual [RawDataType](#) **srctype** () const =0
- virtual std::array< std::int64\_t, 3 > **size** () const =0
- virtual std::array< float, 3 > **orig** () const =0
- virtual std::array< float, 3 > **inc** () const =0
- virtual std::int64\_t **scnt** () const =0
- virtual double **ssum** () const =0
- virtual double **sssq** () const =0
- virtual float **smin** () const =0
- virtual float **smax** () const =0
- virtual std::array< float, 4 > **gpiline** () const =0
- virtual std::array< float, 4 > **gpxline** () const =0
- virtual std::array< double, 4 > **gpx** () const =0
- virtual std::array< double, 4 > **gpy** () const =0
- virtual [RawDataType](#) **datatype** () const =0
- virtual std::string **hprjsys** () const =0
- virtual [RawHorizontalDimension](#) **hdim** () const =0
- virtual double **hunitfactor** () const =0
- virtual std::string **hunitname** () const =0
- virtual [RawVerticalDimension](#) **vdim** () const =0
- virtual double **vunitfactor** () const =0
- virtual std::string **vunitname** () const =0
- virtual std::uint32\_t **slbufsize** () const =0
- virtual const std::array< std::array< double, 2 >, 4 > & **ocp\_index** () const =0
- virtual const std::array< std::array< double, 2 >, 4 > & **ocp\_annot** () const =0



- virtual const std::array< std::array< double, 2 >, 4 > & **ocp\_world** () const =0
- virtual std::int32\_t **nlods** () const =0
- virtual const std::vector< std::array< std::int64\_t, 3 > > & **lodsizes** () const =0
- virtual const std::vector< std::int64\_t > & **alphaoffsets** () const =0
- virtual const std::vector< std::int64\_t > & **brickoffsets** () const =0
- virtual std::int64\_t **bytesperalpha** () const =0
- virtual std::int64\_t **bytesperbrick** () const =0
- virtual std::int64\_t **bytespersample** () const =0
- virtual std::array< double, 2 > **storagetofloat** () const =0
- virtual double **storagetofloat\_slope** () const =0
- virtual double **storagetofloat\_intercept** () const =0
- virtual double **defaultstorage** () const =0
- virtual double **defaultvalue** () const =0
- virtual void **setstats** (std::int64\_t scnt, double ssum, double sssq, float smin, float smax)=0

### Additional Inherited Members

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

- impl/[meta.h](#)

## 14.33 InternalZGY::IJK Struct Reference

### Public Member Functions

- std::ostream & **dump** (std::ostream &os) const

### Public Attributes

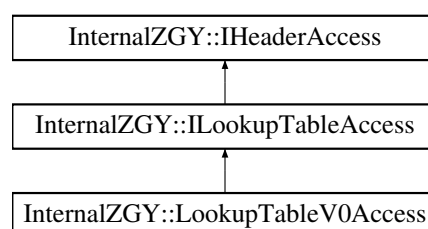
- std::int64\_t **i0**
- std::int64\_t **j0**
- std::int64\_t **k0**
- std::int64\_t **ni**
- std::int64\_t **nj**
- std::int64\_t **nk**

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

- impl/[bulk.cpp](#)

## 14.34 InternalZGY::ILookupTableAccess Class Reference

Inheritance diagram for InternalZGY::ILookupTableAccess:



## Public Member Functions

- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset, std::int64\_t size)=0
- virtual void **byteswap** ()=0
- virtual std::uint64\_t **lookupLinearIndex** (std::int64\_t index) const =0
- virtual std::vector< std::uint64\_t > & **lup** ()=0
- virtual std::vector< std::uint64\_t > & **lupend** ()=0
- virtual const std::vector< std::uint64\_t > & **lup** () const =0
- virtual const std::vector< std::uint64\_t > & **lupend** () const =0

## Additional Inherited Members

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

- impl/[meta.h](#)

## 14.35 InternalZGY::SummaryTimer::Impl Class Reference

### Public Member Functions

- **Impl** (const char \*name)

### Public Attributes

- long long **frequency\_**
- std::atomic< int > **count\_**
- std::atomic< long long > **total\_**
- std::atomic< long long > **last\_**
- char **name\_** [256]
- char **buff\_** [256]

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

- impl/timer.cpp

## 14.36 InternalZGY::ImplicitLinearTransform2d Class Reference

```
#include <iltf2d.h>
```

### Classes

- struct [TiePoint](#)

### Public Types

- typedef double **value\_type**
- typedef [ImplicitLinearTransform2d](#) **this\_type**

### Public Member Functions

- [ImplicitLinearTransform2d](#) ()
- [ImplicitLinearTransform2d](#) (const [TiePoint](#) &pt0, const [TiePoint](#) &pt1, const [TiePoint](#) &pt2)
- bool [operator==](#) (const [this\\_type](#) &other) const
- bool [operator!=](#) (const [this\\_type](#) &other) const
- bool [Compare](#) (const [this\\_type](#) &other, const value\_type &errtol=0) const
- [this\\_type](#) & [Push](#) (const [this\\_type](#) &other)
- [this\\_type](#) & [Translate](#) (value\_type t0, value\_type t1)
- [this\\_type](#) & [Rotate](#) (value\_type r)
- [this\\_type](#) & [Scale](#) (value\_type s0, value\_type s1)
- void [Apply](#) (value\_type \*b, const value\_type \*a) const
- void [operator\(\)](#) (value\_type \*b, const value\_type \*a) const

### Static Public Member Functions

- static [TiePoint](#) [makeTiePoint](#) ()
- static [TiePoint](#) [makeTiePoint](#) (value\_type a0, value\_type a1, value\_type b0, value\_type b1)
- static [this\\_type](#) [Identity](#) ()

### 14.36.1 Detailed Description

This class represents a linear transform  $T: \mathbb{R}^2 \rightarrow \mathbb{R}^2$  from 2-dimensional Euclidean space A to 2-dimensional Euclidean space B, defined implicitly by three given tie points.

### 14.36.2 Member Typedef Documentation

#### 14.36.2.1 this\_type

```
typedef ImplicitLinearTransform2d InternalZGY::ImplicitLinearTransform2d::this\_type
```

Type of values processed by this class.

### 14.36.3 Constructor & Destructor Documentation

## 14.36.3.1 ImplicitLinearTransform2d() [1/2]

```
InternalZGY::ImplicitLinearTransform2d::ImplicitLinearTransform2d ( )
```

Default constructor. Produces 1-to-1 mapping.

## 14.36.3.2 ImplicitLinearTransform2d() [2/2]

```
InternalZGY::ImplicitLinearTransform2d::ImplicitLinearTransform2d (
    const TiePoint & pt0,
    const TiePoint & pt1,
    const TiePoint & pt2 )
```

Construct from three tie-points. Fails by throwing an exception if the tie-points don't span a 2-dimensional space (i.e. lie along a straight line in either space).

### Parameters

<i>pt0</i>	First tie-point.
<i>pt1</i>	Second tie-point.
<i>pt2</i>	Third tie-point.

## 14.36.4 Member Function Documentation

### 14.36.4.1 Apply()

```
void InternalZGY::ImplicitLinearTransform2d::Apply (
    value_type * b,
    const value_type * a ) const
```

Convert from space A to space B.

### Parameters

<i>b</i>	Pointer to buffer holding (at least) two elements of value_type which will receive the transformed coordinate in B.
<i>a</i>	Pointer to buffer holding (at least) two elements of value_type serving as the A coordinate to transform.

### 14.36.4.2 Compare()

```
bool InternalZGY::ImplicitLinearTransform2d::Compare (
    const this_type & other,
    const value_type & errtol = 0 ) const
```

### Parameters

<i>other</i>	Instance to compare with.
<i>errtol</i>	Entry-by-entry absolute error tolerance.

### Returns

True if other is equal to self, otherwise false.

#### 14.36.4.3 Identity()

```
ImplicitLinearTransform2d InternalZGY::ImplicitLinearTransform2d::Identity ( ) [static]
```

### Returns

Identity (one-to-one) transform.

#### 14.36.4.4 makeTiePoint()

```
static TiePoint InternalZGY::ImplicitLinearTransform2d::makeTiePoint ( ) [static]
```

Partial workaround for problems on Windows with forcing instantiation of the above. Define member functions to create instances. Unfortunately this just solves part of the problem, since the compiler may generate calls to the default constructor.

#### 14.36.4.5 operator"!="()

```
bool InternalZGY::ImplicitLinearTransform2d::operator!= (
    const this_type & other ) const
```

### Parameters

<i>other</i>	Instance to compare with.
--------------	---------------------------

### Returns

False if other is equal to self, otherwise true.

#### 14.36.4.6 operator==( )

```
bool InternalZGY::ImplicitLinearTransform2d::operator== (
    const this_type & other ) const
```

## Parameters

<i>other</i>	Instance to compare with.
--------------	---------------------------

## Returns

True if *other* is equal to self, otherwise false.

### 14.36.4.7 Push()

```
ImplicitLinearTransform2d & InternalZGY::ImplicitLinearTransform2d::Push (  
    const this_type & other )
```

Push another transform onto this one, i.e. front-multiply it with this.

## Parameters

<i>other</i>	The other transform.
--------------	----------------------

## Returns

Reference to self.

### 14.36.4.8 Rotate()

```
ImplicitLinearTransform2d & InternalZGY::ImplicitLinearTransform2d::Rotate (  
    value_type r )
```

## Parameters

<i>r</i>	Rotation in radians counter-clockwise relative to first axis (polar rotation).
----------	--

## Returns

Reference to self.

### 14.36.4.9 Scale()

```
ImplicitLinearTransform2d & InternalZGY::ImplicitLinearTransform2d::Scale (  
    value_type s0,  
    value_type s1 )
```

### Parameters

<i>s0</i>	Scaling in first dimension.
<i>s1</i>	Scaling in second dimension.

### Returns

Reference to self.

#### 14.36.4.10 Translate()

```
ImplicitLinearTransform2d & InternalZGY::ImplicitLinearTransform2d::Translate (
    value_type t0,
    value_type t1 )
```

### Parameters

<i>t0</i>	Translation in first dimension.
<i>t1</i>	Translation in second dimension.

### Returns

Reference to self.

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

- [impl/iltf2d.h](#)
- [impl/iltf2d.cpp](#)

## 14.37 InternalZGY::ImplicitLinearTransform2dImp Class Reference

### Public Types

- typedef double **value\_type**

### Static Public Member Functions

- static void [Mult](#) (value\_type result[2][3], const value\_type left[2][3], const value\_type right[2][3])

#### 14.37.1 Detailed Description

This class is used in the implementation of [ImplicitLinearTransform2d](#).

## 14.37.2 Member Function Documentation

### 14.37.2.1 Mult()

```
void InternalZGY::ImplicitLinearTransform2dImp::Mult (
    value_type result[2][3],
    const value_type left[2][3],
    const value_type right[2][3] ) [static]
```

Multiply two 3x3 homogeneous matrices (i.e. 2x3 explicit representation)

#### Parameters

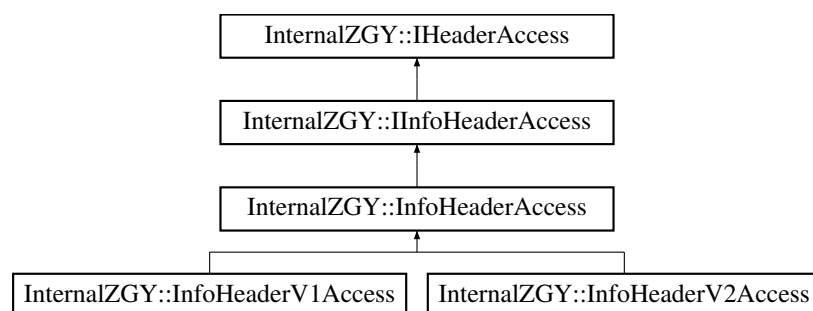
<i>result</i>	Matrix to write result to.
<i>left</i>	Matrix to serve as left-hand of multiplication.
<i>right</i>	Matrix to serve as right-hand of multiplication.

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

- impl/iltf2d.cpp

## 14.38 InternalZGY::InfoHeaderAccess Class Reference

Inheritance diagram for InternalZGY::InfoHeaderAccess:



### Additional Inherited Members

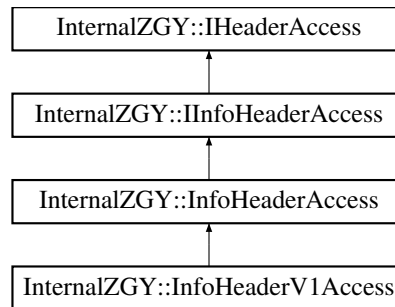
The documentation for this class was generated from the following file:

- impl/meta.cpp



## 14.39 InternalZGY::InfoHeaderV1Access Class Reference

Inheritance diagram for InternalZGY::InfoHeaderV1Access:



### Public Member Functions

- virtual podbytes\_t **podbytes** () const override
- virtual void **read** (const std::shared\_ptr< FileADT > &file, std::int64\_t offset, std::int64\_t size) override
- virtual void **byteswap** () override
- virtual void **calculate\_cache** () override
- virtual void **calculate\_read** (const podbytes\_t &slbuf, const std::shared\_ptr< IHistHeaderAccess > &hh) override
- virtual podbytes\_t **calculate\_write** () override
- virtual std::array< std::int64\_t, 3 > **size** () const override
- virtual std::array< float, 3 > **orig** () const override
- virtual std::array< float, 3 > **inc** () const override
- virtual std::array< float, 4 > **gpiline** () const override
- virtual std::array< float, 4 > **gpxline** () const override
- virtual std::array< double, 4 > **gpx** () const override
- virtual std::array< double, 4 > **gpy** () const override
- virtual RawDataType **datatype** () const override
- virtual std::array< std::int64\_t, 3 > **bricksize** () const override
- virtual std::array< float, 2 > **codingrange** () const override
- virtual std::array< std::uint8\_t, 16 > **dataid** () const override
- virtual std::array< std::uint8\_t, 16 > **verid** () const override
- virtual std::array< std::uint8\_t, 16 > **previd** () const override
- virtual std::string **srcname** () const override
- virtual std::string **srcdesc** () const override
- virtual RawDataType **srctype** () const override
- virtual std::int64\_t **scnt** () const override
- virtual double **ssum** () const override
- virtual double **sssq** () const override
- virtual float **smin** () const override
- virtual float **smax** () const override
- virtual std::string **hprjsys** () const override
- virtual RawHorizontalDimension **hdim** () const override
- virtual double **hunitfactor** () const override
- virtual std::string **hunitname** () const override
- virtual RawVerticalDimension **vdim** () const override
- virtual double **vunitfactor** () const override
- virtual std::string **vunitname** () const override
- virtual std::uint32\_t **slbufsize** () const override

- virtual const std::array< std::array< double, 2 >, 4 > & **ocp\_index** () const override
- virtual const std::array< std::array< double, 2 >, 4 > & **ocp\_annot** () const override
- virtual const std::array< std::array< double, 2 >, 4 > & **ocp\_world** () const override
- virtual const std::vector< std::array< std::int64\_t, 3 > > & **lodsizes** () const override
- virtual std::int32\_t **nlods** () const override
- virtual const std::vector< std::int64\_t > & **alphaoffsets** () const override
- virtual const std::vector< std::int64\_t > & **brickoffsets** () const override
- virtual void **setstats** (std::int64\_t scnt, double ssum, double sssq, float smin, float smax)

## Public Attributes

- [InfoHeaderV1POD](#) \_pod
- float \_cached\_sample\_min
- float \_cached\_sample\_max
- std::int32\_t \_cached\_nlods
- std::vector< std::array< std::int64\_t, 3 > > \_cached\_lodsizes
- std::vector< std::int64\_t > \_cached\_alphaoffsets
- std::vector< std::int64\_t > \_cached\_brickoffsets
- std::array< std::array< double, 2 >, 4 > \_cached\_index
- std::array< std::array< double, 2 >, 4 > \_cached\_annot
- std::array< std::array< double, 2 >, 4 > \_cached\_world

## Additional Inherited Members

### 14.39.1 Member Function Documentation

#### 14.39.1.1 calculate\_cache()

```
void InternalZGY::InfoHeaderV1Access::calculate_cache ( ) [override], [virtual]
```

Calculate derived information that is too expensive to compute on the fly. The code here is needed both after reading an existing file and after creating a new one.

Implements [InternalZGY::InfoHeaderAccess](#).

#### 14.39.1.2 calculate\_read()

```
void InternalZGY::InfoHeaderV1Access::calculate_read (
    const podbytes_t & slbuf,
    const std::shared_ptr< IHistHeaderAccess > & hh ) [override], [virtual]
```

Fix up information that logically belong in this header but is stored elsewhere. This is not quite the same as [calculate\\_cache\(\)](#) and it should only be called after reading an existing fille.

Implements [InternalZGY::InfoHeaderAccess](#).

### 14.39.1.3 calculate\_write()

```
InfoHeaderV1Access::podbytes_t InternalZGY::InfoHeaderV1Access::calculate_write ( ) [override],  
[virtual]
```

Prepare to write this header to disk. If this is a newly created header then the cached derived information needs to be computed as well. It might be safer to do that unconditionally.

Implements [InternalZGY::IInfoHeaderAccess](#).

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

- impl/meta.cpp

## 14.40 InternalZGY::InfoHeaderV1POD Class Reference

Physical layout of Info Header version 1.

### Public Attributes

- std::int32\_t **\_size** [3]
- std::int32\_t **\_orig** [3]
- std::int32\_t **\_inc** [3]
- float **\_incfactor** [3]
- std::int32\_t **\_gpiline** [4]
- std::int32\_t **\_gpxline** [4]
- double **\_gpx** [4]
- double **\_gpy** [4]
- std::uint8\_t **\_datatype**
- std::uint8\_t **\_coordtype**

### 14.40.1 Detailed Description

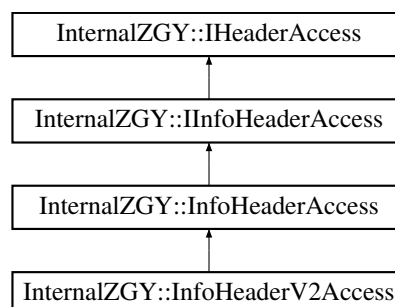
Physical layout of Info Header version 1.

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

- impl/meta.cpp

## 14.41 InternalZGY::InfoHeaderV2Access Class Reference

Inheritance diagram for InternalZGY::InfoHeaderV2Access:



## Public Member Functions

- virtual podbytes\_t **podbytes** () const override
- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset, std::int64\_t size) override
- virtual void **byteswap** () override
- virtual void [calculate\\_read](#) (const podbytes\_t &slbuf, const std::shared\_ptr< [IHistHeaderAccess](#) > &hh) override
- virtual void [calculate\\_cache](#) () override
- virtual podbytes\_t [calculate\\_write](#) () override
- virtual std::array< std::int64\_t, 3 > **bricksize** () const override
- virtual [RawDataType](#) **datatype** () const override
- virtual std::array< float, 2 > **codingrange** () const override
- virtual std::array< std::uint8\_t, 16 > **dataid** () const override
- virtual std::array< std::uint8\_t, 16 > **verid** () const override
- virtual std::array< std::uint8\_t, 16 > **previd** () const override
- virtual std::string **srcname** () const override
- virtual std::string **srcdesc** () const override
- virtual [RawDataType](#) **srctype** () const override
- virtual std::array< float, 3 > **orig** () const override
- virtual std::array< float, 3 > **inc** () const override
- virtual std::array< std::int64\_t, 3 > **size** () const override
- virtual std::int64\_t **scnt** () const override
- virtual double **ssum** () const override
- virtual double **sssq** () const override
- virtual float **smin** () const override
- virtual float **smax** () const override
- virtual std::array< float, 4 > **gpiline** () const override
- virtual std::array< float, 4 > **gpxline** () const override
- virtual std::array< double, 4 > **gpx** () const override
- virtual std::array< double, 4 > **gpy** () const override
- virtual std::string **hprjsys** () const override
- virtual [RawHorizontalDimension](#) **hdim** () const override
- virtual double **hunitfactor** () const override
- virtual std::string **hunitname** () const override
- virtual [RawVerticalDimension](#) **vdim** () const override
- virtual double **vunitfactor** () const override
- virtual std::string **vunitname** () const override
- virtual std::uint32\_t **slbufsize** () const override
- virtual const std::array< std::array< double, 2 >, 4 > & **ocp\_index** () const override
- virtual const std::array< std::array< double, 2 >, 4 > & **ocp\_annot** () const override
- virtual const std::array< std::array< double, 2 >, 4 > & **ocp\_world** () const override
- virtual std::int32\_t **nlods** () const override
- virtual const std::vector< std::array< std::int64\_t, 3 > > & **lodsizes** () const override
- virtual const std::vector< std::int64\_t > & **alphaoffsets** () const override
- virtual const std::vector< std::int64\_t > & **brickoffsets** () const override
- virtual void **setstats** (std::int64\_t scnt, double ssum, double sssq, float smin, float smax)

### Public Attributes

- [InfoHeaderV2POD\\_pod](#)
- std::string **\_srcname**
- std::string **\_srcdesc**
- std::string **\_hprjsys**
- std::string **\_hunitname**
- std::string **\_vunitname**
- std::int32\_t **\_cached\_nlods**
- std::vector< std::array< std::int64\_t, 3 > > **\_cached\_lodsizes**
- std::vector< std::int64\_t > **\_cached\_alphaoffsets**
- std::vector< std::int64\_t > **\_cached\_brickoffsets**
- std::array< std::array< double, 2 >, 4 > **\_cached\_index**
- std::array< std::array< double, 2 >, 4 > **\_cached\_annot**
- std::array< std::array< double, 2 >, 4 > **\_cached\_world**

### Additional Inherited Members

#### 14.41.1 Member Function Documentation

##### 14.41.1.1 calculate\_cache()

```
void InternalZGY::InfoHeaderV2Access::calculate_cache ( ) [override], [virtual]
```

Calculate derived information that is too expensive to compute on the fly. The code here is needed both after reading an existing file and after creating a new one.

Implements [InternalZGY::InfoHeaderAccess](#).

##### 14.41.1.2 calculate\_read()

```
void InternalZGY::InfoHeaderV2Access::calculate_read (
    const podbytes_t & stringlist_in,
    const std::shared_ptr< IHistHeaderAccess > & hh ) [override], [virtual]
```

Fix up information that logically belong in this header but is stored elsewhere. This is not quite the same as [calculate\\_cache\(\)](#) and it should only be called after reading an existing fille.

Implements [InternalZGY::InfoHeaderAccess](#).

### 14.41.1.3 calculate\_write()

```
InfoHeaderV2Access::podbytes_t InternalZGY::InfoHeaderV2Access::calculate_write ( ) [override],  
[virtual]
```

Prepare for writing out this info header. 1) Pack the 5 strings into a string header, which isn't a separate type. 2) Record the size of the string header inside this info header. 3) Return the bytes of the string header.

If this is a newly created header then the cached derived information needs to be computed as well. It might be safer to do that unconditionally.

TODO-WARNING: If I at some point add a method to change the corner coordinates of an existing file then either that function is responsible for updating both the pod and the derved corners or this function needs to compute one from the other. Depending on which was actually changed.

Implements [InternalZGY::InfoHeaderAccess](#).

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

- impl/meta.cpp

## 14.42 InternalZGY::InfoHeaderV2POD Class Reference

Physical layout of Info Header version 2 and 3.

### Public Attributes

- std::int32\_t **\_bricksize** [3]
- std::uint8\_t **\_datatype**
- float **\_codingrange** [2]
- std::uint8\_t **\_dataid** [16]
- std::uint8\_t **\_verid** [16]
- std::uint8\_t **\_previd** [16]
- std::uint8\_t **\_srctype**
- float **\_orig** [3]
- float **\_inc** [3]
- std::int32\_t **\_size** [3]
- std::int32\_t **\_curorig** [3]
- std::int32\_t **\_cursize** [3]
- std::int64\_t **\_scnt**
- double **\_ssum**
- double **\_sssq**
- float **\_smin**
- float **\_smax**
- float **\_srvorig** [3]
- float **\_srvsize** [3]
- std::uint8\_t **\_gdef**
- double **\_gazim** [2]
- double **\_gbinsz** [2]
- float **\_gpiline** [4]
- float **\_gpxline** [4]
- double **\_gpx** [4]
- double **\_gpy** [4]
- std::uint8\_t **\_hdim**
- double **\_hunitfactor**
- std::uint8\_t **\_vdim**
- double **\_vunitfactor**
- std::uint32\_t **\_slbufsize**

### 14.42.1 Detailed Description

Physical layout of Info Header version 2 and 3.

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

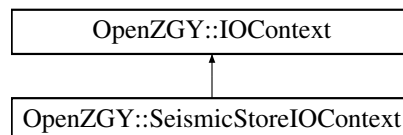
- impl/meta.cpp

## 14.43 OpenZGY::IOContext Class Reference

Base class for backend specific context.

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

Inheritance diagram for OpenZGY::IOContext:



### Public Member Functions

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

### 14.43.1 Detailed Description

Base class for backend specific context.

### 14.43.2 Constructor & Destructor Documentation

#### 14.43.2.1 ~IOContext()

```
OpenZGY::IOContext::~~IOContext ( ) [virtual]
```

The class needs at least one virtual member to make `dynamic_cast` work.

### 14.43.3 Member Function Documentation

### 14.43.3.1 toString()

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

Display the context in a human readable format for debugging.

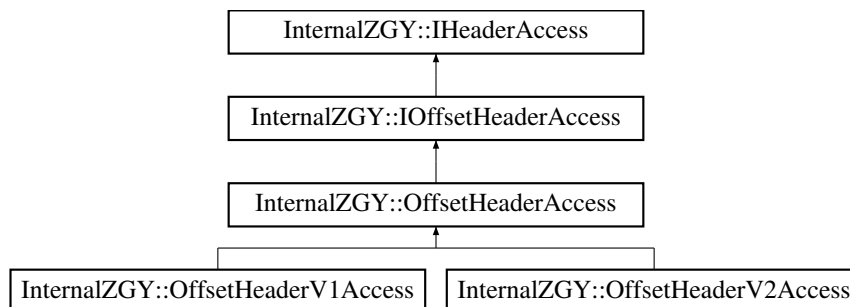
Implemented in [OpenZGY::SeismicStoreIOContext](#).

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

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

## 14.44 InternalZGY::IOffsetHeaderAccess Class Reference

Inheritance diagram for InternalZGY::IOffsetHeaderAccess:



### Public Member Functions

- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset)=0
- virtual void **byteswap** ()=0
- virtual void **calculate** (const std::shared\_ptr< [IInfoHeaderAccess](#) > &ih)=0
- virtual std::int64\_t **infoff** () const =0
- virtual std::int64\_t **stroff** () const =0
- virtual std::int64\_t **alphalupoff** () const =0
- virtual std::int64\_t **bricklupoff** () const =0
- virtual std::int64\_t **histoff** () const =0
- virtual std::int64\_t **infsize** () const =0
- virtual std::int64\_t **histsize** () const =0
- virtual std::int64\_t **alphalupsize** () const =0
- virtual std::int64\_t **bricklupsize** () const =0

### Additional Inherited Members

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

- [impl/meta.h](#)

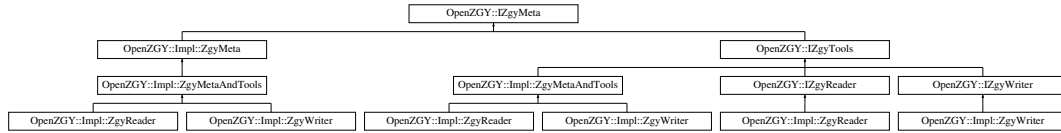


## 14.45 OpenZGY::IZgyMeta Class Reference

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 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 [bricksiz](#) () 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.*

## 14.45.1 Detailed Description

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

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

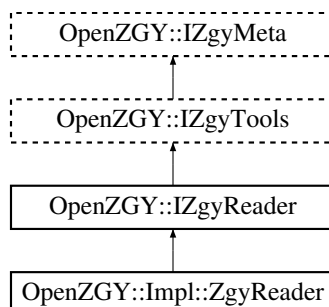
- [api.h](#)
- [api.cpp](#)

## 14.46 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

### 14.46.1 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.

### 14.46.2 Member Function Documentation

#### 14.46.2.1 [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](#).

## 14.46.2.2 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.

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().

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

## 14.46.2.3 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](#).

## 14.46.2.4 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](#).

## 14.46.2.5 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:

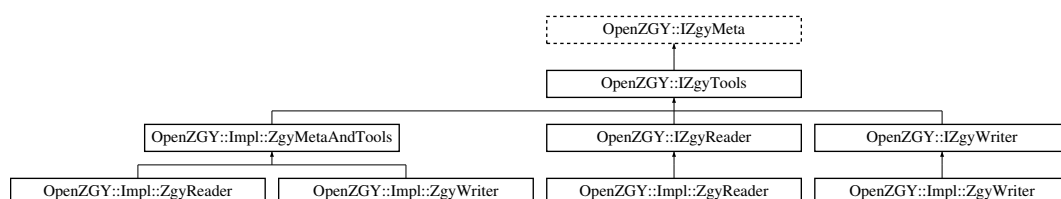
- [api.h](#)
- [api.cpp](#)

## 14.47 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

### 14.47.1 Detailed Description

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

### 14.47.2 Member Function Documentation

#### 14.47.2.1 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](#).

## 14.47.2.2 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:

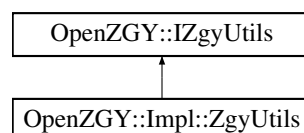
- [api.h](#)
- [api.cpp](#)

## 14.48 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](#).*

### 14.48.1 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.

### 14.48.2 Member Function Documentation

#### 14.48.2.1 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](#).

#### 14.48.2.2 utils()

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

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

#### Parameters

<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:

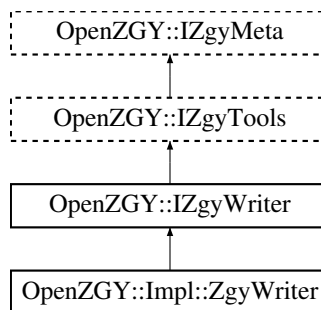
- [api.h](#)
- [api.cpp](#)

## 14.49 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\\_incomplete](#) ()=0  
*Flush the file to disk and close it.*
- virtual void [close](#) ()=0  
*Flush the file to disk and close it.*
- virtual bool [errorflag](#) () const =0
- virtual void [set\\_errorflag](#) (bool)=0

## 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

### 14.49.1 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.

### 14.49.2 Member Function Documentation

#### 14.49.2.1 [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](#).

### 14.49.2.2 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](#).

### 14.49.2.3 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. 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.

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

### 14.49.2.4 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](#).

#### 14.49.2.5 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](#).

#### 14.49.2.6 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](#).

### 14.49.2.7 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](#).

### 14.49.2.8 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](#).

### 14.49.2.9 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.

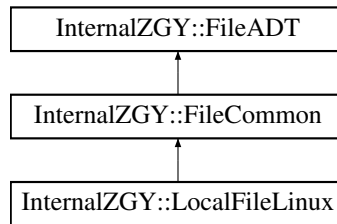
Implemented in [OpenZGY::Impl::ZgyWriter](#).

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

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

## 14.50 InternalZGY::LocalFileLinux Class Reference

Inheritance diagram for InternalZGY::LocalFileLinux:



### Public Member Functions

- **LocalFileLinux** (const std::string &filename, OpenMode mode, const [OpenZGY::IOContext](#) \*iocontext)
- virtual void [xx\\_close](#) () override
- virtual std::int64\_t [xx\\_eof](#) () const override
- virtual bool [xx\\_iscloud](#) () const override
- virtual void [xx\\_read](#) (void \*data, std::int64\_t offset, std::int64\_t size, UsageHint usagehint=UsageHint::↔ Unknown) override
- virtual void [xx\\_readv](#) (const ReadList &requests, bool parallel\_ok=false, bool immutable\_ok=false, bool transient\_ok=false, UsageHint usagehint=UsageHint::Unknown) override
- virtual void [xx\\_write](#) (const void \*data, std::int64\_t offset, std::int64\_t size, UsageHint usagehint=UsageHint::↔ Unknown) override
- virtual bool [xx\\_threadsafe](#) () const override
- virtual std::int64\_t [\\_real\\_eof](#) () const

*Get the current file size for error reporting.*

### Static Public Member Functions

- static std::shared\_ptr< [FileADT](#) > **xx\_make\_instance** (const std::string &filename, OpenMode mode, const [OpenZGY::IOContext](#) \*iocontext)

### Additional Inherited Members

#### 14.50.1 Member Function Documentation

##### 14.50.1.1 \_real\_eof()

```
std::int64_t InternalZGY::LocalFileLinux::_real_eof ( ) const [virtual]
```

Get the current file size for error reporting.

The default implementation in the base class just assumes that the [xx\\_eof\(\)](#) that is (probably) maintained internally is correct.

Reimplemented from [InternalZGY::FileCommon](#).

### 14.50.1.2 `xx_close()`

```
void InternalZGY::LocalFileLinux::xx_close ( ) [override], [virtual]
```

Close the file. This should always be done explicitly instead of assuming the destructor will handle it. If left to the destructor then any exceptions will be swallowed.

Implements [InternalZGY::FileADT](#).

### 14.50.1.3 `xx_eof()`

```
std::int64_t InternalZGY::LocalFileLinux::xx_eof ( ) const [override], [virtual]
```

Return the current end-of-file, i.e. the file size.

Implements [InternalZGY::FileADT](#).

### 14.50.1.4 `xx_iscloud()`

```
bool InternalZGY::LocalFileLinux::xx_iscloud ( ) const [override], [virtual]
```

Return true if the file is on the cloud. This might trigger some optimizations.

Implements [InternalZGY::FileADT](#).

### 14.50.1.5 `xx_read()`

```
void InternalZGY::LocalFileLinux::xx_read (
    void * data,
    std::int64_t offset,
    std::int64_t size,
    UsageHint usagehint = UsageHint::Unknown ) [override], [virtual]
```

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

Implements [InternalZGY::FileADT](#).

## 14.50.1.6 `xx_readv()`

```
void InternalZGY::LocalFileLinux::xx_readv (
    const ReadList & requests,
    bool parallel_ok = false,
    bool immutable_ok = false,
    bool transient_ok = false,
    UsageHint usagehint = UsageHint::Unknown ) [override], [virtual]
```

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

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

The delivery functor is called as `fn(void* data, std::int64_t size)`

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

Implements [InternalZGY::FileADT](#).

## 14.50.1.7 `xx_threadsafe()`

```
bool InternalZGY::LocalFileLinux::xx_threadsafe ( ) const [override], [virtual]
```

Return true if multiple reads can be in progress at the same time.

Implements [InternalZGY::FileADT](#).

## 14.50.1.8 `xx_write()`

```
void InternalZGY::LocalFileLinux::xx_write (
    const void * data,
    std::int64_t offset,
    std::int64_t size,
    UsageHint usagehint = UsageHint::Unknown ) [override], [virtual]
```

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

Implements [InternalZGY::FileADT](#).

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

- [impl/file\\_local.cpp](#)



## 14.51 InternalZGY::LodAllZero< T > Class Template Reference

Set the low resolution data to all zeros.

### Public Member Functions

- **T operator()** (T, T, T, T, T, T, T, T, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::AllZero](#)

### 14.51.1 Detailed Description

```
template<typename T>
class InternalZGY::LodAllZero< T >
```

Set the low resolution data to all zeros.

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

- impl/lodalgo.cpp

## 14.52 InternalZGY::LodAverage< T > Class Template Reference

Arithmetic average of 8 neighboring integer samples.

### Public Member Functions

- **T operator()** (T s0, T s1, T s2, T s3, T s4, T s5, T s6, T s7, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::Average](#)

### 14.52.1 Detailed Description

```
template<typename T>
class InternalZGY::LodAverage< T >
```

Arithmetic average of 8 neighboring integer samples.

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

- impl/lodalgo.cpp

## 14.53 InternalZGY::LodAverage< float > Class Reference

Arithmetic average of 8 neighboring float samples, ignoring NaN.

### Public Member Functions

- float **operator()** (float s0, float s1, float s2, float s3, float s4, float s5, float s6, float s7, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::Average](#)

### 14.53.1 Detailed Description

Arithmetic average of 8 neighboring float samples, ignoring NaN.

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

- impl/lodalgo.cpp

## 14.54 InternalZGY::LodAverageNon0< T > Class Template Reference

Arithmetic average, excluding zero, of integer data.

### Public Member Functions

- T **operator()** (T s0, T s1, T s2, T s3, T s4, T s5, T s6, T s7, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::AverageNon0](#)

### 14.54.1 Detailed Description

```
template<typename T>
class InternalZGY::LodAverageNon0< T >
```

Arithmetic average, excluding zero, of integer data.

This is not useful for seismic data. It might be handy if samples are by nature integer values, not just floating point numbers (such as seismic amplitude) crammed into a too-small integer.

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

- impl/lodalgo.cpp

### 14.55 InternalZGY::LodAverageNon0< float > Class Reference

Arithmetic average, excluding NaN and zero, of float data.

#### Public Member Functions

- float **operator()** (float s0, float s1, float s2, float s3, float s4, float s5, float s6, float s7, int, int, int)

#### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::AverageNon0](#)

#### 14.55.1 Detailed Description

Arithmetic average, excluding NaN and zero, of float data.

CAVEAT: This is not useful for seismic data. It might be handy if samples are by nature integer values, not just floating point numbers (such as seismic amplitude) crammed into a too-small integer. This means that running the algorithm on float data is probably a mistake.

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

- impl/lodalgo.cpp

### 14.56 InternalZGY::LodDecimate< T > Class Template Reference

Use just one of the 8 input values.

#### Public Member Functions

- T **operator()** (T s0, T, T, T, T, T, T, T, int, int, int)

#### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::Decimate](#)

#### 14.56.1 Detailed Description

```
template<typename T>
class InternalZGY::LodDecimate< T >
```

Use just one of the 8 input values.

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

- impl/lodalgo.cpp

## 14.57 InternalZGY::LodDecimateSkipNaN< T > Class Template Reference

Use just one of the 8 integral input values.

### Public Member Functions

- **T operator()** (T s0, T, T, T, T, T, T, T, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::DecimateSkipNaN](#)

### 14.57.1 Detailed Description

```
template<typename T>
class InternalZGY::LodDecimateSkipNaN< T >
```

Use just one of the 8 integral input values.

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

- impl/lodalgo.cpp

## 14.58 InternalZGY::LodDecimateSkipNaN< float > Class Reference

Use just one of the 8 float input values, looking for one not NaN.

### Public Member Functions

- **float operator()** (float s0, float s1, float s2, float s3, float s4, float s5, float s6, float s7, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::DecimateSkipNaN](#)

### 14.58.1 Detailed Description

Use just one of the 8 float input values, looking for one not NaN.

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

- impl/lodalgo.cpp

## 14.59 InternalZGY::LodMaximum< T > Class Template Reference

Maximum of 8 neighboring samples, excluding NaN.

### Public Member Functions

- **T operator()** (T s0, T s1, T s2, T s3, T s4, T s5, T s6, T s7, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::Maximum](#)

#### 14.59.1 Detailed Description

```
template<typename T>
class InternalZGY::LodMaximum< T >
```

Maximum of 8 neighboring samples, excluding NaN.

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

- impl/lodalgo.cpp

## 14.60 InternalZGY::LodMedian< T > Class Template Reference

Median of 8 neighboring integer samples.

### Public Member Functions

- **T operator()** (T s0, T s1, T s2, T s3, T s4, T s5, T s6, T s7, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::Median](#)

#### 14.60.1 Detailed Description

```
template<typename T>
class InternalZGY::LodMedian< T >
```

Median of 8 neighboring integer samples.

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

- impl/lodalgo.cpp

## 14.61 InternalZGY::LodMedian< float > Class Reference

Median of 8 neighboring float samples, ignoring NaN.

### Public Member Functions

- float **operator()** (float s0, float s1, float s2, float s3, float s4, float s5, float s6, float s7, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::Median](#)

### 14.61.1 Detailed Description

Median of 8 neighboring float samples, ignoring NaN.

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

- impl/lodalgo.cpp

## 14.62 InternalZGY::LodMinimum< T > Class Template Reference

Minimum of 8 neighboring samples, excluding NaN.

### Public Member Functions

- T **operator()** (T s0, T s1, T s2, T s3, T s4, T s5, T s6, T s7, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::Minimum](#)

### 14.62.1 Detailed Description

```
template<typename T>
class InternalZGY::LodMinimum< T >
```

Minimum of 8 neighboring samples, excluding NaN.

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

- impl/lodalgo.cpp

## 14.63 InternalZGY::LodMinMax< T > Class Template Reference

Weird algorithm, probably not useful.

### Public Member Functions

- **T operator()** (T s0, T s1, T s2, T s3, T s4, T s5, T s6, T s7, int i, int j, int k)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::MinMax](#)

#### 14.63.1 Detailed Description

```
template<typename T>
class InternalZGY::LodMinMax< T >
```

Weird algorithm, probably not useful.

Return either the minimum or the maximum value in a checkerboard pattern.

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

- impl/lodalgo.cpp

## 14.64 InternalZGY::LodMostFrequent< T, SkipNaN, SkipZero > Class Template Reference

Most frequent value.

### Public Member Functions

- **T operator()** (T s0, T s1, T s2, T s3, T s4, T s5, T s6, T s7, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::DecimateSkipNaN](#)

## 14.64.1 Detailed Description

```
template<typename T, bool SkipNaN, bool SkipZero>
class InternalZGY::LodMostFrequent< T, SkipNaN, SkipZero >
```

Most frequent value.

Return the value that occurs most frequently in the input. If there is a tie, return the first one. NaN or 0 are returned if and only if all inputs have this value. If the input is a mix of 0 and NaN only, the result is 0.

Since there are tests for equality, this algorithm probably only makes sense for integral data.

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

- impl/lodalgo.cpp

## 14.65 InternalZGY::LodSampling Class Reference

Static methods for downsampling. Used by lodalgo.cpp only.

```
#include <lodsampling.h>
```

### Static Public Member Functions

- static void [downSample1D](#) (double \*dst, int sizeDst, const double \*src, int sizeSrc)

### 14.65.1 Detailed Description

Static methods for downsampling. Used by lodalgo.cpp only.

### 14.65.2 Member Function Documentation

#### 14.65.2.1 downSample1D()

```
void InternalZGY::LodSampling::downSample1D (
    double * dst,
    int sizeDst,
    const double * src,
    int sizeSrc ) [static]
```

Downsample a vector with a factor of 2. The vector is low pass filtered and then down sampled.



## Parameters

out	dst	Out vector with downsampled result
in	sizeDst	Size of output vector. It must be sizeDst*2==sizeSrc
in	src	In vector
in	sizeSrc	Size of src vector. It must be sizeSrc==sizeDst*2 and >=5

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

- impl/lodsampling.h
- impl/lodsampling.cpp

## 14.66 InternalZGY::LodWeightedAverage< T > Class Template Reference

Weighted arithmetic average of 8 neighboring samples.

### Public Member Functions

- **LodWeightedAverage** (const std::int64\_t \*hist, int bins, double minHist, double maxHist)
- **T operator()** (T s0, T s1, T s2, T s3, T s4, T s5, T s6, T s7, int, int, int)

### Static Public Attributes

- static const [LodAlgorithm](#) **algorithm** = [LodAlgorithm::WeightedAverage](#)

### 14.66.1 Detailed Description

```
template<typename T>
class InternalZGY::LodWeightedAverage< T >
```

Weighted arithmetic average of 8 neighboring samples.

Each sample is weighted against how common this value is in the survey as a whole. Rare values (likely very high or very low) get higher priority. This prevents tiles from looking more and more washed out the more they get decimated.

TODO-Performance: Profile this function and try to speed it up. It is by far the most expensive of the decimators, ~10x the cost of LowPass. By default it is used for lod 2+ only. So it gets called with just 1/7 of the data passed to LowPass. In sum the code still spends more CPU cycles on this than on LowPass.

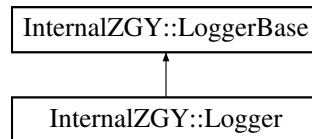
The documentation for this class was generated from the following file:

- impl/lodalgo.cpp

## 14.67 InternalZGY::Logger Class Reference

```
#include <logger.h>
```

Inheritance diagram for InternalZGY::Logger:



### Public Member Functions

- `const LoggerFn & getCallback () const`
- `void setCallback (const LoggerFn &callback)`
- `void setCallback (int currentlevel)`

### Static Public Member Functions

- `static Logger * instance ()`
- `static const LoggerFn & getForwarder ()`

### Additional Inherited Members

#### 14.67.1 Detailed Description

Global singleton logging function.

See [LoggerBase](#) for explanation of the logger framework.

#### 14.67.2 Member Function Documentation

##### 14.67.2.1 `getCallback()`

```
const Logger::LoggerFn & InternalZGY::Logger::getCallback ( ) const
```

Return the callback to be invoked when logging.

### 14.67.2.2 getForwarder()

```
const Logger::LoggerFn & InternalZGY::Logger::getForwarder ( ) [static]
```

Return the callback to be invoked when logging, with an extra level of indirection. This will look up the global singleton on each use. If you are going to store a copy of the logger for a while, the difference between [getCallback\(\)](#) and [getForwarder\(\)](#) is that the former uses the logger that was in force when the copy was made. For short lived use, always prefer [getCallback\(\)](#) because it is more performant.

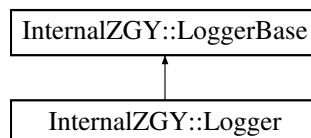
The documentation for this class was generated from the following files:

- [impl/logger.h](#)
- [impl/logger.cpp](#)

## 14.68 InternalZGY::LoggerBase Class Reference

```
#include <logger.h>
```

Inheritance diagram for InternalZGY::LoggerBase:



### Public Types

- typedef std::function< bool(int, const std::string &);> **LoggerFn**

### Static Public Member Functions

- static int **getVerboseFromEnv** (const char \*envname)
- static bool [logger](#) (const LoggerFn &logger, int priority, const std::string &str=std::string())
- static bool [logger](#) (const LoggerFn &logger, int priority, const std::ios &ss)
- static LoggerFn [emptyCallback](#) ()
- static LoggerFn [standardCallback](#) (int level, const std::string &prefix, const std::string &suffix)

## 14.68.1 Detailed Description

Logging framework.

The actual logger is a simple functor that will be invoked whenever the code wants to output a debug message. The functor may or may not output the message somewhere. The return value indicates whether the message was or should have been output. If called with an empty message don't output anything but still return the bool `should_be_output`.

The logger can be stored in a global variable or be passed around as function arguments or class members.

The callback is guaranteed to not be empty. Attempts to set an empty callback should cause a dummy callback with no output to be set.

Simple use:

```
callback(priority, message);
```

Advanced use if message may be expensive to compute:

```
if (callback(priority, ""))
    callback(priority, getMessage());
```

The functor is allowed to lie. Specifically, when the second form is used there will be code that is never executed unless the user turns on a huge amount of logging. It is useful to run some of the tests with a logger that discards all the output but claims it was or would be shown. Then run the same test with no logging, and verify there was no observed difference. This strategy also does wonders for the code coverage.

[LoggerBase](#) contains static convenience functions only. It is possible to avoid this class and use the logger functor directly if it is desirable to limit dependencies.

Need to be exported because it is used by the Python wrapper.

## 14.68.2 Member Function Documentation

### 14.68.2.1 emptyCallback()

```
LoggerBase::LoggerFn InternalZGY::LoggerBase::emptyCallback ( ) [static]
```

To be used instead of an empty functor, so the code knows it is always safe to call the logger. All output is discarded.

### 14.68.2.2 logger() [1/2]

```
bool InternalZGY::LoggerBase::logger (
    const LoggerFn & callback,
    int priority,
    const std::ios & ss ) [static]
```

Invoke the specified logger function with a stringstream argument.

Example usage:

```
if (logger(priority)) logger(priority, std::stringstream() << "Hello" << ", world.");
```

### 14.68.2.3 logger() [2/2]

```
bool InternalZGY::LoggerBase::logger (
    const LoggerFn & callback,
    int priority,
    const std::string & str = std::string() ) [static]
```

Invoke the specified logger function with a string argument. This isn't much different from invoking the callback directly. But it makes debugging slightly simpler to have an easy place to set a breakpoint. It also adds more symmetry with respect to the stringstream version, which does add value.

### 14.68.2.4 standardCallback()

```
LoggerBase::LoggerFn InternalZGY::LoggerBase::standardCallback (
    int currentlevel,
    const std::string & prefix_in,
    const std::string & suffix_in ) [static]
```

Convert an old style integer level to a logger that outputs to std::cerr all messages of the specified priority level and below.

Level 42 is treated specially. It causes all logging code to be executed but nothing will actually be output.

Level < 0 will never show anything. Not even messages with pri < 0.

Suggested use of priority levels: -1 => Serious errors, always show unless messages disabled completely. 0 => Shown during normal execution. Also in production mode. 1+ => For debugging and testing.

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

- impl/[logger.h](#)
- impl/logger.cpp

## 14.69 InternalZGY::LookupTable Class Reference

Static methods to assist working with lookup tables.

```
#include <lookuptable.h>
```

### Classes

- struct [LutInfo](#)

*Decoded contents of the lookup table for one brick or tile.*

## Static Public Member Functions

- static `std::vector< std::uint64_t > calcLookupSize` (const `std::vector< std::uint64_t > &lookup`, `std::int64_t eof`, `std::int64_t maxsize`)
- static `LutInfo getAlphaFilePosition` (`std::int64_t i`, `std::int64_t j`, `std::int64_t lod`, const `std::vector< std::array< std::int64_t, 3 >> &lodsizes`, const `std::vector< std::int64_t > &alphaoffsets`, const `std::vector< std::uint64_t > &alup`, `std::int64_t bytesperalpha`)  
*Get file offset for the specified alpha tile.*
- static `LutInfo getBrickFilePosition` (`std::int64_t i`, `std::int64_t j`, `std::int64_t k`, `std::int64_t lod`, const `std::vector< std::array< std::int64_t, 3 >> &lodsizes`, const `std::vector< std::int64_t > &brickoffsets`, const `std::vector< std::uint64_t > &blup`, const `std::vector< std::uint64_t > &bend`, `std::int64_t bytesperbrick`)  
*Get file offset or constant-value for the specified brick.*
- static void `setBrickFilePosition` (`std::int64_t i`, `std::int64_t j`, `std::int64_t k`, `std::int64_t lod`, const `LutInfo &info`, const `std::vector< std::array< std::int64_t, 3 >> &lodsizes`, const `std::vector< std::int64_t > &brickoffsets`, `std::vector< std::uint64_t > *blup`, `std::vector< std::uint64_t > *bend`)  
*Set file offset or constant-value for the specified brick.*

### 14.69.1 Detailed Description

Static methods to assist working with lookup tables.

I am uncertain whether I should extend this class. Some alternatives are:

- (1) [current choice] the `LookupTable` instance has no data members and only static methods. All context needs to be passed in. Which is tedious but makes it easier to see what the methods are actually doing.
- (2) The `LookupTable` instance is meant to be short lived. It has references to the context it needs but no good memory management or ways to handle stale data.
- (3) The `LookupTable` instance owns the actual lookup tables making this class more visible than just a helper for collecting lookup table related code. `ZgyInternalMeta` will hold a reference to a `LookupTable` instead of the current pointer to an `ILookupTableAccess`
- (4) As (3) but use separate instances for alpha and brick. Make some changes to have the methods more generic. Implement the boilerplate code from `ILookupTableAccess`. The code then becomes a drop in replacement for `LookupTableV0Access`.

### 14.69.2 Member Function Documentation

### 14.69.2.1 calcLookupSize()

```
std::vector< std::uint64_t > InternalZGY::LookupTable::calcLookupSize (
    const std::vector< std::uint64_t > & lookup,
    std::int64_t eof,
    std::int64_t maxsize ) [static]
```

Given an index => start\_offset lookup table, produce an index => end\_offset table by assuming there are no holes in the allocated data.

The function understands constant-value and compressed blocks.

If eof and maxsize are known, the code can also make the following checks:

Blocks that have a start offset > eof are unreadable and should be ignored. Set them to start and end at eof. The same applies to offsets of unknown type i.e. the most significant bit is 1 but the most significant byte is neither 0x80 (constant) nor 0xC0 (compressed). Blocks ending past eof should be assumed to end at eof.

Blocks that appear to be larger than an uncompressed block are probably too large. This may be caused by holes in the allocated data. Assume the block is the same size as an uncompressed block. If a compressed block takes up more room than an uncompressed one then the writer should simply refrain from compressing it. But for extra robustness the code that makes use of this information should be prepared to retry the access of the block really turned out to be larger.

This method might be called unconditionally on file open, or called only if at least one compressed brick was found, or it might be deferred until the first time we read a compressed brick.

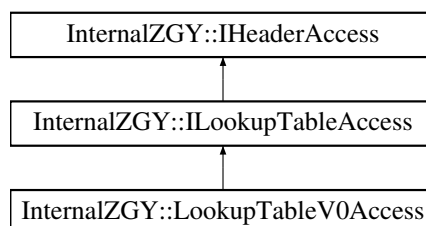
TODO-Low: If alpha tiles are present then both brick and alpha offsets ought to be considered in the same pass. The way it will work now is that for bricks, a few bricks will appear too large because they are followed by some alpha tiles. This is harmless. For alpha tiles the end offsets will be hopelessly wrong. We will need to just assume 4 KB for those.

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

- [impl/lookuptable.h](#)
- [impl/lookuptable.cpp](#)

## 14.70 InternalZGY::LookupTableV0Access Class Reference

Inheritance diagram for InternalZGY::LookupTableV0Access:



## Public Member Functions

- **LookupTableV0Access** (bool mustflip, bool isalpha, std::int64\_t num\_bricks=0)
- virtual podbytes\_t **podbytes** () const override
- virtual void **read** (const std::shared\_ptr< FileADT > &file, std::int64\_t offset, std::int64\_t size) override
- virtual void **byteswap** () override
- virtual void **dump** (std::ostream &out, const std::string &prefix="")
- virtual std::uint64\_t **lookupLinearIndex** (std::int64\_t index) const override
- virtual std::vector< std::uint64\_t > & **lup** () override
- virtual std::vector< std::uint64\_t > & **lupend** () override
- virtual const std::vector< std::uint64\_t > & **lup** () const override
- virtual const std::vector< std::uint64\_t > & **lupend** () const override

## Additional Inherited Members

### 14.70.1 Detailed Description

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

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

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

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

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

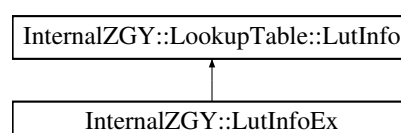
- impl/meta.cpp

## 14.71 InternalZGY::LookupTable::LutInfo Struct Reference

Decoded contents of the lookup table for one brick or tile.

```
#include <lookupable.h>
```

Inheritance diagram for InternalZGY::LookupTable::LutInfo:





## Public Member Functions

- [LutInfo](#) ([BrickStatus](#) status\_in, std::int64\_t offset, std::int64\_t size, std::uint32\_t constant)

Create an instance will all data fields filled in.

## Public Attributes

- [BrickStatus](#) status
- std::int64\_t offset\_in\_file
- std::int64\_t size\_in\_file
- std::uint32\_t raw\_constant

### 14.71.1 Detailed Description

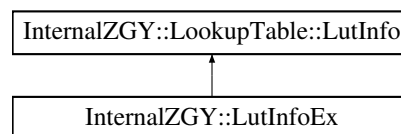
Decoded contents of the lookup table for one brick or tile.

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

- [impl/lookuptable.h](#)

## 14.72 InternalZGY::LutInfoEx Struct Reference

Inheritance diagram for InternalZGY::LutInfoEx:



## Public Member Functions

- [LutInfoEx](#) (const [LookupTable::LutInfo](#) &info, const [IJK](#) &pos\_in, double constvalue\_in)

## Public Attributes

- [IJK](#) survey\_position
- double double\_constvalue

### 14.72.1 Detailed Description

TODO-Low might want to fold this into [LookupTable::LutInfo](#).

Add position in samples. [LookupTable::getBrickFilePosition\(\)](#) cannot easily store this because it only has the brick position and would need to know the brick size to get sample position.

Add constvalue after decoding.

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

- [impl/bulk.cpp](#)

## 14.73 InternalZGY::NullCompressPlugin Class Reference

Example compression plug-in that always fails to compress.

### Classes

- class [Register](#)

*Register the compress and decompress functions in the factory.*

### Static Public Member Functions

- static [compressor\\_t](#) [getCompressor](#) (const std::vector< std::string > &)  
*Get a functor for compressing data.*
- static [rawdata\\_t](#) [compress](#) (const [rawdata\\_t](#) &data, const [index3\\_t](#) &shape)  
*Compress using the chosen algorithm.*
- static [rawdata\\_t](#) [decompress](#) (const [rawdata\\_t](#) &cdata, const [BrickStatus](#) &status, const [index3\\_t](#) &shape)  
*Decompress data if it was compressed by our algorithm.*

### 14.73.1 Detailed Description

Example compression plug-in that always fails to compress.

This class is here just as an example and for documentation. Real compression classes may use "copydoc" instead of repeating what is explained here.

### 14.73.2 Member Function Documentation

#### 14.73.2.1 [compress\(\)](#)

```
static rawdata\_t InternalZGY::NullCompressPlugin::compress (  
    const rawdata\_t & data,  
    const index3\_t & shape ) [static]
```

Compress using the chosen algorithm.

#### Parameters

<i>data</i>	Smart pointer to data, and size in bytes of data.
<i>shape</i>	hint for 3d layout of data in number of samples.

#### Returns

Data in the same format as the input.

The number and type of parameters, except for the first one, varies depending on the algorithm being used. Some algorithms might not need additional parameters at all.

Current assumptions that apply to all compression algorithms. Except for the one about thread safety these are subject to change.

- The method must be thread safe.
- The algorithm is responsible any for big / little endian conversion.
- The compressed data stream must never be larger than the uncompressed data. If it looks like this is going to happen then just return {nullptr, 0} telling the caller that compression is not possible.
- If the algorithm decides that compression is inadvisable for other reasons, such as NaN or Inf in the data, it is also acceptable to return {nullptr, 0}. Never return the input data unchanged because this will then be flagged as compressed. And the uncompress algorithm will fail miserably.
- The compressed data stream should start with a magic number that the decompressor can recognize.

### 14.73.2.2 decompress()

```
static rawdata_t InternalZGY::NullCompressPlugin::decompress (  
    const rawdata_t & cdata,  
    const BrickStatus & status,  
    const index3_t & shape ) [static]
```

Decompress data if it was compressed by our algorithm.

#### Parameters

<i>cdata</i>	Compressed data, possibly with trailing garbage.
<i>status</i>	Might be used to identify which algorithm was used.
<i>shape</i>	rank and size in samples of the uncompressed result.

The algorithm should check both the status argument and any magic number at the start of the buffer to decide whether this is data we know how to decompress. If not then return {nullptr, 0} which should make the caller try something else.

If the algorithm is recognized but the data cannot be decompressed because it is corrupt then the method may throw an exception. In that case no further algorithms will be tried.

Older versions of this function also received the value type of the data that was compressed and the value type we want it returned in This is now redundant since we only allow compressing float data. Anything else is simply not useful.

Passing an uncompressed brick to this function is an error. I.e. status will never be Normal. We don't have enough context to handle uncompressed bricks that might require byteswapping and fix for legacy quirks. Also cannot handle constant bricks, missing bricks, etc.

Current assumptions that apply to all decompression algorithms. Except for the one about thread safety these are subject to change.

- The method must be thread safe.
- The compressed data stream is allowed to have trailing garbage; this must be silently ignored by the decompressor.
- The algorithm may assume that the compressed data stream will never be longer than the uncompressed data. This is enforced by the compressor.
- The reason for the two assumptions above is an implementation detail. The reported size of a compressed brick isn't completely reliable. This might change in the next version of the file format.
- The decompressor must verify that both the status (usually set to `BrickStatus.Compressed`) and the magic number at the start of the compressed stream matches is correct. Relying on only status is currently not sufficient because all compressed bricks get the same status.

### 14.73.2.3 `getCompressor()`

```
static compressor_t InternalZGY::NullCompressPlugin::getCompressor (
    const std::vector< std::string > & ) [static]
```

Get a functor for compressing data.

Parse the stringly typed arguments that the factory sent us. Do a consistency check on the arguments. Capture the now strongly typed compressor arguments in a lambda that can be used to compress one block of data.

The returned lambda function is supposed to be thread safe.

The return is allowed to be an empty function. The caller will treat this as if no compression had been requested. This is not precisely the same as returning a factor that always returns "I give up". The latter case would limit the file to float32 type and might turn off alignments in the file.

If the compressor needs mutable state, e.g. to accumulate performance measurements, it can be declared as an instance method allowing the lambda to capture "this". The class would normally be noncopyable. The returned lambda can be copied at will; the copies will share the same state. Remember to use locks to protect the shared state in "this".

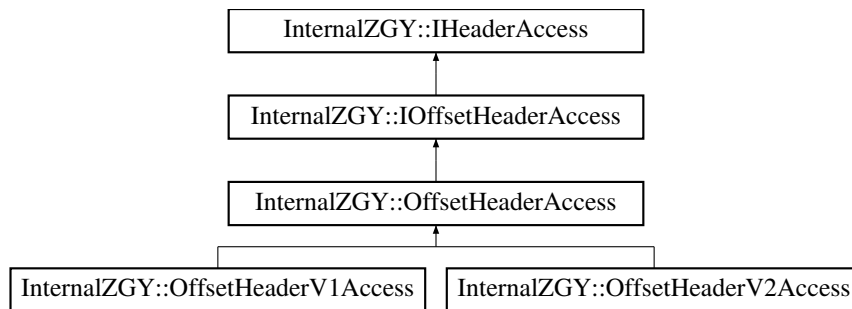
Another option for mutable state is to have this function allocate some kind of "state" instance and capture it in the lambda. How to access that state later, e.g. to extract any timing measurements, is left as an exercise ;-).

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

- [impl/compress\\_null.cpp](#)

## 14.74 InternalZGY::OffsetHeaderAccess Class Reference

Inheritance diagram for InternalZGY::OffsetHeaderAccess:



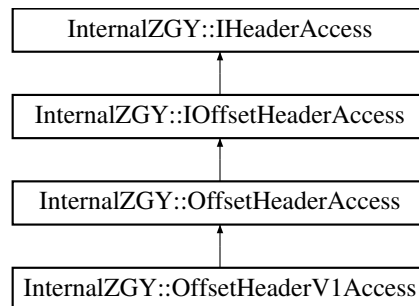
### Additional Inherited Members

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

- impl/meta.cpp

## 14.75 InternalZGY::OffsetHeaderV1Access Class Reference

Inheritance diagram for InternalZGY::OffsetHeaderV1Access:



### Public Member Functions

- virtual podbytes\_t **podbytes** () const override
- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset) override
- virtual void **byteswap** () override
- virtual void **calculate** (const std::shared\_ptr< [InfoHeaderAccess](#) > &ih) override
- virtual std::int64\_t **infoff** () const override
- virtual std::int64\_t **stroff** () const override
- virtual std::int64\_t **alphalupoff** () const override
- virtual std::int64\_t **bricklupoff** () const override
- virtual std::int64\_t **histoff** () const override
- virtual std::int64\_t **infoffsize** () const override
- virtual std::int64\_t **histsize** () const override
- virtual std::int64\_t **alphalupsize** () const override
- virtual std::int64\_t **bricklupsize** () const override

## Public Attributes

- [OffsetHeaderV1POD\\_pod](#)
- std::int64\_t \_derived\_infsize
- std::int64\_t \_derived\_histsize
- std::int64\_t \_derived\_alphalupsize
- std::int64\_t \_derived\_bricklupsize

## Additional Inherited Members

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

- impl/meta.cpp

## 14.76 InternalZGY::OffsetHeaderV1POD Class Reference

Physical layout of Offset Header version 1.

## Public Attributes

- std::int64\_t \_inffoff
- std::int64\_t \_alphalupoff
- std::int64\_t \_bricklupoff
- std::int64\_t \_histoff

### 14.76.1 Detailed Description

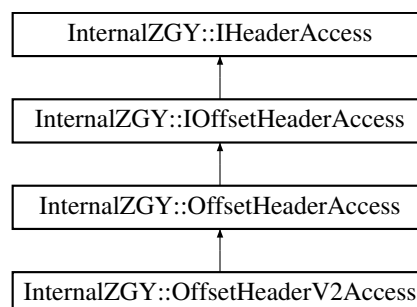
Physical layout of Offset Header version 1.

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

- impl/meta.cpp

## 14.77 InternalZGY::OffsetHeaderV2Access Class Reference

Inheritance diagram for InternalZGY::OffsetHeaderV2Access:



### Public Member Functions

- virtual podbytes\_t **podbytes** () const override
- virtual void **read** (const std::shared\_ptr< [FileADT](#) > &file, std::int64\_t offset) override
- virtual void **byteswap** () override
- virtual void **calculate** (const std::shared\_ptr< [IInfoHeaderAccess](#) > &ih) override
- virtual std::int64\_t **infoff** () const override
- virtual std::int64\_t **stroff** () const override
- virtual std::int64\_t **alupalupoff** () const override
- virtual std::int64\_t **bricklupoff** () const override
- virtual std::int64\_t **histoff** () const override
- virtual std::int64\_t **infsz** () const override
- virtual std::int64\_t **histsz** () const override
- virtual std::int64\_t **alupalupsz** () const override
- virtual std::int64\_t **bricklupsz** () const override

### Public Attributes

- std::int64\_t **\_derived\_infoff**
- std::int64\_t **\_derived\_stroff**
- std::int64\_t **\_derived\_alupalupoff**
- std::int64\_t **\_derived\_bricklupoff**
- std::int64\_t **\_derived\_histoff**
- std::int64\_t **\_derived\_infsz**
- std::int64\_t **\_derived\_strsz**
- std::int64\_t **\_derived\_histsz**
- std::int64\_t **\_derived\_alupalupsz**
- std::int64\_t **\_derived\_bricklupsz**

### Additional Inherited Members

#### 14.77.1 Member Function Documentation

##### 14.77.1.1 calculate()

```
void InternalZGY::OffsetHeaderV2Access::calculate (
    const std::shared_ptr< IInfoHeaderAccess > & ih ) [override], [virtual]
```

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

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

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

Implements [InternalZGY::IOffsetHeaderAccess](#).

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

- impl/meta.cpp

## 14.78 InternalZGY::OffsetHeaderV2POD Class Reference

Physical layout of Offser Header version 2 and 3 (empty).

### 14.78.1 Detailed Description

Physical layout of Offser Header version 2 and 3 (empty).

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

- impl/meta.cpp

## 14.79 InternalZGY::OrderedCornerPoints Class Reference

```
#include <cornerpoints.h>
```

### Classes

- struct [Element](#)

### Public Types

- enum { [Min0Min1](#), [Max0Min1](#), [Min0Max1](#), [Max0Max1](#) }
- typedef std::int64\_t [index\\_type](#)
- typedef float [annot\\_type](#)
- typedef double [coord\\_type](#)

### Public Member Functions

- [OrderedCornerPoints](#) ()
- [OrderedCornerPoints](#) ([annot\\_type](#) il0, [annot\\_type](#) ilinc, size\_t ilcnt, [annot\\_type](#) xl0, [annot\\_type](#) xlinc, size\_t xlcnt, const std::array< std::array< [coord\\_type](#), 2 >, 4 > &ocp)
- [OrderedCornerPoints](#) ([annot\\_type](#) il0, [annot\\_type](#) ilinc, size\_t ilcnt, [annot\\_type](#) xl0, [annot\\_type](#) xlinc, size\_t xlcnt, [annot\\_type](#) acp0il, [annot\\_type](#) acp0xl, [coord\\_type](#) acp0x, [coord\\_type](#) acp0y, [annot\\_type](#) acp1il, [annot\\_type](#) acp1xl, [coord\\_type](#) acp1x, [coord\\_type](#) acp1y, [annot\\_type](#) acp2il, [annot\\_type](#) acp2xl, [coord\\_type](#) acp2x, [coord\\_type](#) acp2y)
- const [Element](#) & [operator\[\]](#) (size\_t i) const
- std::array< std::array< [coord\\_type](#), 2 >, 4 > [index\\_coords](#) () const  
*corner coordinates in index space.*
- std::array< std::array< [coord\\_type](#), 2 >, 4 > [annot\\_coords](#) () const  
*corner coordinates in annotation space.*
- std::array< std::array< [coord\\_type](#), 2 >, 4 > [world\\_coords](#) () const  
*corner coordinates in world space.*



### 14.79.1 Detailed Description

This class can be used to calculate the map projection (x, y) coordinates of the four corners of a cube from a set of arbitrary control points (ACP). The result is ordered according to the Petrel Ordered Corner Points (OCP) definition, which is as follows corresponding to bulk data access indices:

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

Ref: PetrelOrientationHandling

### 14.79.2 Member Typedef Documentation

#### 14.79.2.1 annot\_type

```
typedef float InternalZGY::OrderedCornerPoints::annot_type
```

Bulk-data index type.

#### 14.79.2.2 coord\_type

```
typedef double InternalZGY::OrderedCornerPoints::coord_type
```

Inline/crossline annotation index datatype.

### 14.79.3 Member Enumeration Documentation

#### 14.79.3.1 anonymous enum

```
anonymous enum
```

Ordering

Enumerator

Max0Min1	Lowest index in dimension 0 (I), lowest index in dimension 1 (J).
Min0Max1	Highest index in dimension 0 (I), lowest index in dimension 1 (J).
Max0Max1	Lowest index in dimension 0 (I), highest index in dimension 1 (J). Highest index in dimension 0 (I), highest index in dimension 1 (J).

## 14.79.4 Constructor & Destructor Documentation

### 14.79.4.1 OrderedCornerPoints() [1/3]

```
InternalZGY::OrderedCornerPoints::OrderedCornerPoints ( )
```

Default constructor. Initializes OCPs to zero.

### 14.79.4.2 OrderedCornerPoints() [2/3]

```
InternalZGY::OrderedCornerPoints::OrderedCornerPoints (
    annot_type il0,
    annot_type ilinc,
    size_t ilcnt,
    annot_type xl0,
    annot_type xlinc,
    size_t xlcnt,
    const std::array< std::array< coord_type, 2 >, 4 > & ocp )
```

Construct OCPs from cube extent in annotation space and four OCP coordinates.

#### Parameters

<i>il0</i>	Inline annotation corresponding to bulk data index 0 along dimension 0 (I).
<i>ilinc</i>	Inline annotation increment corresponding to bulk data index increment of 1 along dimension 0 (I).
<i>ilcnt</i>	Number of samples along dimension 0 (I).
<i>xl0</i>	Crossline annotation corresponding to bulk data index 0 along dimension 1 (J).
<i>xlinc</i>	Crossline annotation increment corresponding to bulk data index increment of 1 along dimension 1 (J).
<i>xlcnt</i>	Number of samples along dimension 1 (J).
<i>ocp</i>	Four sets of (x, y) map projection coordinates expected to be in correct OCP ordering.

### 14.79.4.3 OrderedCornerPoints() [3/3]

```
InternalZGY::OrderedCornerPoints::OrderedCornerPoints (
    annot_type il0,
    annot_type ilinc,
    size_t ilcnt,
    annot_type xl0,
    annot_type xlinc,
    size_t xlcnt,
    annot_type acp0il,
    annot_type acp0xl,
    coord_type acp0x,
    coord_type acp0y,
    annot_type acp1il,
```

```
annot_type acp1xl,  
coord_type acp1x,  
coord_type acp1y,  
annot_type acp2il,  
annot_type acp2xl,  
coord_type acp2x,  
coord_type acp2y )
```

Construct OCPs from cube extent in annotation space and three ACPs.

### Parameters

<i>il0</i>	Inline annotation corresponding to bulk data index 0 along dimension 0 (I).
<i>ilinc</i>	Inline annotation increment corresponding to bulk data index increment of 1 along dimension 0 (I).
<i>ilcnt</i>	Number of samples along dimension 0 (I).
<i>xl0</i>	Crossline annotation corresponding to bulk data index 0 along dimension 1 (J).
<i>xlinc</i>	Crossline annotation increment corresponding to bulk data index increment of 1 along dimension 1 (J).
<i>xlcnt</i>	Number of samples along dimension 1 (J).
<i>acp0il</i>	Inline annotation for first ACP.
<i>acp0xl</i>	Crossline annotation for first ACP.
<i>acp0x</i>	X (easting) map coordinate for first ACP.
<i>acp0y</i>	Y (northing) map coordinate for first ACP.
<i>acp1il</i>	Inline annotation for second ACP.
<i>acp1xl</i>	Crossline annotation for second ACP.
<i>acp1x</i>	X (easting) map coordinate for second ACP.
<i>acp1y</i>	Y (northing) map coordinate for second ACP.
<i>acp2il</i>	Inline annotation for third ACP.
<i>acp2xl</i>	Crossline annotation for third ACP.
<i>acp2x</i>	X (easting) map coordinate for third ACP.
<i>acp2y</i>	Y (northing) map coordinate for third ACP.

holds (inline, crossline)

receives (x, y)

## 14.79.5 Member Function Documentation

### 14.79.5.1 operator[]()

```
const Element& InternalZGY::OrderedCornerPoints::operator[] (   
    size_t i ) const
```

Get an OCP.

### Parameters

<i>i</i>	OCP index.
----------	------------

## Returns

[Element](#) corresponding to ith OCP.

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

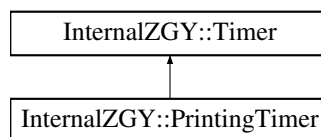
- [impl/cornerpoints.h](#)
- [impl/cornerpoints.cpp](#)

## 14.80 InternalZGY::PrintingTimer Class Reference

[Timer](#) that prints its result when going out of scope.

```
#include <timer.h>
```

Inheritance diagram for InternalZGY::PrintingTimer:



## Public Member Functions

- **PrintingTimer** (const char \*name, int level=1, bool startrunning=true)
- void **print** ()

## Additional Inherited Members

### 14.80.1 Detailed Description

[Timer](#) that prints its result when going out of scope.

Convenience class that stops (if needed) the timer and logs the result when the instance goes out of scope.

TODO-Low it would be more elegant to pass the logger function as an argument and elide the dependency to the logger module.

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

- [impl/timer.h](#)
- [impl/timer.cpp](#)

## 14.81 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.*

#### 14.81.1 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.

#### 14.81.2 Constructor & Destructor Documentation

##### 14.81.2.1 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.

## 14.81.3 Member Function Documentation

### 14.81.3.1 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](#)

## 14.82 InternalZGY::PushEnvironment Class Reference

```
#include <environment.h>
```

### Public Member Functions

- [PushEnvironment](#) (const char \*name, const char \*value=nullptr)
- [~PushEnvironment](#) ()  
*Restore the environment variable to its previous content.*
- void [restore](#) ()  
*Restore the environment variable to its previous content.*

### 14.82.1 Detailed Description

Set the environment as specified, and arrange for the previous value to be restored when the instance goes out of scope.

### 14.82.2 Constructor & Destructor Documentation

#### 14.82.2.1 PushEnvironment()

```
InternalZGY::PushEnvironment::PushEnvironment (
    const char * name,
    const char * value = nullptr )
```

Set the environment as specified, and arrange for the previous value to be restored when the instance goes out of scope.

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

- impl/[environment.h](#)
- impl/environment.cpp

## 14.83 InternalZGY::RawDataTypeDetails Class Reference

```
#include <types.h>
```

### Public Member Functions

- **RawDataTypeDetails** ([RawDataType](#) type)

### Public Attributes

- std::size\_t **size**
- double **lowest**
- double **highest**
- bool **is\_integer**
- bool **is\_signed**

#### 14.83.1 Detailed Description

Helper for enum RawDataType in [enum.h](#): Given an enum tag, return traits for the C++ type it represents.

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

- impl/[types.h](#)
- impl/types.cpp

## 14.84 InternalZGY::RawDataTypeTraits< T > Struct Template Reference

```
#include <types.h>
```

## 14.84.1 Detailed Description

```
template<typename T>
struct InternalZGY::RawDataTypeTraits< T >
```

Helper for enum RawDataType in [enum.h](#): convert a templated type into a RawDataType enum.

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

- impl/[types.h](#)

## 14.85 InternalZGY::RawDataTypeTraits< float > Struct Reference

### Static Public Attributes

- static const [RawDataType](#) **datatype** = RawDataType::Float32

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

- impl/[types.h](#)

## 14.86 InternalZGY::RawDataTypeTraits< std::int16\_t > Struct Reference

### Static Public Attributes

- static const [RawDataType](#) **datatype** = RawDataType::SignedInt16

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

- impl/[types.h](#)

## 14.87 InternalZGY::RawDataTypeTraits< std::int32\_t > Struct Reference

### Static Public Attributes

- static const [RawDataType](#) **datatype** = RawDataType::SignedInt32

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

- impl/[types.h](#)



## 14.88 InternalZGY::RawDataTypeTraits< std::int8\_t > Struct Reference

### Static Public Attributes

- static const [RawDataType](#) **datatype** = RawDataType::SignedInt8

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

- impl/[types.h](#)

## 14.89 InternalZGY::RawDataTypeTraits< std::uint16\_t > Struct Reference

### Static Public Attributes

- static const [RawDataType](#) **datatype** = RawDataType::UnsignedInt16

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

- impl/[types.h](#)

## 14.90 InternalZGY::RawDataTypeTraits< std::uint32\_t > Struct Reference

### Static Public Attributes

- static const [RawDataType](#) **datatype** = RawDataType::UnsignedInt32

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

- impl/[types.h](#)

## 14.91 InternalZGY::RawDataTypeTraits< std::uint8\_t > Struct Reference

### Static Public Attributes

- static const [RawDataType](#) **datatype** = RawDataType::UnsignedInt8

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

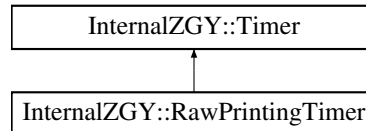
- impl/[types.h](#)

## 14.92 InternalZGY::RawPrintingTimer Class Reference

[Timer](#) that prints its result when going out of scope.

```
#include <timer.h>
```

Inheritance diagram for InternalZGY::RawPrintingTimer:



### Public Member Functions

- **RawPrintingTimer** (const char \*name, int level=1)
- void **print** ()

### Additional Inherited Members

#### 14.92.1 Detailed Description

[Timer](#) that prints its result when going out of scope.

Convenience class that stops (if needed) the timer and logs the result directly to stdout when the instance goes out of scope. Hopefully this is safe even when run at exit (e.g. in a static destructor) when the higher level logging mechanism or even iostreams might be shut down. Use for ad-hoc debugging only.

TODO-Low if [PrintingTimer](#) had taken a logger argument, there would be no need for this variant. On the other hand I would get a dependency to STL in this file.

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

- impl/[timer.h](#)
- impl/timer.cpp

## 14.93 InternalZGY::ReadRequest Class Reference

### Public Types

- typedef std::function< void(const void \*, std::int64\_t)> **delivery\_t**

### Public Member Functions

- **ReadRequest** (std::int64\_t offset\_in, std::int64\_t size\_in, const delivery\_t &delivery\_in)

## Public Attributes

- `std::int64_t` **offset**
- `std::int64_t` **size**
- `delivery_t` **delivery**

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

- [impl/file.h](#)

## 14.94 InternalZGY::NullCompressPlugin::Register Class Reference

Register the compress and decompress functions in the factory.

### Public Member Functions

- [Register](#) ()  
*Register this plug-in.*
- [~Register](#) ()  
*Unregister this plug-in.*

### 14.94.1 Detailed Description

Register the compress and decompress functions in the factory.

Compressors use a factory so the application code can write something like  
`compressor = ZgyCompressFactory("ZFP", "snr", "30");`

without getting a link time dependency on the "ZFP" code.

Decompressors use a factory because when reading blocks from the file it is not possible to hard code a specific decompressor.

It is safe to construct an instance from a static initializer. Do note that this is not true for the destructor. If Register will be instantiated from a static initializer then the destructor should be empty or omitted altogether.

The reason for this is that the destruction is done in an unspecified order at exit. Also, cleaning up the registry on exit would be pointless anyway.

### 14.94.2 Constructor & Destructor Documentation

#### 14.94.2.1 Register()

```
InternalZGY::NullCompressPlugin::Register::Register ( )
```

Register this plug-in.

It is safe to construct an instance from a static initializer. Do note that this is not true for the destructor. If Register will be instantiated from a static initializer then the destructor should be empty or omitted altogether.

## 14.94.2.2 ~Register()

```
InternalZGY::NullCompressPlugin::Register::~~Register ( )
```

Unregister this plug-in.

Do NOTHING if Register will be used as a static initializer. We don't want any cleanup to happen on application exit. That is pointless and it is also tricky to get destructors called in the right order.

If Register is going to be instantiated only when needed then a destructor is in order. See e.g. MockCompressPlugin and MockCompressInstaller in the unit tests.

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

- [impl/compress\\_null.cpp](#)

## 14.95 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*

## 14.95.1 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 documentation for this class was generated from the following file:

- [api.h](#)

## 14.96 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.*

## 14.96.1 Detailed Description

Statistics of all sample values on the file.

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

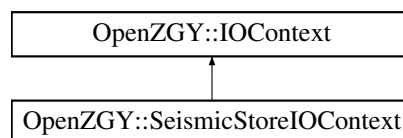
- [api.h](#)

## 14.97 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](#) & [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)

### 14.97.1 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.

### 14.97.2 Member Function Documentation

#### 14.97.2.1 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.

#### 14.97.2.2 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.

#### 14.97.2.3 legaltag()

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

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

## 14.97.2.4 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.

## 14.97.2.5 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.

## 14.97.2.6 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.



### 14.97.2.7 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.

### 14.97.2.8 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.

### 14.97.2.9 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.

### 14.97.2.10 segsize()

```
SeismicStoreIOContext& OpenZGY::SeismicStoreIOContext::segsz (
    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 1024 (i.e. 1 GB). The default should work fine in almost all cases.

### 14.97.2.11 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.

## 14.97.2.12 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.

## 14.97.2.13 toString()

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

Display the context in a human readable format for debugging.

Implements [OpenZGY::IOContext](#).

## 14.97.2.14 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 files:

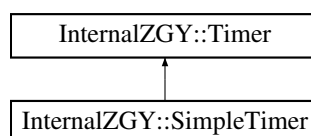
- [iocontext.h](#)
- [iocontext.cpp](#)

## 14.98 InternalZGY::SimpleTimer Class Reference

[Timer](#) that knows where to store the result.

```
#include <timer.h>
```

Inheritance diagram for InternalZGY::SimpleTimer:



## Public Member Functions

- **SimpleTimer** ([SummaryTimer](#) &owner, bool enabled=true)

## Additional Inherited Members

### 14.98.1 Detailed Description

[Timer](#) that knows where to store the result.

[SimpleTimer](#) is normally short lived. The [SimpleTimer](#) constructor takes a [SummaryTimer](#) as an argument. When the [SimpleTimer](#) goes out of scope the result is added to the accumulate timer. The caller is responsible for not allowing the [SummaryTimer](#) to go out of scope before the [SimpleTimer](#) does.

There is also a [SummaryPrintingTimer](#) class that extends [SummaryTimer](#) to print a single line report when it goes out of scope.

Note that [Timer](#) and [SimpleTimer](#) are not designed to be threadsafe. This makes no sense when measuring a single piece of code execution. [SummaryTimer::add\(\)](#) is threadsafe. This means the following example with the [SummaryTimer](#) being shared among threads will work. The example measures every execution of the function and prints a report on application exit.

```
void SomeFunction() {  
    static SummaryPrintingTimer pt("MyTimer");  
    SimpleTimer tt(pt);  
    // timing the following code...  
}
```

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

- impl/[timer.h](#)

## 14.99 InternalZGY::StatisticData Class Reference

Holds the result of computing statistics.

```
#include <statisticdata.h>
```

## Public Types

- typedef std::int64\_t **count\_type**

## Public Member Functions

- [StatisticData](#) ()
- [StatisticData](#) (count\_type cnt, count\_type inf, double sum, double ssq, double min, double max)
- [StatisticData](#) (const count\_type \*bins, int nbins, double range\_min, double range\_max, bool is\_int8)
- void [add](#) (double value)
- [StatisticData](#) & [operator+=](#) (const [StatisticData](#) &other)
- [StatisticData](#) & [operator-=](#) (const [StatisticData](#) &other)
- [StatisticData](#) & [operator\\*=](#) (count\_type factor)
- void [scale](#) (double oldmin, double oldmax, double newmin, double newmax)
- void [trimRange](#) (const count\_type \*bins, int nbins, double range\_min, double range\_max)

## 14.99.1 Detailed Description

Holds the result of computing statistics.

## 14.99.2 Constructor & Destructor Documentation

### 14.99.2.1 `StatisticData()` [1/3]

```
InternalZGY::StatisticData::StatisticData ( )
```

Initialize a [StatisticData](#) to empty. min/max are by definition irrelevant when count is zero, but they will be set to 0 here just to have a consistent value.

### 14.99.2.2 `StatisticData()` [2/3]

```
InternalZGY::StatisticData::StatisticData (
    count_type cnt,
    count_type inf,
    double sum,
    double ssq,
    double min,
    double max )
```

Initialize a [StatisticData](#) from discrete values.

### 14.99.2.3 `StatisticData()` [3/3]

```
InternalZGY::StatisticData::StatisticData (
    const count_type * bins,
    int nbins,
    double range_min,
    double range_max,
    bool is_int8 )
```

Create a [StatisticData](#) from histogram information.

If `is_int8` is true, the histogram was populated from 8-bit data and the histogram has 256 bins. In this case the result will be precise. The method needs to be told about this. Otherwise it would have to add 1/2 bin width slop, not knowing that there can only be a single value in each bin. That value might not be integral any longer due to scaling, but there can still only be one.

If `is_int8` is false, this isn't as accurate as the actual statistics since each sample will be rounded to the center of the bin it falls in. Also, `Inf` will always be 0 since the histogram doesn't count outliers. The result is still useful as a consistency check. And also to update the min/max range in the real statistics after subtracting samples.

## 14.99.3 Member Function Documentation

### 14.99.3.1 add()

```
void InternalZGY::StatisticData::add (
    double value )
```

Add a single sample to the statistics.

### 14.99.3.2 operator\*=( )

```
StatisticData & InternalZGY::StatisticData::operator*= (
    count_type factor )
```

Multiply [StatisticData](#) with a constant N, equivalent to creating a new instance and adding the old one to it N times. N can also be negative. The min/max range is not affected.

### 14.99.3.3 operator+=( )

```
StatisticData & InternalZGY::StatisticData::operator+= (
    const StatisticData & other )
```

Add more samples to an existing [StatisticData](#). other is also allowed to hold negative counts, this will cause samples to be removed. But the min/max range will still be expanded.

### 14.99.3.4 operator-= ( )

```
StatisticData & InternalZGY::StatisticData::operator-= (
    const StatisticData & other )
```

Remove samples from an existing [StatisticData](#). The resulting min/max range will be the union of the two classes, not the intersection. Could also have chosen to leave the min/max uncanged. But it is more consistent to do a union, because -a+b will then work like b-a. Either way, you can call [trimRange\(\)](#) afterwards if you know what the range ought to be.

### 14.99.3.5 scale()

```
void InternalZGY::StatisticData::scale (
    double oldmin,
    double oldmax,
    double newmin,
    double newmax )
```

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

Note: If caler has (slope, intercept) instead of the 4 factors, pass oldmin=0, oldmax=1, newmin=intercept, newmax=intercept+slope.

### 14.99.3.6 trimRange()

```
void InternalZGY::StatisticData::trimRange (
    const count_type * bins,
    int nbins,
    double range_min,
    double range_max )
```

Try to narrow the range found in the statistics based on the information we have in the histogram. This is particularly useful after subtracting something from the histogram.

The range found in the histogram is normally wider than or equal to the statistics range. If this is not the case, we know the min/max in the statistics are incorrect. We cannot know the precise new range without rescanning all the data, but we can approximate it using the histogram. There is no point in doing any of this if the histogram is empty. In that case min/max are irrelevant, and will be reset once data has been added. If any of the overflow bins are in use, these are ignored. That should not happen in normal circumstances, but could occur if stats are calculated from application provided data, but maintained in a fixed histogram because that is what the file format dictates.

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

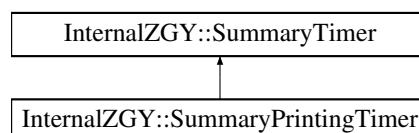
- [impl/statisticdata.h](#)
- [impl/statisticdata.cpp](#)

## 14.100 InternalZGY::SummaryPrintingTimer Class Reference

[SummaryTimer](#) that prints its result when going out of scope.

```
#include <timer.h>
```

Inheritance diagram for InternalZGY::SummaryPrintingTimer:



### Public Member Functions

- **SummaryPrintingTimer** (const char \*name)
- virtual void **print** ()

## 14.100.1 Detailed Description

[SummaryTimer](#) that prints its result when going out of scope.

[SimpleTimer](#) is normally short lived. The [SimpleTimer](#) constructor takes a [SummaryTimer](#) as an argument. When the [SimpleTimer](#) goes out of scope the result is added to the accumulate timer. The caller is responsible for not allowing the [SummaryTimer](#) to go out of scope before the [SimpleTimer](#) does.

There is also a [SummaryPrintingTimer](#) class that extends [SummaryTimer](#) to print a single line report when it goes out of scope.

Note that [Timer](#) and [SimpleTimer](#) are not designed to be threadsafe. This makes no sense when measuring a single piece of code execution. [SummaryTimer::add\(\)](#) is threadsafe. This means the following example with the [SummaryTimer](#) being shared among threads will work. The example measures every execution of the function and prints a report on application exit.

```
void SomeFunction() {  
    static SummaryPrintingTimer pt("MyTimer");  
    SimpleTimer tt(pt);  
    // timing the following code...  
}
```

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

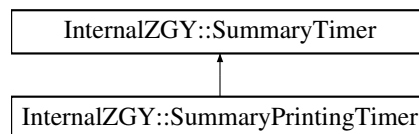
- [impl/timer.h](#)
- [impl/timer.cpp](#)

## 14.101 InternalZGY::SummaryTimer Class Reference

Hold the timing results from zero or more [Timer](#) instances.

```
#include <timer.h>
```

Inheritance diagram for InternalZGY::SummaryTimer:



## Classes

- class [Impl](#)

## Public Member Functions

- **SummaryTimer** (const char \*name)
- **SummaryTimer** (const [SummaryTimer](#) &)=delete
- [SummaryTimer](#) & **operator=** (const [SummaryTimer](#) &)=delete
- double **getFrequency** () const
- int **getCount** () const
- double **getTotal** () const
- double **getLast** () const
- const char \* **getName** () const
- const char \* **getValue** (bool details, bool msonly) const
- void **reset** ()  
*Clear all counters.*
- void **add** (int count, double total, double last)  
*Add the contents of the specified [Timer](#) to our summary.*
- void **add** (const [Timer](#) &t)  
*Add the contents of the specified [Timer](#) to our summary.*

## 14.101.1 Detailed Description

Hold the timing results from zero or more [Timer](#) instances.

[SimpleTimer](#) is normally short lived. The [SimpleTimer](#) constructor takes a [SummaryTimer](#) as an argument. When the [SimpleTimer](#) goes out of scope the result is added to the accumulate timer. The caller is responsible for not allowing the [SummaryTimer](#) to go out of scope before the [SimpleTimer](#) does.

There is also a [SummaryPrintingTimer](#) class that extends [SummaryTimer](#) to print a single line report when it goes out of scope.

Note that [Timer](#) and [SimpleTimer](#) are not designed to be threadsafe. This makes no sense when measuring a single piece of code execution. [SummaryTimer::add\(\)](#) is threadsafe. This means the following example with the [SummaryTimer](#) being shared among threads will work. The example measures every execution of the function and prints a report on application exit.

```
void SomeFunction() {  
    static SummaryPrintingTimer pt("MyTimer");  
    SimpleTimer tt(pt);  
    // timing the following code...  
}
```

## 14.101.2 Member Function Documentation

### 14.101.2.1 add() [1/2]

```
void InternalZGY::SummaryTimer::add (  
    const Timer & t )
```

Add the contents of the specified [Timer](#) to our summary.

The caller is responsible for stopping the timer first.

### 14.101.2.2 add() [2/2]

```
void InternalZGY::SummaryTimer::add (  
    int count,  
    double total,  
    double last )
```

Add the contents of the specified [Timer](#) to our summary.

The information to add is passed as discrete arguments, so it is possible to e.g. add an managed timer to an unmanaged one.

### 14.101.2.3 reset()

```
void InternalZGY::SummaryTimer::reset ( )
```

Clear all counters.

The timer name is unchanged.

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

- [impl/timer.h](#)
- [impl/timer.cpp](#)



## 14.102 InternalZGY::ImplicitLinearTransform2d::TiePoint Struct Reference

```
#include <iltf2d.h>
```

### Public Member Functions

- [TiePoint](#) ()
- [TiePoint](#) (value\_type a0, value\_type a1, value\_type b0, value\_type b1)

### Public Attributes

- value\_type a [2]
- value\_type b [2]

### 14.102.1 Detailed Description

Type of this class. Tie-point definition.

### 14.102.2 Constructor & Destructor Documentation

#### 14.102.2.1 TiePoint() [1/2]

```
InternalZGY::ImplicitLinearTransform2d::TiePoint::TiePoint ( )
```

Coordinates in space B. Default constructor, initialize to all-zeroes.

#### 14.102.2.2 TiePoint() [2/2]

```
InternalZGY::ImplicitLinearTransform2d::TiePoint::TiePoint (
    value_type a0,
    value_type a1,
    value_type b0,
    value_type b1 )
```

Construct from arguments.

#### Parameters

<i>a0</i>	Coordinate in first dimension of space A.
<i>a1</i>	Coordinate in second dimension of space A.
<i>b0</i>	Coordinate in first dimension of space B.
<i>b1</i>	Coordinate in second dimension of space B.

## 14.102.3 Member Data Documentation

### 14.102.3.1 b

`value_type InternalZGY::ImplicitLinearTransform2d::TiePoint::b[2]`

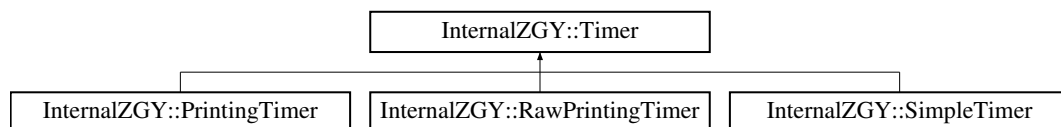
Coordinates in space A.

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

- [impl/iltf2d.h](#)
- [impl/iltf2d.cpp](#)

## 14.103 InternalZGY::Timer Class Reference

Inheritance diagram for InternalZGY::Timer:



### Public Member Functions

- [Timer](#) (bool enable=true, const char \*name=0, int skip=0, bool startrunning=true)
- bool **getEnabled** () const
- double **getFrequency** () const
- double **getLast** () const
- double **getTotal** () const
- double **getOverhead** () const
- int **getCount** () const
- const char \* **getName** () const
- int **getSkip** () const
- bool **getRunning** () const
- int **getVerbose** () const
- const char \* [getValue](#) (bool details=false, bool msonly=false)
- void **setVerbose** (int v)
- void **start** ()
- void **stop** ()
- void **reset** ()

### Static Public Member Functions

- static void [getValue\\_s](#) (char \*buf, int len, const char \*name, int count, double total, bool running, bool details, bool msonly)

## 14.103.1 Constructor & Destructor Documentation

### 14.103.1.1 Timer()

```
InternalZGY::Timer::Timer (
    bool enabled = true,
    const char * name = 0,
    int skip = 0,
    bool startrunning = true )
```

High resolution timer for performance measurement.

The timer starts running as soon as it is constructed, but for added accuracy it is recommended to invoke start() explicitly. The previous start time is then ignored.

After calling stop(), the formatted elapsed time is available in [getValue\(\)](#). start() and stop() may be called multiple times to accumulate statistics. In this case, it might be useful to pass a "skip" argument to the constructor telling it that the first N laps are not representative and should be ignored.

The resolution and accuracy for timing short duration calls is approx. +/- 1 microsecond. Note that the class tries to adjust for the overhead of calling the Windows performance timer. This is done heuristically, so if you time something that takes just a few nanoseconds to execute then the result could end up negative.

Applications may use [Timer](#) directly, but the higher level NamedTimer will often be more convenient. The following examples show the use:

```
// One-shot timer:
static bool enable = Timer.getNumericEnv("SALMON_TIMERS")>0;
Timer t(enable);
DoTheWork();
t.Stop();
printf("It took %s\n", t.getValue());
//Statistics timer:
static Timer t(enable);
DoTheWork();
t.Stop();
if (last_call)
    printf("It took %s\n", t.getValue());
```

The first example is often too noisy, and there is no way for the user to configure the timer to only print statistics. The second approach may be tricky because it is not always obvious when to print the gathered statistics. The higher level NamedTimer and TimerPool solves these problems. Create a new [Timer](#) instance, optionally giving it a name. Optionally pass enabled=false to create a low overhead stub. Optionally pass an initial count not to be included in statistics. Once the instance is created the enabled state cannot be changed.

## 14.103.2 Member Function Documentation

### 14.103.2.1 getValue()

```
const char * InternalZGY::Timer::getValue (
    bool details = false,
    bool msonly = false )
```

Show the elapsed time and lap count as a human readable string. The function only uses public information, so callers need not use this method if they don't like the particular formatting being done.

The result of [getValue\(\)](#) points to a class member that is overwritten at each call. So this method is definitely not thread safe. But safe to call in two different instances at the same time. The result from [getValue\\_s\(\)](#) is thread safe because caller supplies the buffer,

By default the result is shown in a "reasonable" unit. us, ms, s, etc. depending on the elapsed time. This can be confusing if printing a list of timers wanting to quickly see where the time is being spent. Pass msonly=true for those cases.

## Parameters

<i>details</i>	If true, include the timer name.
<i>msonly</i>	If true, always show result in milliseconds.

### 14.103.2.2 `getValue_s()`

```
void InternalZGY::Timer::getValue_s (
    char * result,
    int buffer_size,
    const char * name,
    int count,
    double total,
    bool running,
    bool details,
    bool msonly ) [static]
```

Show the elapsed time and lap count as a human readable string. The function only uses public information, so callers need not use this method if they don't like the particular formatting being done.

The result of [getValue\(\)](#) points to a class member that is overwritten at each call. So this method is definitely not thread safe. But safe to call in two different instances at the same time. The result from [getValue\\_s\(\)](#) is thread safe because caller supplies the buffer,

By default the result is shown in a "reasonable" unit. us, ms, s, etc. depending on the elapsed time. This can be confusing if printing a list of timers wanting to quickly see where the time is being spent. Pass `msonly=true` for those cases.

## Parameters

<i>details</i>	If true, include the timer name.
<i>msonly</i>	If true, always show result in milliseconds.

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

- [impl/timer.h](#)
- [impl/timer.cpp](#)

## 14.104 InternalZGY::TmpLookupEntry Struct Reference

### Public Attributes

- `std::uint64_t` **offset**
- `std::uint64_t` **endpos**
- `std::uint64_t` **type**
- `std::uint64_t` **ordinal**

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

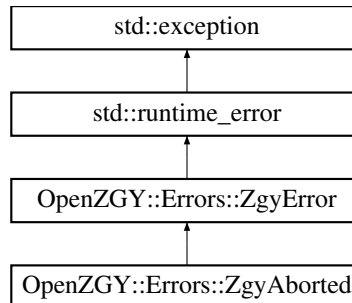
- [impl/meta.cpp](#)

## 14.105 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

#### 14.105.1 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.

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

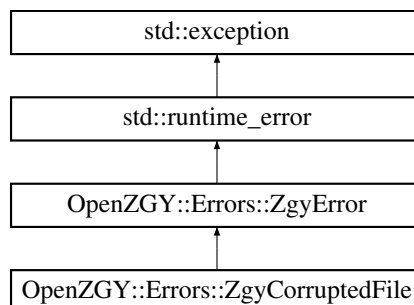
- [exception.h](#)
- [exception.cpp](#)

## 14.106 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

### 14.106.1 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.

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

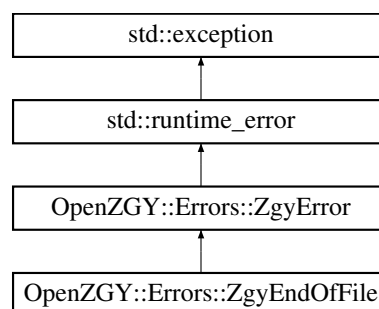
- [exception.h](#)
- [exception.cpp](#)

## 14.107 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

#### 14.107.1 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](#).

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

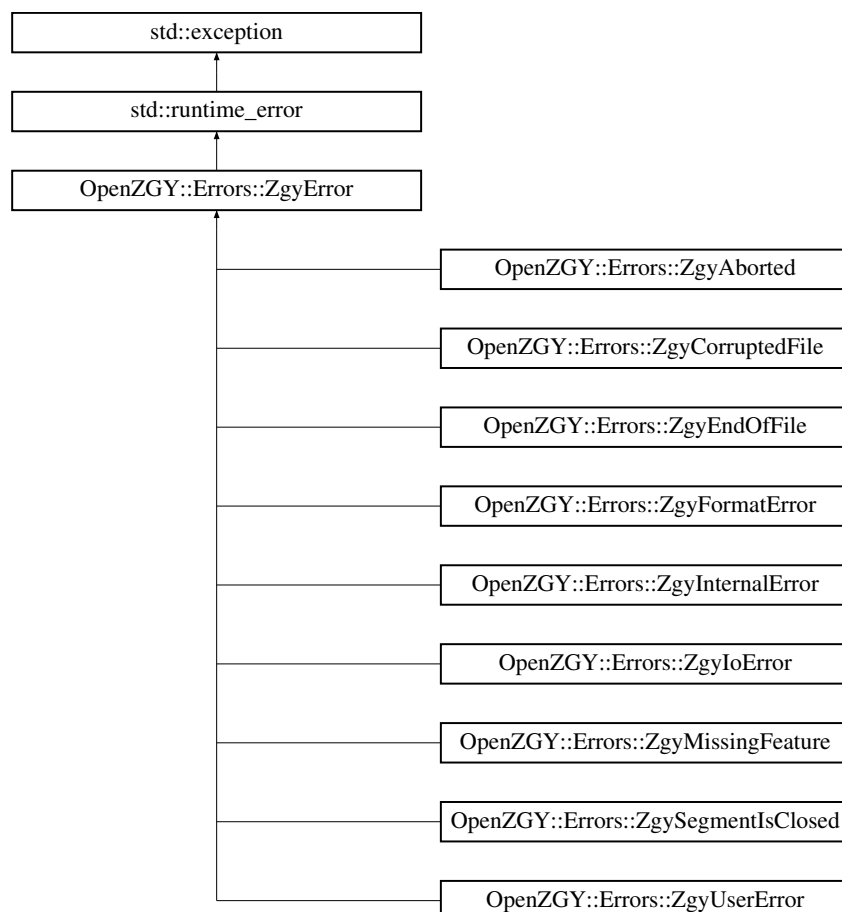
- [exception.h](#)
- [exception.cpp](#)

### 14.108 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.*

### 14.108.1 Detailed Description

Base class for all exceptions thrown by OpenZGY.

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

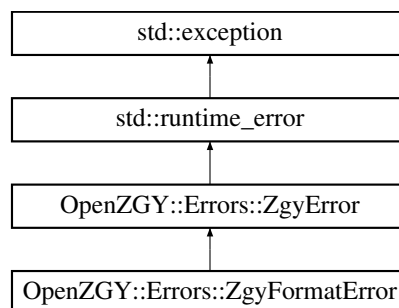
- [exception.h](#)
- [exception.cpp](#)

## 14.109 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

### 14.109.1 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.

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

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



## 14.110 InternalZGY::ZgyInternalBulk Class Reference

```
#include <bulk.h>
```

### Classes

- class [ErrorsWillCorruptFile](#)

### Public Types

- typedef std::function< bool(int, const std::string &);> **LoggerFn**

### Public Member Functions

- **ZgyInternalBulk** (const std::shared\_ptr< [FileADT](#) > &file, const std::shared\_ptr< [ZgyInternalMeta](#) > &metadata, bool compressed\_write, const LoggerFn &logger=LoggerFn())
- std::pair< bool, double > [readConstantValue](#) (const std::array< std::int64\_t, 3 > &start, const std::array< std::int64\_t, 3 > &size, int32\_t lod, bool as\_float)  
*Get hint about all constant region.*
- void [readToExistingBuffer](#) (const std::shared\_ptr< [DataBuffer](#) > &data, const std::array< std::int64\_t, 3 > &start, int32\_t lod, bool as\_float)
- std::shared\_ptr< [DataBuffer](#) > [readToNewBuffer](#) (const std::array< std::int64\_t, 3 > &start, const std::array< std::int64\_t, 3 > &size, int32\_t lod, bool as\_float, bool check\_constant)
- void **writeRegion** (const std::shared\_ptr< [DataBuffer](#) > &data, const std::array< std::int64\_t, 3 > &start, int32\_t lod, bool is\_storage, const [compressor\\_t](#) &compressor)
- bool **errorflag** () const
- void **set\_errorflag** (bool flag)
- LoggerFn **set\_logger** (const LoggerFn &logger)
- std::array< double, 2 > **valueRangeWritten** () const

### 14.110.1 Detailed Description

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

### 14.110.2 Member Function Documentation

## 14.110.2.1 readConstantValue()

```
std::pair< bool, double > InternalZGY::ZgyInternalBulk::readConstantValue (
    const std::array< std::int64_t, 3 > & start,
    const std::array< std::int64_t, 3 > & size,
    int32_t lod,
    bool as_float )
```

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.

## 14.110.2.2 readToExistingBuffer()

```
void InternalZGY::ZgyInternalBulk::readToExistingBuffer (
    const std::shared_ptr< DataBuffer > & result,
    const std::array< std::int64_t, 3 > & start,
    int32_t lod,
    bool as_float )
```

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

## 14.110.2.3 readToNewBuffer()

```
std::shared_ptr< DataBuffer > InternalZGY::ZgyInternalBulk::readToNewBuffer (
    const std::array< std::int64_t, 3 > & start,
    const std::array< std::int64_t, 3 > & size,
    int32_t lod,
    bool as_float,
    bool check_constant )
```

Read bulk data starting at "start" in index space size "size". Return the result in a newly allocated [DataBuffer](#). The result will either be a scalar (constant-value) buffer or a regular buffer.

Pass check\_constant=true to check extra hard for all-constant data. A region written with all samples identical, as opposed to a region flagged as constant without taking up space, will also be detected.

Start should be in the range (0,0,0) to Size-1. It is valid to pass a size that includes the padding area between the survey and the end of the current brick. But not past that point.

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

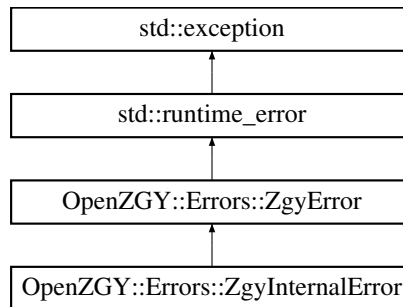
- impl/[bulk.h](#)
- impl/[bulk.cpp](#)

### 14.111 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

##### 14.111.1 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 files:

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

### 14.112 InternalZGY::ZgyInternalMeta Class Reference

```
#include <meta.h>
```

#### Classes

- class [ErrorsWillCorruptFile](#)

## Public Types

- typedef std::function< bool(int, const std::string &);> **LoggerFn**

## Public Member Functions

- const [IFileHeaderAccess](#) & **fh** () const
- const [IInfoHeaderAccess](#) & **ih** () const
- const [IHistHeaderAccess](#) & **hh** () const
- const [ILookupTableAccess](#) & **alup** () const
- const [ILookupTableAccess](#) & **blup** () const
- [IInfoHeaderAccess](#) & **ih** ()
- [IHistHeaderAccess](#) & **hh** ()
- [ILookupTableAccess](#) & **alup** ()
- [ILookupTableAccess](#) & **blup** ()
- **ZgyInternalMeta** (const [ZgyInternalMeta](#) &)=delete
- [ZgyInternalMeta](#) & **operator=** (const [ZgyInternalMeta](#) &)=delete
- **ZgyInternalMeta** (const std::shared\_ptr< [FileADT](#) > &file, const LoggerFn &logger=LoggerFn())
- **ZgyInternalMeta** (const [ZgyInternalWriterArgs](#) &args, const LoggerFn &logger=LoggerFn())
- void **dump** (std::ostream &out, const std::string &prefix="")
- bool **errorflag** () const
- void **set\_errorflag** (bool flag)
- LoggerFn **set\_logger** (const LoggerFn &logger)
- void **flushMeta** (const std::shared\_ptr< [FileADT](#) > &file)

### 14.112.1 Detailed Description

Holds references to all the individual headers needed to access ZGY.

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

- impl/[meta.h](#)
- impl/meta.cpp

### 14.113 InternalZGY::ZgyInternalWriterArgs Class Reference

Internal counterpart to [OpenZGY::ZgyWriterArgs](#).

```
#include <meta.h>
```

### Public Attributes

- `std::string filename`
- `std::array< std::int64_t, 3 > size`
- `std::array< std::int64_t, 3 > bricksize`
- [RawDataType](#) `datatype`
- `std::array< float, 2 > datarange`
- [RawVerticalDimension](#) `zunitdim`
- [RawHorizontalDimension](#) `hunitdim`
- `std::string zunitname`
- `std::string hunitname`
- `double zunitfactor`
- `double hunitfactor`
- `float zstart`
- `float zinc`
- `std::array< float, 2 > annotstart`
- `std::array< float, 2 > annotinc`
- `std::array< std::array< double, 2 >, 4 > corners`

### 14.113.1 Detailed Description

Internal counterpart to [OpenZGY::ZgyWriterArgs](#).

As with enum types, code in the api level (i.e. [api.cpp](#)) knows how to create one of these instances from a `Zgy↔WriterArgs`. Conversion in the other direction is not useful.

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.

Compared to `ZgyWriterArgs` there are some members missing here because they are processed at a higher level. These are:

- `template`
- `iocontext`
- `compressor`
- `lodcompressor`

The following enums need to be mapped from api types to internal.

- `datatype`
- `zunitdim`
- `hunitdim`.

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

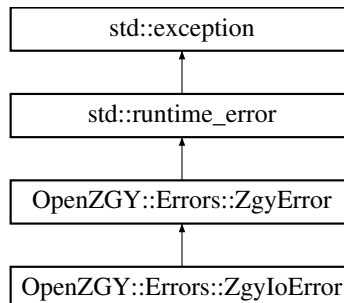
- `impl/meta.h`

### 14.114 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

##### 14.114.1 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 [ZgyloError](#).

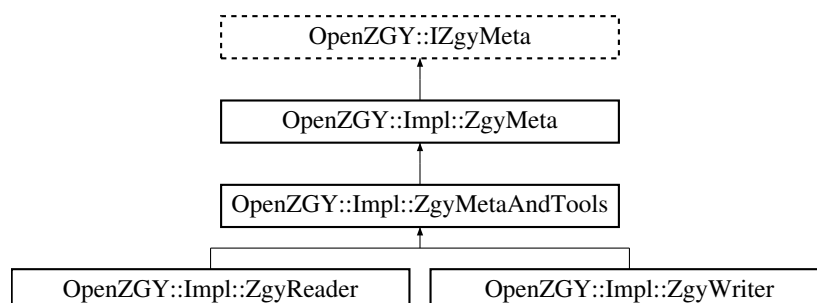
The documentation for this class was generated from the following files:

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

### 14.115 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 `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 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

### 14.115.1 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 constructing 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](#).

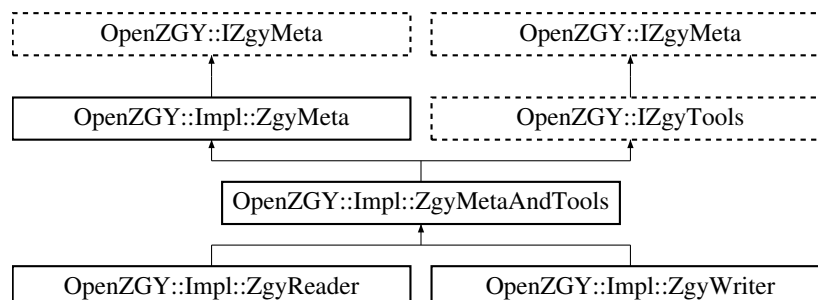
The documentation for this class was generated from the following file:

- [api.cpp](#)

### 14.116 `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

### 14.116.1 Detailed Description

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

### 14.116.2 Member Function Documentation

#### 14.116.2.1 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](#).

#### 14.116.2.2 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:

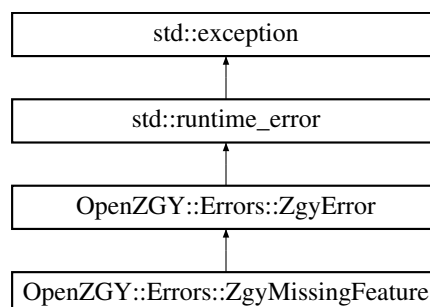
- [api.cpp](#)

## 14.117 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

### 14.117.1 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.

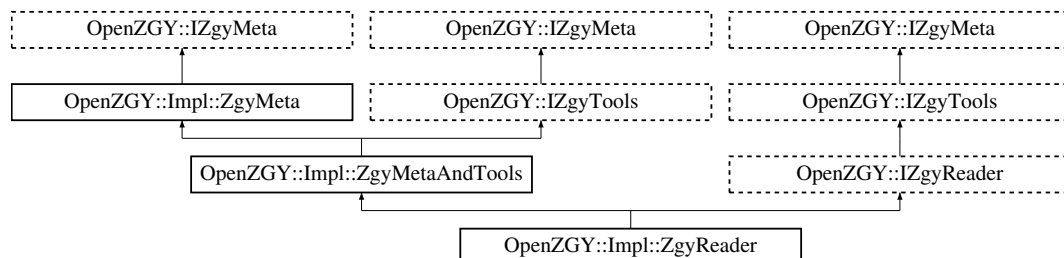
The documentation for this class was generated from the following files:

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

## 14.118 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

### 14.118.1 Detailed Description

Concrete implementation of [IZgyReader](#).

### 14.118.2 Constructor & Destructor Documentation

#### 14.118.2.1 ZgyReader()

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

Open a ZGY file for reading.

### 14.118.3 Member Function Documentation

#### 14.118.3.1 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](#).

#### 14.118.3.2 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.

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](#).

### 14.118.3.3 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](#).

### 14.118.3.4 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](#).

### 14.118.3.5 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:

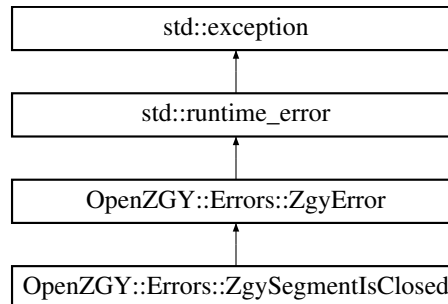
- [api.cpp](#)

### 14.119 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

##### 14.119.1 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 files:

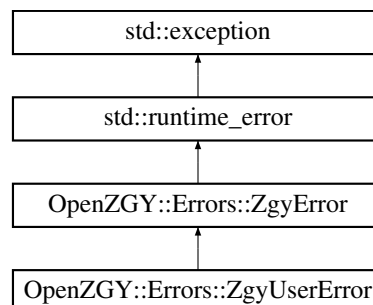
- [exception.h](#)
- [exception.cpp](#)

### 14.120 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

### 14.120.1 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.

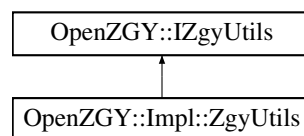
The documentation for this class was generated from the following files:

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

## 14.121 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

### 14.121.1 Detailed Description

Concrete implementation of [IZgyUtils](#).

### 14.121.2 Constructor & Destructor Documentation

#### 14.121.2.1 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.

## 14.121.3 Member Function Documentation

### 14.121.3.1 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](#).

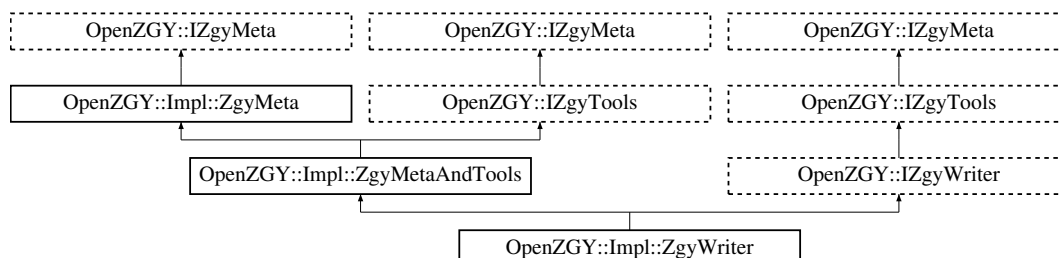
The documentation for this class was generated from the following file:

- [api.cpp](#)

## 14.122 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\\_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

#### 14.122.1 Detailed Description

Concrete implementation of [IZgyWriter](#).

#### 14.122.2 Constructor & Destructor Documentation

##### 14.122.2.1 ZgyWriter()

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

Create a ZGY file and open it for writing.

## 14.122.2.2 ~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.

## 14.122.3 Member Function Documentation

### 14.122.3.1 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](#).

### 14.122.3.2 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](#).

### 14.122.3.3 errorflag()

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

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.

Implements [OpenZGY::IZgyWriter](#).

### 14.122.3.4 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. 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.

Implements [OpenZGY::IZgyWriter](#).

### 14.122.3.5 set\_errorflag()

```
void OpenZGY::Impl::ZgyWriter::set_errorflag (
```

```
bool value ) [override], [virtual]
```

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

Implements [OpenZGY::IZgyWriter](#).

#### 14.122.3.6 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](#).

#### 14.122.3.7 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](#).

#### 14.122.3.8 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](#).

### 14.122.3.9 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](#).

### 14.122.3.10 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](#).

### 14.122.3.11 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](#)

## 14.123 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** (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** & **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.*

## 14.123.1 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});
```

## 14.123.2 Member Function Documentation

### 14.123.2.1 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.

## 14.123.2.2 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.

## 14.123.2.3 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.

## 14.123.2.4 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".

## 14.123.2.5 filename()

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

Set file to open.

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

## 14.123.2.6 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.

#### 14.123.2.7 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.

#### 14.123.2.8 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.

#### 14.123.2.9 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.

#### 14.123.2.10 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.

## 14.123.2.11 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.

## 14.123.2.12 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).

## 14.123.2.13 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.

## 14.123.2.14 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.

### 14.123.2.15 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.

### 14.123.2.16 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.

### 14.123.2.17 zinc()

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

Set increment (distance between samples) in vertical direction.

Vertical annotation is generally safe to have non-integral.

### 14.123.2.18 zstart()

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

Set first time/depth.

Vertical annotation is generally safe to have non-integral.

### 14.123.2.19 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](#)

# Alpha version. Expect changes.

## Chapter 15

## File Documentation

### 15.1 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::EnumMapper](#)  
*convert between internal and external data types.*
- 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.*

### 15.1.1 Detailed Description

Implements the pure interfaces of the API.

## 15.2 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 "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.*
- [InternalZGY](#)  
*Implementation not visible to clients.*

## 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.*

## 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.*

### 15.2.1 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 debuggig harder.

## 15.3 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)

### 15.3.1 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
```

## 15.4 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](#).*

#### 15.4.1 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.

## 15.5 impl/arrayops.h File Reference

```
#include <array>
#include <ostream>
```

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

#### 15.5.1 Detailed Description

Extend `std::array<>` with numeric types to support arithmetic operations between two instances (operations done per element) and between an instance and a scalar (the scalar operates on each element).

- `a @ b` – implemented for `+, -, *, /`
- `a = b` – NOT implemented
- `a @ scalar` – implemented for `+, -, *, /`
- `a = scalar` – NOT implemented
- `scalar @ a` – NOT implemented

TODO-Low: Should I write a new class and add behavior to that instead of trying to "improve" `std::array`? In particular the question is relevant because I might only ever need `std::array<std::int64_t, 3>` so my new class won't even need to be templated. The downside is that this type may end up visible in the API and I don't want to introduce yet another custom defined type if I can help it.

To make use of these operators you will need to use:

```
using namespace InternalZGY::ArrayOps;
```

## 15.6 impl/bulk.h File Reference

Bulk data read/write.

```
#include "../declspec.h"
#include "enum.h"
#include <memory>
#include <functional>
#include <array>
#include <vector>
#include <tuple>
#include <sstream>
```

### Classes

- class [InternalZGY::ZgyInternalBulk](#)

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### 15.6.1 Detailed Description

Bulk data read/write.

## 15.7 impl/compress\_null.cpp File Reference

Example code for the compression plug-in mechanism.

```
#include "compression.h"
#include "../exception.h"
```

### Classes

- class [InternalZGY::NullCompressPlugin](#)  
*Example compression plug-in that always fails to compress.*
- class [InternalZGY::NullCompressPlugin::Register](#)  
*Register the compress and decompress functions in the factory.*

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### 15.7.1 Detailed Description

Example code for the compression plug-in mechanism.

A real implementation will typically use Doxygen's copydoc statements instead of copyint the overly verbose explanations in this file.

## 15.8 impl/compression.cpp File Reference

Compression factory.

```
#include "compression.h"
#include "../exception.h"
#include <algorithm>
#include <sstream>
```

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### 15.8.1 Detailed Description

Compression factory.

## 15.9 impl/compression.h File Reference

Factory for handling compression.

```
#include <string>
#include <vector>
#include <memory>
#include <functional>
#include <map>
#include "../declspec.h"
#include "enum.h"
```

### Classes

- class [InternalZGY::CompressFactoryImpl](#)

*Registry of known compress and decompress algorithms.*

## Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### 15.9.1 Detailed Description

Factory for handling compression.

## 15.10 impl/cornerpoints.h File Reference

Deprecated. See [transform.h](#).

```
#include "../declspec.h"
#include <stddef.h>
#include <array>
#include <cstdint>
```

## Classes

- class [InternalZGY::OrderedCornerPoints](#)
- struct [InternalZGY::OrderedCornerPoints::Element](#)

## Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### 15.10.1 Detailed Description

Deprecated. See [transform.h](#).

## 15.11 impl/databuffer.h File Reference

Each DataBuffer instance represents some in memory data.

```
#include "../declspec.h"
#include "enum.h"
#include "types.h"
#include <cstdint>
#include <array>
#include <memory>
#include <string>
```

## Classes

- class [InternalZGY::DataBuffer](#)  
*Each [DataBuffer](#) instance represents some in memory data.*
- class [InternalZGY::DataBufferNd< T, NDim >](#)

## Namespaces

- [InternalZGY](#)  
*Implementation not visible to clients.*

### 15.11.1 Detailed Description

Each [DataBuffer](#) instance represents some in memory data.

## 15.12 impl/enum.h File Reference

enums and type aliases not visible to the public API.

```
#include <array>
#include <cstdint>
#include <memory>
#include <functional>
```

## Namespaces

- [InternalZGY](#)  
*Implementation not visible to clients.*

## Typedefs

- typedef std::array< std::int64\_t, 3 > [InternalZGY::index3\\_t](#)  
*type equivalent to std::int64\_t[3]*
- typedef std::pair< std::shared\_ptr< const void >, std::int64\_t > [InternalZGY::rawdata\\_t](#)  
*Shared data plus size. No other information.*
- typedef std::function< rawdata\_t(const rawdata\_t &, const index3\_t &)> [InternalZGY::compressor\\_t](#)  
*Function for compressing a brick.*

## Enumerations

- enum [InternalZGY::RawDataType](#) {  
    **SignedInt8** = 0, **UnsignedInt8** = 1, **SignedInt16** = 2, **UnsignedInt16** = 3,  
    **SignedInt32** = 4, **UnsignedInt32** = 5, **Float32** = 6, **IbmFloat32** = 7 }
- enum [InternalZGY::RawCoordType](#) {  
    **Unknown** = 0, **Meters** = 1, **Feet** = 2, **ArcSec** = 3,  
    **ArcDeg** = 4, **ArcDegMinSec** = 5 }
- enum [InternalZGY::RawHorizontalDimension](#) { **Unknown** = 0, **Length** = 1, **ArcAngle** = 2 }
- enum [InternalZGY::RawVerticalDimension](#) { **Unknown** = 0, **Depth** = 1, **SeismicTWT** = 2, **SeismicOWT** = 3 }
- enum [InternalZGY::RawGridDefinition](#) { **Unknown** = 0, **Parametric** = 1, **ThreePoint** = 2, **FourPoint** = 3 }
- enum [InternalZGY::BrickStatus](#) { **Missing** = 0, **Constant** = 1, **Normal** = 2, **Compressed** = 3 }
- enum [InternalZGY::UpdateMode](#) { **Never** = 0, **Constant** = 1, **Always** = 4, **Pedantic** = 5 }

## 15.12.1 Detailed Description

enums and type aliases not visible to the public API.

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

## 15.13 impl/environment.h File Reference

Reading and writing environment variables.

```
#include "../declspec.h"
#include <string>
```

### Classes

- class [InternalZGY::Environment](#)
- class [InternalZGY::PushEnvironment](#)

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

## 15.13.1 Detailed Description

Reading and writing environment variables.

## 15.14 impl/file.h File Reference

Low level I/O, abstract layer.

```
#include <stdint>
#include <vector>
#include <string>
#include <memory>
#include <functional>
#include "../declspec.h"
#include "timer.h"
```



## Classes

- class [InternalZGY::Config](#)
- class [InternalZGY::FileConfig](#)
- class [InternalZGY::ReadRequest](#)
- class [InternalZGY::FileADT](#)
- class [InternalZGY::FileFactory](#)
- class [InternalZGY::FileCommon](#)

*Implementation of some methods that might be shared.*

## Namespaces

- [OpenZGY](#)

*The entire public API is in this namespace.*

- [InternalZGY](#)

*Implementation not visible to clients.*

## Typedefs

- typedef std::vector< ReadRequest > **InternalZGY::ReadList**
- typedef std::vector< ReadList > **InternalZGY::ReadDoubleList**

## Enumerations

- enum **OpenMode** { **Closed** = 0, **ReadOnly**, **ReadWrite**, **Truncate** }
- enum **UsageHint** {  
    **Unknown** = 0x00, **TextFile** = 0x01, **Header** = 0x10, **Data** = 0x20,  
    **Compressed** = 0x40, **Mixed** = 0x40 }

### 15.14.1 Detailed Description

Low level I/O, abstract layer.

This file contains the base class for low level I/O either to on-prem data using the regular read and write methods of the OS or to a cloud back-end.

```
InternalZGY::Config:
InternalZGY::FileConfig(Config):
InternalZGY::SDConfig(Config):
    Details such as user credentials etc. established when the
    file is open. Specific to the backend type.
    Note that there is currently no way to pass a configuration
    object along with every read and write request. This might
    have been useful for a server type application but would
    require the config parameter to ripple across at least 50
    existing methods. I doubt this would be worth the trouble.
InternalZGY::FileADT: <=== file.h
InternalZGY::LocalFile(FileADT): <=== file_local.h
InternalZGY::LocalFileOther(LocalFile): <=== file_local.h
InternalZGY::LocalFileLinux(LocalFile): <=== file_local.h
InternalZGY::SeismicStoreFile(FileADT): <=== file_sd.h
InternalZGY::SeismicStoreFileDelayedWrite(FileADT): <=== file_sd.h
    Higher level code should only access the polymorphic FileADT
    base class and the InternalZGY::FileFactory that creates an
    instance of the desired type.
```

## 15.15 impl/file\_local.cpp File Reference

Low level I/O, regular files.

```
#include "file.h"
#include "../exception.h"
#include "timer.h"
#include "environment.h"
#include <vector>
#include <string>
#include <memory>
#include <functional>
#include <iostream>
#include <sstream>
#include <algorithm>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
```

### Classes

- class [InternalZGY::LocalFileLinux](#)

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### 15.15.1 Detailed Description

Low level I/O, regular files.

## 15.16 impl/file\_sd.h File Reference

Low level I/O utilities, Seismic Store.

```
#include "file.h"
```

### Classes

- class [InternalZGY::FileUtilsSeismicStore](#)

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

#### 15.16.1 Detailed Description

Low level I/O utilities, Seismic Store.

This class declares an interface for utility functions specific to one backend. The actual implementation is in some derived FileADT class, allowing us to use the same factory mechanism to access the functions.

It is also convenient because several of the methods require an open file handle. Even if not associated with a particular file. The application can open a long lived file handle that represents the seismic store connections with credentials etc. and just use that instance for performing seismic store related operations. For this reasons most of the methods are declared const. They don't modify the file handle itself.

## 15.17 impl/genlod.h File Reference

Generate low resolution bricks.

```
#include "enum.h"
#include "lodalgo.h"
#include "histogramdata.h"
#include "histogrambuilder.h"
#include "statisticdata.h"
#include "databuffer.h"
#include "../exception.h"
#include <memory>
#include <stdint>
#include <array>
#include <vector>
#include <functional>
#include <iostream>
```

### Classes

- class [InternalZGY::GenLodBase](#)
- class [InternalZGY::GenLodImpl](#)
- class [InternalZGY::GenLodC](#)

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

#### 15.17.1 Detailed Description

Generate low resolution bricks.

## 15.18 impl/guid.h File Reference

Simplified GUID handling. Only big endian, random number guids.

```
#include <cstdint>
#include <cstddef>
#include <array>
#include <string>
#include <ostream>
```

### Classes

- class [InternalZGY::GUID](#)

*Simplified GUID handling. Only big endian, random number guids.*

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### Functions

- `std::ostream & InternalZGY::Formatters::operator<< (std::ostream &os, const ::InternalZGY::GUID &guid)`

### 15.18.1 Detailed Description

Simplified GUID handling. Only big endian, random number guids.

## 15.19 impl/histogrambuilder.h File Reference

Collect statistics and histogram for bulk data.

```
#include "../declspec.h"
#include "statisticdata.h"
#include "histogramdata.h"
#include <cmath>
#include <limits>
#include <iterator>
#include <assert.h>
```

### Classes

- class [InternalZGY::HistogramBuilder](#)

*Collect statistics and histogram for bulk data.*

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

#### 15.19.1 Detailed Description

Collect statistics and histogram for bulk data.

## 15.20 impl/histogramdata.h File Reference

provides class [InternalZGY::HistogramData](#).

```
#include "../declspec.h"
#include "roundandclip.h"
#include <memory>
#include <math.h>
#include <cmath>
```

### Classes

- class [InternalZGY::HistogramData](#)

*A histogram for a data set.*

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

#### 15.20.1 Detailed Description

provides class [InternalZGY::HistogramData](#).

## 15.21 impl/iltf2d.h File Reference

Deprecated. See [transform.h](#).

### Classes

- class [InternalZGY::ImplicitLinearTransform2d](#)
- struct [InternalZGY::ImplicitLinearTransform2d::TiePoint](#)

## Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### 15.21.1 Detailed Description

Deprecated. See [transform.h](#).

## 15.22 impl/lodalgo.h File Reference

Decimation algorithms to output low resolution bricks.

```
#include "../declspec.h"
#include "enum.h"
#include <stdint>
#include <memory>
#include <array>
```

## Classes

- class [InternalZGY::DataBufferNd< T, NDim >](#)

## Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

## Enumerations

- enum [InternalZGY::LodAlgorithm](#) {  
    [InternalZGY::LodAlgorithm::LowPass](#) = 0, [InternalZGY::LodAlgorithm::WeightedAverage](#), [InternalZGY::LodAlgorithm::Average](#),  
    [InternalZGY::LodAlgorithm::Median](#),  
    [InternalZGY::LodAlgorithm::Minimum](#), [InternalZGY::LodAlgorithm::Maximum](#), [InternalZGY::LodAlgorithm::MinMax](#),  
    [InternalZGY::LodAlgorithm::Decimate](#),  
    [InternalZGY::LodAlgorithm::DecimateSkipNaN](#), [InternalZGY::LodAlgorithm::DecimateRandom](#), [InternalZGY::LodAlgorithm::All](#),  
    [InternalZGY::LodAlgorithm::WhiteNoise](#),  
    [InternalZGY::LodAlgorithm::MostFrequent](#), [InternalZGY::LodAlgorithm::MostFrequentNon0](#), [InternalZGY::LodAlgorithm::Average](#)  
}

### Functions

- void [InternalZGY::createLod](#) (const std::shared\_ptr< DataBuffer > &result, const std::shared\_ptr< const DataBuffer > &input, LodAlgorithm algorithm, const std::int64\_t \*hist, std::int32\_t bincount, double histogram\_min, double histogram\_max)

*Main entry point for low resolution compute.*

- void [InternalZGY::createLodMT](#) (const std::shared\_ptr< DataBuffer > &result, const std::shared\_ptr< const DataBuffer > &input, LodAlgorithm algorithm, const std::int64\_t \*hist, std::int32\_t bincount, double histogram\_min, double histogram\_max)

*Main entry point for low resolution compute.*

- void [InternalZGY::createLodST](#) (const std::shared\_ptr< DataBuffer > &result, const std::shared\_ptr< const DataBuffer > &input, LodAlgorithm algorithm, const std::int64\_t \*hist, std::int32\_t bincount, double histogram\_min, double histogram\_max)

*Main entry point for low resolution compute.*

### 15.22.1 Detailed Description

Decimation algorithms to output low resolution bricks.

## 15.23 impl/lodsampling.h File Reference

Static methods for downsampling. Used by lodalgo.cpp only.

### Classes

- class [InternalZGY::LodSampling](#)

*Static methods for downsampling. Used by lodalgo.cpp only.*

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### 15.23.1 Detailed Description

Static methods for downsampling. Used by lodalgo.cpp only.

## 15.24 impl/logger.h File Reference

Logging framework.

```
#include "../declspec.h"
#include <string>
#include <sstream>
#include <functional>
```

## Classes

- class [InternalZGY::LoggerBase](#)
- class [InternalZGY::Logger](#)

## Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

## Functions

- OPENZGY\_API bool **InternalZGY::newlogger** (int priority, const std::string &str)
- OPENZGY\_API bool **InternalZGY::newlogger** (int priority, const std::ios &ss)

### 15.24.1 Detailed Description

Logging framework.

See [InternalZGY::LoggerBase](#) for details.

## 15.25 impl/lookuptable.h File Reference

Static methods to assist working with lookup tables.

```
#include "enum.h"
#include <cstdint>
#include <vector>
#include <array>
#include <string>
```

## Classes

- class [InternalZGY::LookupTable](#)  
*Static methods to assist working with lookup tables.*
- struct [InternalZGY::LookupTable::LutInfo](#)  
*Decoded contents of the lookup table for one brick or tile.*

## Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*



### 15.25.1 Detailed Description

Static methods to assist working with lookup tables.

## 15.26 impl/meta.h File Reference

Meta data read/write.

```
#include "../declspec.h"
#include "../exception.h"
#include "file.h"
#include "enum.h"
#include "lookuptable.h"
#include <stdint>
#include <memory>
#include <ostream>
#include <sstream>
#include <iomanip>
#include <vector>
#include <cstring>
```

### Classes

- class [InternalZGY::ZgyInternalWriterArgs](#)  
*Internal counterpart to [OpenZGY::ZgyWriterArgs](#).*
- class [InternalZGY::IHeaderAccess](#)
- class [InternalZGY::IFileHeaderAccess](#)
- class [InternalZGY::IOffsetHeaderAccess](#)
- class [InternalZGY::IInfoHeaderAccess](#)
- class [InternalZGY::IHistHeaderAccess](#)
- class [InternalZGY::ILookupTableAccess](#)
- class [InternalZGY::HeaderAccessFactory](#)
- class [InternalZGY::ZgyInternalMeta](#)

### Namespaces

- [InternalZGY](#)  
*Implementation not visible to clients.*

## 15.26.1 Detailed Description

Meta data read/write.

The meta data in an [OpenZGY](#) file is broken down into multiple headers. In this file, a set of classes for each header type is responsible for reading and decoding that header. A lot of the code is inlined to make it easier to maintain. Keeping everything that needs to be updated close together.

```
class FooHeaderV1POD
class FooHeaderV1POD etc.
```

- POD class corresponding exactly to what is stored on the file. A new class needs to be created each time the version changes. The compiler gets to decide the actual layout, but is instructed to not use any padding. Hopefully this makes the layout platform independant.

```
class IFooHeaderAccess : public IHeaderAccess
```

- Pure interface, more or less matching the layout of the most recent version of the POD. The purpose is to hide differences between versions when reading. Writing and updating need not be handled the same way. Those are simpler. Creating a new file only needs to worry about the latest version, and updating meta data is only allowed on very few (currently none at all) attributes.

```
class FooHeaderAccess : public IHeaderAccess
```

- Adds a concrete dump() method for debugging.

```
class FooHeaderV1Access : public FooHeaderAccess
class FooHeaderV2Access : public FooHeaderAccess
etc.
```

- Aggregates (i.e. does not inherit from) the corresponding POD. Implements the interface that allows version independant access of the information on file. Where there is a mismatch the code should compute the result on the fly if possible. The last resort is to cache data in the virtual compute() method and storing the result in this instance instead if the pod. Note that this gets messy if the cached data can become stale.
- TODO-WARNING, the current implementation assumes that the computer architecture and the C++ compiler allows unaligned access. According to the C++ standard this gives **UNDEFINED BEHAVIOR** even on architectures such as x86\_64 that do allow unaligned loads. On newer x86\_64 processors this is actually fairly efficient too. BUT the compiler is free to use SSE instructions for load/store and those do NOT allow unaligned pointers. <https://pzentsov.github.io/2016/11/06/bug-story-alignment-on-x86.html>

```
HeaderAccessFactory::createFoo(std::uint32_t version)
```

- Construct and return the appropriate FooHeaderV?Access instance.

This file was initially generated with some help from `openzgy.tools.cppmeta`. But that was only a starting point. If major changes are needed to the format (will hopefully never happen) then the code generator might still be useful. otherwise this file is expected to be maintained by hand.

## 15.27 impl/roundandclip.h File Reference

Conversion between scalar types.

```
#include <stdint>
#include <cmath>
#include <limits>
```

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### Functions

- `template<typename T >`  
`bool InternalZGY::IsFiniteT (T)`
- `template<> bool InternalZGY::IsFiniteT< float > (float value)`
- `template<> bool InternalZGY::IsFiniteT< double > (double value)`
- `template<> bool InternalZGY::IsFiniteT< long double > (long double value)`
- `template<typename T >`  
`bool InternalZGY::IsNanT (T)`
- `template<> bool InternalZGY::IsNanT< float > (float value)`
- `template<> bool InternalZGY::IsNanT< double > (double value)`
- `template<> bool InternalZGY::IsNanT< long double > (long double value)`
- `template<typename T >`  
`T InternalZGY::RoundAndClip (double value)`
- `template<typename T >`  
`T InternalZGY::RoundAndClip (double value, T nan)`

### 15.27.1 Detailed Description

Conversion between scalar types.

The original code in the ZGY library and in Salmon went a bit overboard with optimizations. Up to and including inline assembly code. It is possible to bring this back, but not until it can be verified that those hacks are still relevant with current compilers.

Note: Use the more portable `std::isnan()` in `<cmath>` for all platforms. The bare "isnan" might not exist on all platforms, and/or it might be a macro which gets undefined if `<cmath>` happens to be included elsewhere. For similar reasons use `std::isfinite` unless it turns out to be proveably slower than the alternatives.

## 15.28 impl/statisticdata.h File Reference

provides class [InternalZGY::StatisticData](#).

```
#include "../declspec.h"
#include <stdint>
#include <math.h>
#include <cmath>
```

## Classes

- class [InternalZGY::StatisticData](#)  
*Holds the result of computing statistics.*

## Namespaces

- [InternalZGY](#)  
*Implementation not visible to clients.*

### 15.28.1 Detailed Description

provides class [InternalZGY::StatisticData](#).

## 15.29 impl/structaccess.h File Reference

Low level tools for data conversion.

```
#include "../declspec.h"
#include <array>
#include <sstream>
#include <iomanip>
#include <cstring>
#include <type_traits>
```

## Namespaces

- [InternalZGY](#)  
*Implementation not visible to clients.*

## Functions

- `template<typename T, std::size_t N>`  
`std::array< T, N > InternalZGY::ptr\_to\_array (const T *in)`
- `template<typename T >`  
`T InternalZGY::align (const T &in)`
- `template<typename T, std::size_t N>`  
`std::string InternalZGY::array\_to\_string (const std::array< T, N > &a)`
- `template<typename T, std::size_t N>`  
`std::string InternalZGY::array\_to\_hex (const std::array< T, N > &a)`
- `template<typename T, std::size_t N>`  
`std::string InternalZGY::ptr\_to\_string (const T *a)`
- `template<typename T, std::size_t N>`  
`std::string InternalZGY::ptr\_to\_hex (const T *a)`
- `OPENZGY_TEST_API void InternalZGY::byteswapV1Long (std::int64_t *ptr, size_t n=1)`
- `OPENZGY_TEST_API void InternalZGY::byteswapV1Long (std::uint64_t *ptr, size_t n=1)`
- `template<typename T, typename U, int N>`  
`std::array< T, N > InternalZGY::array\_cast (const std::array< U, N > &in)`

## 15.29.1 Detailed Description

Low level tools for data conversion.

## 15.30 impl/subtiling.cpp File Reference

Handle 8x8 subtiling.

```
#include "subtiling.h"
#include <stdint>
#include <array>
#include <string.h>
```

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### Functions

- void [InternalZGY::subtiling](#) (const std::array< std::int64\_t, 3 > &bricksizes, std::int64\_t itemsize, void \*dst, const void \*src, bool remove)

## 15.30.1 Detailed Description

Handle 8x8 subtiling.

Version 1 of the uncompressed ZGY format reorganized the contents of alpha tiles and data bricks before writing. Instead of keeping the 64x64(x64) layout, they were changed to consisting of 8x8 subtiles/bricks each holding 8x8(x64) samples. This was done to provide faster access to inline/crossline sections, as the reader would then not need to load full 64x64(x64) tiles/bricks, but could concentrate only on the much smaller (1/64th) 8x8(x64) subtiles/bricks that intersected the section of interest.

Starting with version 2, such subtiling is no longer done.

When reading and writing files of version 1, the 8x8 subtiling is handled by treating the data as being of higher dimension and using a general permutation function. The rationale is as follows.

Consider a brick of size N[3]. In the two first dimensions it is subtiled with tiles of size n[2]. The offset of sample i[3] is then:

```
offset(i) = (i[0] DIV n[0]) * (n[0]*N[1]*N[2])
           + (i[1] DIV n[1]) * (n[0]*n[1]*N[2])
           + (i[0] MOD n[0]) * (n[1]*N[2])
           + (i[1] MOD n[1]) * (N[2])
           + (i[2])             * (1)
```

where the two first lines correspond to indexing a specific subtile, and the last three lines correspond to indexing within the subtile.

For the non-subtiled variant of the brick, the offset of sample  $i[3]$  is simply:

```
offset(i) = i[0]*N[1]*N[2] + i[1]*N[2] + i[2]
```

This can be rearranged into

```
offset(i) = (i[0] DIV n[0]) * (n[0]*N[1]*N[2])
           + (i[1] DIV n[1]) * (n[1]*N[2])
           + (i[0] MOD n[0]) * (N[1]*N[2])
           + (i[1] MOD n[1]) * (N[2])
           + (i[2])           * (1)
```

which takes the same form as the offset for the subtiled brick.

Both cases can now be viewed as finding the offset of sample  $x[5]$  in a 5- dimensional dataset of size  $\{ N[0]/n[0], N[1]/n[1], n[0], n[1], N[2] \}$  with stride  $S[5]$ :

```
offset(x) = x[0]*S[0] + x[1]*S[1] + x[2]*S[2] + x[3]*S[3] + x[4]*S[4]
```

where the indices in both cases are defined by

```
x = { i[0] DIV n[0], i[1] DIV n[1], i[0] MOD n[0], i[1] MOD n[1], i[2] }
```

and the strides are

```
S = { n[0]*N[1]*N[2], n[0]*n[1]*N[2], n[1]*N[2], N[2], 1 }
S = { n[0]*N[1]*N[2], n[1]*N[2], N[1]*N[2], N[2], 1 }
```

for the subtiled and non-subtiled bricks, respectively.

With only strides differing (in zero-based element 1 and 2), we can then convert between the two by applying a general 5-dimensional permutation. Since the last two strides are equal, we can in practice achieve the correct permutation by using only 4-dimensions. This corresponds to treating each vertical plane of the subtiles as a 1-dimensional array.

## 15.31 impl/subtiling.h File Reference

Handle 8x8 subtiling.

```
#include <stdint>
#include <array>
```

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

## Functions

- void [InternalZGY::subtiling](#) (const std::array< std::int64\_t, 3 > &bricksizes, std::int64\_t itemsize, void \*dst, const void \*src, bool remove)

### 15.31.1 Detailed Description

Handle 8x8 subtiling.

## 15.32 impl/timer.h File Reference

Easy to use performance measurement.

```
#include "../declspec.h"
```

## Classes

- class [InternalZGY::Timer](#)
- class [InternalZGY::PrintingTimer](#)  
*Timer that prints its result when going out of scope.*
- class [InternalZGY::RawPrintingTimer](#)  
*Timer that prints its result when going out of scope.*
- class [InternalZGY::SummaryTimer](#)  
*Hold the timing results from zero or more [Timer](#) instances.*
- class [InternalZGY::SummaryPrintingTimer](#)  
*SummaryTimer that prints its result when going out of scope.*
- class [InternalZGY::SimpleTimer](#)  
*Timer that knows where to store the result.*

## Namespaces

- [InternalZGY](#)  
*Implementation not visible to clients.*

### 15.32.1 Detailed Description

Easy to use performance measurement.

The Timer class is implemented using basic, old style techniques. The goal is to make it as safe as possible, usable also if the stop & print happens in a static destructor where stdio and (if using .NET) the CLR might be shut down.

It is also a goal to avoid as many includes as possible in the header file, so as to not introduce additional dependencies. There are a few enhancements that have been rejected because of this. I might need to revisit those, though.

The SummaryTimer and derived classes loosen these requirements somewhat. A pimpl is used (since those classes are less likely to be performance critical) and atomics are used to make add() threadsafe.

- Allowing `#include<functional>` in the header would make PrintingTimer more general. Or maybe template PrintingTimer on the functor that does the printing and the data type of the counters (to hide atomic). `typedef PrintingTimerT<std::atomic<long long>, std::functor<const std::string&>> PrintingTimer`
- Allowing `#include<chrono>` would make the code more portable but requiring c++11. Note that the actual timers should probably be long long microseconds since epoch so the chrono include isn't visible in the header.

## 15.33 impl/transform.h File Reference

General coordinate conversion based on 3 control points.

```
#include <stdint>
#include <stddef>
```

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*

### Functions

- bool [InternalZGY::generalTransform](#) (double AX0, double AY0, double AX1, double AY1, double AX2, double AY2, double BX0, double BY0, double BX1, double BY1, double BX2, double BY2, double \*X, double \*Y, std::size\_t length)

*General coordinate conversion based on 3 control points.*

#### 15.33.1 Detailed Description

General coordinate conversion based on 3 control points.

## 15.34 impl/types.h File Reference

Types and enums only used internally.

```
#include "enum.h"
#include <stdint>
```

### Classes

- struct [InternalZGY::RawDataTypeTraits< T >](#)
- struct [InternalZGY::RawDataTypeTraits< std::int8\\_t >](#)
- struct [InternalZGY::RawDataTypeTraits< std::uint8\\_t >](#)
- struct [InternalZGY::RawDataTypeTraits< std::int16\\_t >](#)
- struct [InternalZGY::RawDataTypeTraits< std::uint16\\_t >](#)
- struct [InternalZGY::RawDataTypeTraits< std::int32\\_t >](#)
- struct [InternalZGY::RawDataTypeTraits< std::uint32\\_t >](#)
- struct [InternalZGY::RawDataTypeTraits< float >](#)
- class [InternalZGY::RawDataTypeDetails](#)

### Namespaces

- [InternalZGY](#)

*Implementation not visible to clients.*



### 15.34.1 Detailed Description

Types and enums only used internally.

## 15.35 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

- [InternalZGY](#)  
*Implementation not visible to clients.*
- [OpenZGY](#)  
*The entire public API is in this namespace.*

### 15.35.1 Detailed Description

Backend specific context.

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



# Alpha version. Expect changes.

## Index

- `_accumulate`  
    `InternalZGY::GenLodImpl`, [78](#)
  - `_calculate`  
    `InternalZGY::GenLodImpl`, [78](#)
  - `_check_short_read`  
    `InternalZGY::FileCommon`, [68](#)
  - `_decimate`  
    `InternalZGY::GenLodImpl`, [78](#)
  - `_format_result`  
    `InternalZGY::GenLodBase`, [73](#)
  - `_paste1`  
    `InternalZGY::GenLodImpl`, [78](#)
  - `_paste4`  
    `InternalZGY::GenLodImpl`, [79](#)
  - `_prefix`  
    `InternalZGY::GenLodBase`, [73](#)
  - `_read`  
    `InternalZGY::GenLodBase`, [73](#)  
    `InternalZGY::GenLodC`, [75](#)
  - `_real_eof`  
    `InternalZGY::FileCommon`, [68](#)  
    `InternalZGY::LocalFileLinux`, [124](#)
  - `_report`  
    `InternalZGY::GenLodBase`, [73](#)
  - `_savestats`  
    `InternalZGY::GenLodBase`, [73](#)  
    `InternalZGY::GenLodC`, [75](#)
  - `_write`  
    `InternalZGY::GenLodBase`, [74](#)  
    `InternalZGY::GenLodC`, [76](#)
- `~IOContext`  
    `OpenZGY::IOContext`, [109](#)
- `~Register`  
    `InternalZGY::NullCompressPlugin::Register`, [161](#)
- `~ZgyWriter`  
    `OpenZGY::Impl::ZgyWriter`, [199](#)
- `add`  
    `InternalZGY::HistogramBuilder`, [86](#)  
    `InternalZGY::StatisticData`, [170](#)  
    `InternalZGY::SummaryTimer`, [174](#)
- `align`  
    `InternalZGY`, [36](#)
- `aligned`  
    `OpenZGY::SeismicStoreIOContext`, [165](#)
- `AllZero`  
    `InternalZGY`, [35](#)
- `annot_type`  
    `InternalZGY::OrderedCornerPoints`, [151](#)
- `api.cpp`, [211](#)
- `api.h`, [212](#)
- `Apply`  
    `InternalZGY::ImplicitLinearTransform2d`, [98](#)
- `array_to_hex`  
    `InternalZGY`, [36](#)
- `array_to_string`  
    `InternalZGY`, [37](#)
- `Average`  
    `InternalZGY`, [35](#)
- `AverageNon0`  
    `InternalZGY`, [35](#)
- `b`  
    `InternalZGY::ImplicitLinearTransform2d::TiePoint`, [176](#)
- `bricksiz`  
    `OpenZGY::ZgyWriterArgs`, [205](#)
- `BrickStatus`  
    `InternalZGY`, [34](#)
- `byteswapV1Long`  
    `InternalZGY`, [37](#)
- `calcLookupSize`  
    `InternalZGY::LookupTable`, [140](#)
- `calculate`  
    `InternalZGY::OffsetHeaderV2Access`, [149](#)
- `calculate_cache`  
    `InternalZGY::InfoHeaderV1Access`, [104](#)  
    `InternalZGY::InfoHeaderV2Access`, [107](#)
- `calculate_read`  
    `InternalZGY::InfoHeaderV1Access`, [104](#)  
    `InternalZGY::InfoHeaderV2Access`, [107](#)
- `calculate_write`  
    `InternalZGY::InfoHeaderV1Access`, [104](#)  
    `InternalZGY::InfoHeaderV2Access`, [107](#)
- `calculateConversionFactors`  
    `InternalZGY::HistogramData`, [89](#)
- `call`  
    `InternalZGY::GenLodImpl`, [79](#)
- `clear`  
    `InternalZGY::DataBufferNd< T, NDim >`, [54](#)  
    `InternalZGY::HistogramData`, [89](#)
- `clone`  
    `InternalZGY::DataBuffer`, [50](#)  
    `InternalZGY::DataBufferNd< T, NDim >`, [54](#)
- `close`  
    `OpenZGY::Impl::ZgyReader`, [194](#)  
    `OpenZGY::Impl::ZgyWriter`, [200](#)  
    `OpenZGY::IZgyReader`, [113](#)  
    `OpenZGY::IZgyWriter`, [120](#)

- close\_incomplete
  - OpenZGY::Impl::ZgyWriter, [200](#)
  - OpenZGY::IZgyWriter, [120](#)
- Compare
  - InternalZGY::ImplicitLinearTransform2d, [98](#)
- compress
  - InternalZGY::NullCompressPlugin, [144](#)
- compressor
  - OpenZGY::ZgyWriterArgs, [205](#)
- compressor\_t
  - InternalZGY, [33](#)
- coord\_type
  - InternalZGY::OrderedCornerPoints, [151](#)
- copyFrom
  - InternalZGY::DataBuffer, [51](#)
  - InternalZGY::DataBufferNd< T, NDim >, [54](#)
- copyTo
  - InternalZGY::GUID, [81](#)
- corners
  - OpenZGY::ZgyWriterArgs, [206](#)
- create
  - InternalZGY::FileFactory, [69](#)
- createGenericLevelOfDetail
  - InternalZGY, [37](#)
- createGenericLevelOfDetailLoPass
  - InternalZGY, [37](#)
- createLevelOfDetail
  - InternalZGY, [38](#)
- createLod
  - InternalZGY, [38](#)
- createLodMT
  - InternalZGY, [38](#)
- createLodPart
  - InternalZGY, [39](#)
- createLodST
  - InternalZGY, [40](#)
- createLodT
  - InternalZGY, [40](#)
- datarange
  - OpenZGY::ZgyWriterArgs, [206](#)
- debug\_trace
  - OpenZGY::SeismicStoreIOContext, [165](#)
- Decimate
  - InternalZGY, [35](#)
- DecimateRandom
  - InternalZGY, [35](#)
- DecimateSkipNaN
  - InternalZGY, [35](#)
- decompress
  - InternalZGY::CompressFactoryImpl, [45](#)
  - InternalZGY::NullCompressPlugin, [145](#)
- deletefile
  - OpenZGY::Impl::ZgyUtils, [198](#)
  - OpenZGY::IZgyUtils, [118](#)
- downSample1D
  - InternalZGY::LodSampling, [134](#)
- Element
  - InternalZGY::OrderedCornerPoints::Element, [58](#), [59](#)
- emptyCallback
  - InternalZGY::LoggerBase, [138](#)
- errorflag
  - OpenZGY::Impl::ZgyWriter, [200](#)
- example.h, [215](#)
- exception.h, [216](#)
- filename
  - OpenZGY::ZgyWriterArgs, [206](#)
- fill
  - InternalZGY::DataBufferNd< T, NDim >, [54](#)
- finalize
  - OpenZGY::Impl::ZgyWriter, [201](#)
  - OpenZGY::IZgyWriter, [121](#)
- GenLodBase
  - InternalZGY::GenLodBase, [72](#)
- GenLodC
  - InternalZGY::GenLodC, [75](#)
- GenLodImpl
  - InternalZGY::GenLodImpl, [77](#)
- get
  - InternalZGY::HistogramData, [90](#)
- getCallback
  - InternalZGY::Logger, [136](#)
- getCompressor
  - InternalZGY::NullCompressPlugin, [146](#)
- getcount
  - InternalZGY::HistogramData, [90](#)
- getForwarder
  - InternalZGY::Logger, [136](#)
- gethiststats
  - InternalZGY::HistogramBuilder, [86](#)
- getLinearTransform
  - InternalZGY::HistogramData, [90](#)
- getNumericEnv
  - InternalZGY::Environment, [62](#)
- getStringEnv
  - InternalZGY::Environment, [62](#)
- getValue
  - InternalZGY::Timer, [177](#)
- getValue\_s
  - InternalZGY::Timer, [178](#)
- HistogramBuilder
  - InternalZGY::HistogramBuilder, [85](#)
- HistogramData
  - InternalZGY::HistogramData, [89](#)
- hunit
  - OpenZGY::ZgyWriterArgs, [206](#)
- Identity
  - InternalZGY::ImplicitLinearTransform2d, [99](#)
- il
  - InternalZGY::OrderedCornerPoints::Element, [59](#)
- ilinc
  - OpenZGY::ZgyWriterArgs, [207](#)

- ilstart
  - OpenZGY::ZgyWriterArgs, 207
- impl/arrayops.h, 217
- impl/bulk.h, 218
- impl/compress\_null.cpp, 218
- impl/compression.cpp, 219
- impl/compression.h, 219
- impl/cornerpoints.h, 220
- impl/databuffer.h, 220
- impl/enum.h, 221
- impl/environment.h, 222
- impl/file.h, 222
- impl/file\_local.cpp, 224
- impl/file\_sd.h, 224
- impl/genlod.h, 225
- impl/guid.h, 226
- impl/histogrambuilder.h, 226
- impl/histogramdata.h, 227
- impl/iltf2d.h, 227
- impl/lodalgo.h, 228
- impl/lodsampling.h, 229
- impl/logger.h, 229
- impl/lookuptable.h, 230
- impl/meta.h, 231
- impl/roundandclip.h, 233
- impl/statisticdata.h, 233
- impl/structaccess.h, 234
- impl/subtiling.cpp, 235
- impl/subtiling.h, 236
- impl/timer.h, 237
- impl/transform.h, 238
- impl/types.h, 238
- ImplicitLinearTransform2d
  - InternalZGY::ImplicitLinearTransform2d, 97, 98
- index3\_t
  - InternalZGY, 34
- InternalZGY, 29
  - align, 36
  - AllZero, 35
  - array\_to\_hex, 36
  - array\_to\_string, 37
  - Average, 35
  - AverageNon0, 35
  - BrickStatus, 34
  - byteswapV1Long, 37
  - compressor\_t, 33
  - createGenericLevelOfDetail, 37
  - createGenericLevelOfDetailLoPass, 37
  - createLevelOfDetail, 38
  - createLod, 38
  - createLodMT, 38
  - createLodPart, 39
  - createLodST, 40
  - createLodT, 40
  - Decimate, 35
  - DecimateRandom, 35
  - DecimateSkipNaN, 35
  - index3\_t, 34
  - IsFiniteT, 40
  - IsNanT, 40
  - LodAlgorithm, 34
  - LowPass, 35
  - Maximum, 35
  - Median, 35
  - Minimum, 35
  - MinMax, 35
  - MostFrequent, 35
  - MostFrequentNon0, 35
  - ptr\_to\_array, 40
  - ptr\_to\_hex, 41
  - ptr\_to\_string, 41
  - RawCoordType, 35
  - rawdata\_t, 34
  - RawDataType, 35
  - RawGridDefinition, 35
  - RawHorizontalDimension, 35
  - RawVerticalDimension, 35
  - RoundAndClip, 41
  - subtiling, 41
  - UpdateMode, 36
  - WeightedAverage, 35
  - WhiteNoise, 35
- InternalZGY::CompressFactoryImpl, 45
  - decompress, 45
  - knownCompressors, 46
  - knownDecompressors, 46
  - registerDecompressor, 46
- InternalZGY::Config, 46
- InternalZGY::DataBuffer, 47
  - clone, 50
  - copyFrom, 51
  - makeDataBuffer3d, 51
  - scaleToFloat, 51
  - scaleToStorage, 52
- InternalZGY::DataBufferNd< T, NDim >, 52
  - clear, 54
  - clone, 54
  - copyFrom, 54
  - fill, 54
  - s\_scaleFromFloat, 55
  - s\_scaleToFloat, 55
  - scaleToFloat, 55
  - scaleToStorage, 56
  - size3d, 56
  - sizeptr, 56
  - slice, 56
  - slice1, 57
  - stride3d, 57
  - strideptr, 57
- InternalZGY::Environment, 62
  - getNumericEnv, 62
  - getStringEnv, 62
- InternalZGY::FileADT, 64
  - xx\_close, 65
  - xx\_eof, 65
  - xx\_iscloud, 65

- [xx\\_read](#), [65](#)
  - [xx\\_readv](#), [66](#)
  - [xx\\_threadsafe](#), [66](#)
  - [xx\\_write](#), [66](#)
- [InternalZGY::FileCommon](#), [67](#)
  - [\\_check\\_short\\_read](#), [68](#)
  - [\\_real\\_eof](#), [68](#)
- [InternalZGY::FileConfig](#), [68](#)
- [InternalZGY::FileFactory](#), [69](#)
  - [create](#), [69](#)
- [InternalZGY::FileHeaderAccess](#), [70](#)
- [InternalZGY::FileHeaderPOD](#), [70](#)
- [InternalZGY::FileUtilsSeismicStore](#), [71](#)
- [InternalZGY::GenLodBase](#), [71](#)
  - [\\_format\\_result](#), [73](#)
  - [\\_prefix](#), [73](#)
  - [\\_read](#), [73](#)
  - [\\_report](#), [73](#)
  - [\\_savestats](#), [73](#)
  - [\\_write](#), [74](#)
  - [GenLodBase](#), [72](#)
- [InternalZGY::GenLodC](#), [74](#)
  - [\\_read](#), [75](#)
  - [\\_savestats](#), [75](#)
  - [\\_write](#), [76](#)
  - [GenLodC](#), [75](#)
- [InternalZGY::GenLodImpl](#), [76](#)
  - [\\_accumulate](#), [78](#)
  - [\\_calculate](#), [78](#)
  - [\\_decimate](#), [78](#)
  - [\\_paste1](#), [78](#)
  - [\\_paste4](#), [79](#)
  - [call](#), [79](#)
  - [GenLodImpl](#), [77](#)
  - [suggestHistogramRange](#), [79](#)
- [InternalZGY::GUID](#), [80](#)
  - [copyTo](#), [81](#)
- [InternalZGY::HeaderAccessFactory](#), [81](#)
- [InternalZGY::HistHeaderAccess](#), [82](#)
- [InternalZGY::HistHeaderV1Access](#), [82](#)
- [InternalZGY::HistHeaderV1POD](#), [83](#)
- [InternalZGY::HistHeaderV2Access](#), [83](#)
- [InternalZGY::HistHeaderV2POD](#), [84](#)
- [InternalZGY::HistogramBuilder](#), [84](#)
  - [add](#), [86](#)
  - [gethiststats](#), [86](#)
  - [HistogramBuilder](#), [85](#)
  - [operator!=](#), [86](#)
  - [operator\\*=](#), [86](#)
  - [operator+=](#), [87](#)
  - [operator-=](#), [87](#)
  - [operator==](#), [87](#)
  - [scale](#), [87](#)
- [InternalZGY::HistogramData](#), [88](#)
  - [calculateConversionFactors](#), [89](#)
  - [clear](#), [89](#)
  - [get](#), [90](#)
  - [getcount](#), [90](#)
  - [getLinearTransform](#), [90](#)
  - [HistogramData](#), [89](#)
  - [operator!=](#), [90](#)
  - [operator\\*=](#), [90](#)
  - [operator+=](#), [90](#)
  - [operator-=](#), [91](#)
  - [operator=](#), [91](#)
  - [operator==](#), [91](#)
  - [scale](#), [91](#)
- [InternalZGY::IFileHeaderAccess](#), [92](#)
- [InternalZGY::IHeaderAccess](#), [92](#)
- [InternalZGY::IHistHeaderAccess](#), [93](#)
- [InternalZGY::IInfoHeaderAccess](#), [94](#)
- [InternalZGY::IJK](#), [95](#)
- [InternalZGY::ILookupTableAccess](#), [95](#)
- [InternalZGY::ImplicitLinearTransform2d](#), [96](#)
  - [Apply](#), [98](#)
  - [Compare](#), [98](#)
  - [Identity](#), [99](#)
  - [ImplicitLinearTransform2d](#), [97](#), [98](#)
  - [makeTiePoint](#), [99](#)
  - [operator!=](#), [99](#)
  - [operator==](#), [99](#)
  - [Push](#), [100](#)
  - [Rotate](#), [100](#)
  - [Scale](#), [100](#)
  - [this\\_type](#), [97](#)
  - [Translate](#), [101](#)
- [InternalZGY::ImplicitLinearTransform2d::TiePoint](#), [175](#)
  - [b](#), [176](#)
  - [TiePoint](#), [175](#)
- [InternalZGY::ImplicitLinearTransform2dImp](#), [101](#)
  - [Mult](#), [102](#)
- [InternalZGY::InfoHeaderAccess](#), [102](#)
- [InternalZGY::InfoHeaderV1Access](#), [103](#)
  - [calculate\\_cache](#), [104](#)
  - [calculate\\_read](#), [104](#)
  - [calculate\\_write](#), [104](#)
- [InternalZGY::InfoHeaderV1POD](#), [105](#)
- [InternalZGY::InfoHeaderV2Access](#), [105](#)
  - [calculate\\_cache](#), [107](#)
  - [calculate\\_read](#), [107](#)
  - [calculate\\_write](#), [107](#)
- [InternalZGY::InfoHeaderV2POD](#), [108](#)
- [InternalZGY::IOffsetHeaderAccess](#), [110](#)
- [InternalZGY::LocalFileLinux](#), [124](#)
  - [\\_real\\_eof](#), [124](#)
  - [xx\\_close](#), [124](#)
  - [xx\\_eof](#), [125](#)
  - [xx\\_iscloud](#), [125](#)
  - [xx\\_read](#), [125](#)
  - [xx\\_readv](#), [125](#)
  - [xx\\_threadsafe](#), [126](#)
  - [xx\\_write](#), [126](#)
- [InternalZGY::LodAllZero< T >](#), [127](#)
- [InternalZGY::LodAverage< float >](#), [128](#)
- [InternalZGY::LodAverage< T >](#), [127](#)
- [InternalZGY::LodAverageNon0< float >](#), [129](#)

- InternalZGY::LodAverageNon0< T >, [128](#)
- InternalZGY::LodDecimate< T >, [129](#)
- InternalZGY::LodDecimateSkipNaN< float >, [130](#)
- InternalZGY::LodDecimateSkipNaN< T >, [130](#)
- InternalZGY::LodMaximum< T >, [131](#)
- InternalZGY::LodMedian< float >, [132](#)
- InternalZGY::LodMedian< T >, [131](#)
- InternalZGY::LodMinimum< T >, [132](#)
- InternalZGY::LodMinMax< T >, [133](#)
- InternalZGY::LodMostFrequent< T, SkipNaN, SkipZero >, [133](#)
- InternalZGY::LodSampling, [134](#)
  - downSample1D, [134](#)
- InternalZGY::LodWeightedAverage< T >, [135](#)
- InternalZGY::Logger, [136](#)
  - getCallback, [136](#)
  - getForwarder, [136](#)
- InternalZGY::LoggerBase, [137](#)
  - emptyCallback, [138](#)
  - logger, [138](#)
  - standardCallback, [139](#)
- InternalZGY::LookupTable, [139](#)
  - calcLookupSize, [140](#)
- InternalZGY::LookupTable::LutInfo, [142](#)
- InternalZGY::LookupTableV0Access, [141](#)
- InternalZGY::LutInfoEx, [143](#)
- InternalZGY::NullCompressPlugin, [144](#)
  - compress, [144](#)
  - decompress, [145](#)
  - getCompressor, [146](#)
- InternalZGY::NullCompressPlugin::Register, [161](#)
  - ~Register, [161](#)
  - Register, [161](#)
- InternalZGY::OffsetHeaderAccess, [147](#)
- InternalZGY::OffsetHeaderV1Access, [147](#)
- InternalZGY::OffsetHeaderV1POD, [148](#)
- InternalZGY::OffsetHeaderV2Access, [148](#)
  - calculate, [149](#)
- InternalZGY::OffsetHeaderV2POD, [150](#)
- InternalZGY::OrderedCornerPoints, [150](#)
  - annot\_type, [151](#)
  - coord\_type, [151](#)
  - Max0Max1, [151](#)
  - Max0Min1, [151](#)
  - Min0Max1, [151](#)
  - operator[], [153](#)
  - OrderedCornerPoints, [152](#)
- InternalZGY::OrderedCornerPoints::Element, [58](#)
  - Element, [58, 59](#)
  - il, [59](#)
  - j, [59](#)
  - x, [59](#)
  - xl, [59](#)
  - y, [59](#)
- InternalZGY::PrintingTimer, [154](#)
- InternalZGY::PushEnvironment, [156](#)
  - PushEnvironment, [157](#)
- InternalZGY::RawDataTypeDetails, [157](#)
- InternalZGY::RawDataTypeTraits< float >, [158](#)
- InternalZGY::RawDataTypeTraits< std::int16\_t >, [158](#)
- InternalZGY::RawDataTypeTraits< std::int32\_t >, [158](#)
- InternalZGY::RawDataTypeTraits< std::int8\_t >, [159](#)
- InternalZGY::RawDataTypeTraits< std::uint16\_t >, [159](#)
- InternalZGY::RawDataTypeTraits< std::uint32\_t >, [159](#)
- InternalZGY::RawDataTypeTraits< std::uint8\_t >, [159](#)
- InternalZGY::RawDataTypeTraits< T >, [157](#)
- InternalZGY::RawPrintingTimer, [160](#)
- InternalZGY::ReadRequest, [160](#)
- InternalZGY::SimpleTimer, [168](#)
- InternalZGY::StatisticData, [169](#)
  - add, [170](#)
  - operator\*=[171](#)
  - operator+=, [171](#)
  - operator-=, [171](#)
  - scale, [171](#)
  - StatisticData, [170](#)
  - trimRange, [171](#)
- InternalZGY::SummaryPrintingTimer, [172](#)
- InternalZGY::SummaryTimer, [173](#)
  - add, [174](#)
  - reset, [174](#)
- InternalZGY::SummaryTimer::Impl, [96](#)
- InternalZGY::Timer, [176](#)
  - getValue, [177](#)
  - getValue\_s, [178](#)
  - Timer, [177](#)
- InternalZGY::TmpLookupEntry, [178](#)
- InternalZGY::ZgyInternalBulk, [183](#)
  - readConstantValue, [183](#)
  - readToExistingBuffer, [184](#)
  - readToNewBuffer, [184](#)
- InternalZGY::ZgyInternalBulk::ErrorsWillCorruptFile, [63](#)
- InternalZGY::ZgyInternalMeta, [185](#)
- InternalZGY::ZgyInternalMeta::ErrorsWillCorruptFile, [63](#)
- InternalZGY::ZgyInternalWriterArgs, [186](#)
- iocontext
  - OpenZGY::ZgyWriterArgs, [207](#)
- iocontext.h, [239](#)
- IsFiniteT
  - InternalZGY, [40](#)
- IsNaNT
  - InternalZGY, [40](#)
- j
  - InternalZGY::OrderedCornerPoints::Element, [59](#)
- knownCompressors
  - InternalZGY::CompressFactoryImpl, [46](#)
- knownDecompressors
  - InternalZGY::CompressFactoryImpl, [46](#)
- legaltag
  - OpenZGY::SeismicStoreIOContext, [165](#)
- List of exceptions, [27](#)
- LodAlgorithm
  - InternalZGY, [34](#)
- lodcompressor

- OpenZGY::ZgyWriterArgs, 207
- logger
  - InternalZGY::LoggerBase, 138
- LowPass
  - InternalZGY, 35
- makeDataBuffer3d
  - InternalZGY::DataBuffer, 51
- makeTiePoint
  - InternalZGY::ImplicitLinearTransform2d, 99
- mapDecimationTypeToLodAlgorithm
  - OpenZGY::Impl::EnumMapper, 60
- mapRawDataTypeToSampleDataType
  - OpenZGY::Impl::EnumMapper, 61
- mapRawHorizontalDimensionToUnitDimension
  - OpenZGY::Impl::EnumMapper, 61
- mapRawVerticalDimensionToUnitDimension
  - OpenZGY::Impl::EnumMapper, 61
- mapSampleDataTypeToRawDataType
  - OpenZGY::Impl::EnumMapper, 61
- mapUnitDimensionToRawHorizontalDimension
  - OpenZGY::Impl::EnumMapper, 61
- mapUnitDimensionToRawVerticalDimension
  - OpenZGY::Impl::EnumMapper, 61
- Max0Max1
  - InternalZGY::OrderedCornerPoints, 151
- Max0Min1
  - InternalZGY::OrderedCornerPoints, 151
- maxhole
  - OpenZGY::SeismicStoreIOContext, 165
- Maximum
  - InternalZGY, 35
- maxsize
  - OpenZGY::SeismicStoreIOContext, 166
- Median
  - InternalZGY, 35
- metafrom
  - OpenZGY::ZgyWriterArgs, 207
- Min0Max1
  - InternalZGY::OrderedCornerPoints, 151
- Minimum
  - InternalZGY, 35
- MinMax
  - InternalZGY, 35
- MostFrequent
  - InternalZGY, 35
- MostFrequentNon0
  - InternalZGY, 35
- Mult
  - InternalZGY::ImplicitLinearTransform2dImp, 102
- OpenZGY, 42
- OpenZGY::Errors, 43
- OpenZGY::Errors::ZgyAborted, 179
- OpenZGY::Errors::ZgyCorruptedFile, 179
- OpenZGY::Errors::ZgyEndOfFile, 180
- OpenZGY::Errors::ZgyError, 181
- OpenZGY::Errors::ZgyFormatError, 182
- OpenZGY::Errors::ZgyInternalError, 185
- OpenZGY::Errors::ZgyIoError, 188
- OpenZGY::Errors::ZgyMissingFeature, 192
- OpenZGY::Errors::ZgySegmentIsClosed, 196
- OpenZGY::Errors::ZgyUserError, 196
- OpenZGY::Formatters, 43
- OpenZGY::Impl, 44
- OpenZGY::Impl::EnumMapper, 60
  - mapDecimationTypeToLodAlgorithm, 60
  - mapRawDataTypeToSampleDataType, 61
  - mapRawHorizontalDimensionToUnitDimension, 61
  - mapRawVerticalDimensionToUnitDimension, 61
  - mapSampleDataTypeToRawDataType, 61
  - mapUnitDimensionToRawHorizontalDimension, 61
  - mapUnitDimensionToRawVerticalDimension, 61
- OpenZGY::Impl::ZgyMeta, 188
- OpenZGY::Impl::ZgyMetaAndTools, 190
  - transform, 191
  - transform1, 192
- OpenZGY::Impl::ZgyReader, 193
  - close, 194
  - read, 194, 195
  - readconst, 195
  - ZgyReader, 194
- OpenZGY::Impl::ZgyUtils, 197
  - deletefile, 198
  - ZgyUtils, 197
- OpenZGY::Impl::ZgyWriter, 198
  - ~ZgyWriter, 199
  - close, 200
  - close\_incomplete, 200
  - errorflag, 200
  - finalize, 201
  - set\_errorflag, 201
  - write, 202
  - writeconst, 202, 203
  - ZgyWriter, 199
- OpenZGY::IOContext, 109
  - ~IOContext, 109
  - toString, 109
- OpenZGY::IZgyMeta, 111
- OpenZGY::IZgyReader, 112
  - close, 113
  - read, 113, 114
  - readconst, 114
- OpenZGY::IZgyTools, 115
  - transform, 116
  - transform1, 117
- OpenZGY::IZgyUtils, 117
  - deletefile, 118
  - utils, 118
- OpenZGY::IZgyWriter, 119
  - close, 120
  - close\_incomplete, 120
  - finalize, 121
  - write, 121, 122
  - writeconst, 122, 123
- OpenZGY::ProgressWithDots, 154
  - operator(), 156



- ProgressWithDots, 155
- OpenZGY::SampleHistogram, 162
- OpenZGY::SampleStatistics, 163
- OpenZGY::SeismicStoreIOContext, 164
  - aligned, 165
  - debug\_trace, 165
  - legaltag, 165
  - maxhole, 165
  - maxsize, 166
  - sdapikey, 166
  - sdtoken, 166
  - sdtokencb, 167
  - sdurl, 167
  - segsizes, 167
  - seismicmeta, 167
  - threads, 167
  - toString, 168
  - writeid, 168
- OpenZGY::ZgyWriterArgs, 204
  - bricksize, 205
  - compressor, 205
  - corners, 206
  - datarange, 206
  - filename, 206
  - hunit, 206
  - ilinc, 207
  - ilstart, 207
  - iocontext, 207
  - lodcompressor, 207
  - metafrom, 207
  - size, 208
  - xlinc, 208
  - xlstart, 208
  - zfp\_compressor, 208
  - zfp\_lodcompressor, 209
  - zinc, 209
  - zstart, 209
  - zunit, 209
- operator!=
  - InternalZGY::HistogramBuilder, 86
  - InternalZGY::HistogramData, 90
  - InternalZGY::ImplicitLinearTransform2d, 99
- operator\*=
  - InternalZGY::HistogramBuilder, 86
  - InternalZGY::HistogramData, 90
  - InternalZGY::StatisticData, 171
- operator()
  - OpenZGY::ProgressWithDots, 156
- operator+=
  - InternalZGY::HistogramBuilder, 87
  - InternalZGY::HistogramData, 90
  - InternalZGY::StatisticData, 171
- operator-=
  - InternalZGY::HistogramBuilder, 87
  - InternalZGY::HistogramData, 91
  - InternalZGY::StatisticData, 171
- operator=
  - InternalZGY::HistogramData, 91
- operator==
  - InternalZGY::HistogramBuilder, 87
  - InternalZGY::HistogramData, 91
  - InternalZGY::ImplicitLinearTransform2d, 99
- operator[]
  - InternalZGY::OrderedCornerPoints, 153
- OrderedCornerPoints
  - InternalZGY::OrderedCornerPoints, 152
- ProgressWithDots
  - OpenZGY::ProgressWithDots, 155
- ptr\_to\_array
  - InternalZGY, 40
- ptr\_to\_hex
  - InternalZGY, 41
- ptr\_to\_string
  - InternalZGY, 41
- Push
  - InternalZGY::ImplicitLinearTransform2d, 100
- PushEnvironment
  - InternalZGY::PushEnvironment, 157
- RawCoordType
  - InternalZGY, 35
- rawdata\_t
  - InternalZGY, 34
- RawDataType
  - InternalZGY, 35
- RawGridDefinition
  - InternalZGY, 35
- RawHorizontalDimension
  - InternalZGY, 35
- RawVerticalDimension
  - InternalZGY, 35
- read
  - OpenZGY::Impl::ZgyReader, 194, 195
  - OpenZGY::IZgyReader, 113, 114
- readconst
  - OpenZGY::Impl::ZgyReader, 195
  - OpenZGY::IZgyReader, 114
- readConstantValue
  - InternalZGY::ZgyInternalBulk, 183
- readToExistingBuffer
  - InternalZGY::ZgyInternalBulk, 184
- readToNewBuffer
  - InternalZGY::ZgyInternalBulk, 184
- Register
  - InternalZGY::NullCompressPlugin::Register, 161
- registerDecompressor
  - InternalZGY::CompressFactoryImpl, 46
- reset
  - InternalZGY::SummaryTimer, 174
- Rotate
  - InternalZGY::ImplicitLinearTransform2d, 100
- RoundAndClip
  - InternalZGY, 41
- s\_scaleFromFloat
  - InternalZGY::DataBufferNd< T, NDim >, 55

- s\_scaleToFloat
  - InternalZGY::DataBufferNd< T, NDim >, 55
- Scale
  - InternalZGY::ImplicitLinearTransform2d, 100
- scale
  - InternalZGY::HistogramBuilder, 87
  - InternalZGY::HistogramData, 91
  - InternalZGY::StatisticData, 171
- scaleToFloat
  - InternalZGY::DataBuffer, 51
  - InternalZGY::DataBufferNd< T, NDim >, 55
- scaleToStorage
  - InternalZGY::DataBuffer, 52
  - InternalZGY::DataBufferNd< T, NDim >, 56
- sdapikey
  - OpenZGY::SeismicStoreIOContext, 166
- sdtoken
  - OpenZGY::SeismicStoreIOContext, 166
- sdtokencb
  - OpenZGY::SeismicStoreIOContext, 167
- sdurl
  - OpenZGY::SeismicStoreIOContext, 167
- segsizesize
  - OpenZGY::SeismicStoreIOContext, 167
- seismicmeta
  - OpenZGY::SeismicStoreIOContext, 167
- set\_errorflag
  - OpenZGY::Impl::ZgyWriter, 201
- size
  - OpenZGY::ZgyWriterArgs, 208
- size3d
  - InternalZGY::DataBufferNd< T, NDim >, 56
- sizeptr
  - InternalZGY::DataBufferNd< T, NDim >, 56
- slice
  - InternalZGY::DataBufferNd< T, NDim >, 56
- slice1
  - InternalZGY::DataBufferNd< T, NDim >, 57
- standardCallback
  - InternalZGY::LoggerBase, 139
- StatisticData
  - InternalZGY::StatisticData, 170
- stride3d
  - InternalZGY::DataBufferNd< T, NDim >, 57
- strideptr
  - InternalZGY::DataBufferNd< T, NDim >, 57
- subtiling
  - InternalZGY, 41
- suggestHistogramRange
  - InternalZGY::GenLodImpl, 79
- this\_type
  - InternalZGY::ImplicitLinearTransform2d, 97
- threads
  - OpenZGY::SeismicStoreIOContext, 167
- TiePoint
  - InternalZGY::ImplicitLinearTransform2d::TiePoint, 175
- Timer
  - InternalZGY::Timer, 177
- toString
  - OpenZGY::IOContext, 109
  - OpenZGY::SeismicStoreIOContext, 168
- transform
  - OpenZGY::Impl::ZgyMetaAndTools, 191
  - OpenZGY::IZgyTools, 116
- transform1
  - OpenZGY::Impl::ZgyMetaAndTools, 192
  - OpenZGY::IZgyTools, 117
- Translate
  - InternalZGY::ImplicitLinearTransform2d, 101
- trimRange
  - InternalZGY::StatisticData, 171
- UpdateMode
  - InternalZGY, 36
- utils
  - OpenZGY::IZgyUtils, 118
- WeightedAverage
  - InternalZGY, 35
- WhiteNoise
  - InternalZGY, 35
- write
  - OpenZGY::Impl::ZgyWriter, 202
  - OpenZGY::IZgyWriter, 121, 122
- writeconst
  - OpenZGY::Impl::ZgyWriter, 202, 203
  - OpenZGY::IZgyWriter, 122, 123
- writeid
  - OpenZGY::SeismicStoreIOContext, 168
- x
  - InternalZGY::OrderedCornerPoints::Element, 59
- xl
  - InternalZGY::OrderedCornerPoints::Element, 59
- xlinc
  - OpenZGY::ZgyWriterArgs, 208
- xlstart
  - OpenZGY::ZgyWriterArgs, 208
- xx\_close
  - InternalZGY::FileADT, 65
  - InternalZGY::LocalFileLinux, 124
- xx\_eof
  - InternalZGY::FileADT, 65
  - InternalZGY::LocalFileLinux, 125
- xx\_iscloud
  - InternalZGY::FileADT, 65
  - InternalZGY::LocalFileLinux, 125
- xx\_read
  - InternalZGY::FileADT, 65
  - InternalZGY::LocalFileLinux, 125
- xx\_readv
  - InternalZGY::FileADT, 66
  - InternalZGY::LocalFileLinux, 125
- xx\_threadsafe
  - InternalZGY::FileADT, 66
  - InternalZGY::LocalFileLinux, 126

# Alpha version. Expect changes.

xx\_write

InternalZGY::FileADT, [66](#)

InternalZGY::LocalFileLinux, [126](#)

y

InternalZGY::OrderedCornerPoints::Element, [59](#)

zfp\_compressor

OpenZGY::ZgyWriterArgs, [208](#)

zfp\_lodcompressor

OpenZGY::ZgyWriterArgs, [209](#)

ZgyReader

OpenZGY::Impl::ZgyReader, [194](#)

ZgyUtils

OpenZGY::Impl::ZgyUtils, [197](#)

ZgyWriter

OpenZGY::Impl::ZgyWriter, [199](#)

zinc

OpenZGY::ZgyWriterArgs, [209](#)

zstart

OpenZGY::ZgyWriterArgs, [209](#)

zunit

OpenZGY::ZgyWriterArgs, [209](#)