OpenEXR File Layout

Last Update: Aug 26, 2018

Florian Kainz

Industrial Light & Magic

Document Purpose and Audience

This document gives an overview of the layout of OpenEXR 2.0 image files as byte sequences. It covers both single and multi-part formats, and how deep data is handled.

The text assumes that the reader is familiar with OpenEXR terms such as "channel", "attribute", "data window" or "chunk". For an explanation of those terms see the *Technical Introduction to OpenEXR*.

Note: This document does not define the OpenEXR file format. OpenEXR is defined as the file format that is read and written by the IlmImf open-source C++ library. If this document and the IlmImf library disagree, then the library takes precedence.

Table of Contents

Backwards Compatibility and New or Changed Functionality

OpenEXR Backwards Compatibility (1.7 and 2.0)

OpenEXR 1.7 and earlier format files are fully supported by OpenEXR 2.0. You can still use the 1.7 file format with the 2.0 library. If you use the 2.0 format for single-part scan line image and tile image data, your data will be stored in the same way as the 1.7 files. You can recompile your 1.7 files to take advantage of the new format EXRs (multiple-part files, and/or deep scan line and deep tile data).

New Features for OpenEXR 2.0: Multi-Part and Deep Data

The multi-part format is an extension of the OpenEXR 1.7 single-part file format. In addition to supporting the OpenEXR 1.7 data storage (a single scan line or tiled image), OpenEXR 2.0 files can be used to store multiple views and/or deep data (deep scan line or deep tiles).

While you can continue to use the 1.7 format for files, these changes to the file layout are required to support the new multi-part and deep data features:

Feature	Description	See
Version field	Bits 11 and 12 indicate whether the file contains deep data (bit 11), or more than one part (bit 12).	*Deep Data* on page 6
Header	To store more than one part in the file, you need to have a header for each part.	*Structure* on page 7
Header attributes	There are a number of attributes which have been defined to store data which is relevant to deep data and multi-part files. These include: name (one for each part), data type (you can have different types of data in different views), and the maximum number of samples to take in a deep data channel.	*Multi-Part and Deep Data Header Attributes* on page 9
Offset tables and chunks	To store more than one part in the file, you need to have an offset table for each part, and chunks for each part. The chunks must begin with a part number.	*Component Four: Offset Tables* on page 10, and *Chunk Layout* on page 11.
Deep Data	Deep data has a unique storage format.	*Deep Data* on page 13

Basic Data Types

An OpenEXR file is a sequence of 8-bit bytes. Groups of bytes represent basic objects such as integral numbers, floating-point numbers and text. Those objects are grouped together to form compound objects such as attributes or scan lines.

Integers

Binary integral numbers with 8, 16, 32 or 64 bits are stored as 1, 2, 4 or 8 bytes. Integral numbers can be signed or unsigned. Signed numbers are represented using two's complement. Integral numbers are little-endian (that is, the least significant byte is closest to the start of the file).

OpenEXR uses the following six integer data types:

name	signed	size in bytes
`unsigned char`	no	1
`short`	yes	2
`unsigned short`	no	2
`int`	yes	4
`unsigned int`	no	4
`unsigned long`	no	8

Floating-Point Numbers

Binary floating-point numbers with 16, 32 or 64 bits are stored as 2, 4 or 8 bytes. The representation of 32-bit and 64-bit floating-point numbers conforms to the IEEE 754 standard. The representation of 16-bit floating-point numbers is analogous to IEEE 754, but with 5 exponent bits and 10 bits for the fraction. The exponent bias is 15. Floating-point numbers are little-endian (that is: the least significant bits of the fraction are in the byte closest to the beginning of the file, while the sign bit and the most significant bits of the exponent are in the byte closest to the end of the file).

The following table lists the names and sizes of OpenEXR's floating-point data types:

name	size in bytes
`half`	2
`float`	4
`double`	8

Text

Text strings are represented as sequences of 1-byte characters of type `char`. Depending on the context, either the end of a string is indicated by a null character (0x00), or the length of the string is indicated by an `int` that precedes the string.

Packing

Data in an OpenEXR file are densely packed; the file contains no "padding". For example, consider the following C struct:

```
struct SI
{
    short s;
    int i;
};
```

On most computers, the in-memory representation of an `SI` object occupies 8 bytes: 2 bytes for `s`, 2 padding bytes to ensure four-byte alignment of `i`, and 4 bytes for `i`. In an OpenEXR file the same object would consume only 6 bytes: 2 bytes for `s` and 4 bytes for `i`. The 2 padding bytes are not stored in the file.

File Layout

High-Level Layout

Depending on whether the pixels in an OpenEXR file are stored as scan lines or as tiles, the file consists of the following components:

component	single-part file with...		multi-part file:
	scan lines:	**tiles:**	
one	`magic number`	`magic number`	`magic number`
two	`version field`	`version field`	`version field`

three	`header`	`header`	`part 0 header`
			`[part 1 header]`
			`...`
			`[<empty header>]`

four	`line offset table`	`tile offset table`	`part 0 chunk offset table`
			`[part 1 chunk offset table]`
			`...`

five	`scan line blocks`	`tiles`	`chunks`

It is the version field part which indicates whether the file is single or multi-part and whether the file contains deep data. “Chunk” is a general term to describe blocks of pixel data. A chunk can be a scan line block, a tile or deep data (scan line or tile).

Deep data has no unique component structure of its own, but uses the structure that the file would have if it did not have deep data in it.

Comparison between Single-Part and Multi-Part File Layouts

Multi-part files have the same high level structure as single-part OpenEXR files, except the header, offset table and chunk components can have any number (two or more) parts. There must be the same number of headers as offset tables, and they must be in the same order. In addition, the header component of a multi-part file must end with a null byte (0x00). In multi-part files, each chunk contains a field that indicates which part's data it contains.

Components One and Two: Magic Number and Version Field

Magic Number

The magic number, of type `int`, is always 20000630 (decimal). It allows file readers to distinguish OpenEXR files from other files, since the first four bytes of an OpenEXR file are always 0x76, 0x2f, 0x31 and 0x01.

Version Field

The version field, of type `int`, is the four-byte group following the magic number, and it is treated as two separate bit fields.

<table>
<tr><th>Byte/bit position</th><th colspan="3">Description and notes</th></tr>
<tr><td>first byte (bits 0 through 7)</td><td colspan="3">The 8 least significant bits, they contain the file format version number.
The current OpenEXR version number is version 2.</td></tr>
<tr><td>second, third and fourth bytes (bits 8 through 31)</td><td colspan="3">The 24 most significant bits, these are treated as a set of boolean flags.</td></tr>
<tr><td></td><td>Bit 9 (the single tile bit)
bit mask: 0x200</td><td>Indicates that this is a single-part file which is in tiled format.</td><td>If bit 9 is 1:
this is a regular single-part image and the pixels are stored as tiles, and
bits 11 and 12 must be 0.
If bit 9 is 0, and bits 11 and 12 are also 0: the data is stored as regular single-part scan line file.
This bit is for backwards compatibility with older libraries: it is only set when there is one "normal" tiled image in the file.</td></tr>
<tr><td></td><td>Bit 10 (the long name bit)
bit mask: 0x400</td><td>Indicates whether the file contains “long names”.</td><td>If bit 10 is 1, the maximum length is 255 bytes.
If bit 10 is 0, the maximum length of attribute names, attribute type names and channel names is 31 bytes.</td></tr>
<tr><td></td><td>Bit 11 (the non-image bit)
bit mask: 0x800</td><td>Indicates whether the file contains any “non-image parts” (deep data).</td><td>If bit 11 is 1, there is at least one part which is not a regular scan line image or regular tiled image (that is, it is a deep format).
If bit 11 is 0, all parts are entirely single or multiple scan line or tiled images.
New in 2.0.</td></tr>
<tr><td></td><td>Bit 12 (the multipart bit)
bit mask: 0x1000</td><td>Indicates the file is a multi-part file.</td><td>If bit 12 is 1:
the file does not contain exactly 1 part and the 'end of header' byte must be included at the end of each header part, and
the part number fields must be added to the chunks.
If bit 12 is 0, this is not a multi-part file and the 'end of header' byte and part number fields in chunks must be omitted.
New in 2.0.</td></tr>
<tr><td colspan="4">The remaining 19 flags in the version field are currently unused and should be set to 0.</td></tr>
</table>

Version field, valid values

All valid combinations of the version field bits are as follows:

Description	Compatible with	bit 9	bit 10	bit 11
Single-part scan line. One normal scan line image.	All versions of OpenEXR.	0	0	0
Single-part tile. One normal tiled image.	All versions of OpenEXR.	1	0	0
Multi-part (new in 2.0). Multiple normal images (scan line and/or tiled).	OpenEXR 2.0.	0	0	1
Single-part deep data (new in 2.0). One deep tile or deep scan line part.	OpenEXR 2.0.	0	1	0
Multi-part deep data (new in 2.0). Multiple parts (any combination of: tiles, scan lines, deep tiles and/or deep scan lines).	OpenEXR 2.0.	0	1	1

Note: The version field bits define what capabilities must be available in the software so it can handle the file, rather than the exact format of the file. While the 9 and 11 bit settings must agree with the type attributes of all parts, in OpenEXR 2.0 the data format of each type is definitively set by the type attribute in that part's header alone.

Component Three: Header

Structure

Single-part file

The header component of the single-part file holds a single header (for single-part files).

Each header is a sequence of attributes ended by a null byte.

The file has the same structure as a 1.7 file. That is, the multi-part bit (bit 12) must be 0, and the single null byte that signals the end of the headers must be omitted. This structure also applies to single-part deep data files.

Multi-part file (new in 2.0)

The header component of a multi-part file holds a set of headers, with a separate header for each part (in multi-part files) and a null byte signalling the end of the header component:

```
part 0 header
[part 1 header]
...
[<empty header>]
```

Each header is a sequence of attributes ended by a null byte.

The multipart bit (bit 12) must be set to 1, and the list of headers must be followed by a single null byte (0x00) (that is, an empty header).

Attribute Layout

The layout of an attribute is as follows:

attribute name

attribute type

attribute size

attribute value

The **attribute name** and the **attribute type** are null-terminated text strings. Excluding the null byte, the name and type must each be as least 1 byte and at most :

- 31 bytes long (if bit 10 is set to 0), or
- 255 bytes long (if bit 10 is set to 1).

Both single-part and multi-part files use the same attribute types.

The **attribute size**, of type `int`, indicates the size (in bytes) of the attribute value.

The layout of the **attribute value** depends on the attribute type. The IlmImf library predefines several different attribute types (see page 15). Application programs can define and store additional attribute types.

Header Attributes (All Files)

The header of every OpenEXR file must contain at least the following attributes:

attribute name	attribute type
`channels`	`chlist`
`compression`	`compression`
`dataWindow`	`box2i`
`displayWindow`	`box2i`
`lineOrder`	`lineOrder`
`pixelAspectRatio`	`float`
`screenWindowCenter`	`v2f`
`screenWindowWidth`	`float`

For descriptions of what these attributes are for, see the *Technical Introduction to OpenEXR*.

Tile Header Attribute

This attributes is required in the header for all files which contain one or more tiles:

attribute name	attribute type	Notes
`tiles`	`tiledesc`	Determines the size of the tiles and the number of resolution levels in the file. **Note:** The IlmImf library ignores tile description attributes in scan line based files. The decision whether the file contains scan lines or tiles is based on the value of bit 9 in the file's version field, not on the presence of a tile description attribute.

Multi-View Header Attribute

This attribute can be used in the header for multi-part files:

attribute name	attribute type	Notes
`view`	`text`	

Multi-Part and Deep Data Header Attributes (New in 2.0)

These attributes are required in the header for all multi-part and/or deep data OpenEXR files.

attribute name	attribute type	Notes
`name`	`string`	Required if either the multipart bit (12) or the non-image bit (11) is set.
`type`	`string`	Required if either the multipart bit (12) or the non-image bit (11) is set. Set to one of: • scanlineimage • tiledimage • deepscanline, or • deeptile. **Note:** This value must agree with the version field's tile bit (9) and non-image (deep data) bit (11) settings.
`version`	`int`	This document describes version 1 data for all part types. version is required for deep data (deepscanline and deeptile) parts. If not specified for other parts, assume version=1.
`chunkCount`	`int`	Required if either the multipart bit (12) or the non-image bit (11) is set.
`tiles`	`tileDesc`	Required for parts of type `tiledimage` and `deeptile`.

For more information about the standard OpenEXR attributes and optional attributes such as `preview images`, see the *OpenEXR File Layout* document.

Deep Data Header Attributes (New in 2.0)

These attributes are required in the header for all files which contain deep data (deepscanline or deeptile):

attribute name	attribute type	Notes
`tiles`	`tileDesc`	Required for parts of type tiledimage and deeptile.

attribute name	attribute type	Notes
`maxSamplesPerPixel`	`int`	Required for deep data (deepscanline and deeptile) parts. **Note:** Since the value of **maxSamplesPerPixel** maybe be unknown at the time of opening the file, the value "`-1`" is written to the file to indicate an unknown value. When the file is closed, this will be overwritten with the correct value. If file writing does not complete correctly due to an error, the value `-1` will remain. In this case, the value must be derived by decoding each chunk in the part.
`version`	`int`	Should be set to `1`. It will be changed if the format is updated.
`type`	`string`	Must be set to `deepscanline` or `deeptile`.

For information about channel layout and a list of reserved channel names, see the *Technical Introduction to OpenEXR* document, C*hannel Names* section.

Component Four: Offset Tables

Offset Tables

An offset table allows random access to pixel data chunks. An offset table is a sequence of offsets, with one offset per chunk. Each offset (of type `unsigned long`) indicates the distance, in bytes, between the start of the file and the start of the chunk.

Chunks can be of any of the four data types.

Offset Table Size

The number of entries in an offset table is defined in one of two ways:

1. If the multipart (12) bit is unset and the chunkCount is not present, the number of entries in the chunk table is computed using the dataWindow and tileDesc attributes and the compression format.
2. If the multipart (12) bit is set, the header must contain a chunkCount attribute (which indicates the size of the table and the number of chunks).

Scan Lines

For scan line blocks, the line offset table is a sequence of scan line offsets, with one offset per scan line block. In the table, scan line offsets are ordered according to increasing scan line y coordinates.

Tiles

For tiles, the offset table is a sequence of tile offsets, one offset per tile. In the table, scan line offsets are sorted the same way as tiles in `INCREASING_Y` order.

Multi-part (New in 2.0)

For multi-part files, each part defined in the header component has a corresponding chunk offset table.

Component Five: Pixel data

Chunk Layout (New in 2.0)

A “chunk” is a general term for a pixel data block. The scan line and tile images have the same format that they did in OpenEXR 1.7. OpenEXR 2.0 introduces two new types (deep scan line and deep tile).

The layout of each chunk is as follows:

> [part number] (if multi-part bit is set)
>
> chunk data

The **part number** (of type `unsigned long`) is only present in multi-part files. It indicates which part this chunk belongs to. 0 indicates the chunk belongs to the part defined by the first header and the first chunk offset table. The part number is omitted if the multi-part bit (12) is not set (this saves space and enforces backwards compatibility to software which does not support multi-part files).

The **chunk data** is dependent on the type attribute - but (other than the part number) has the same structure as a single-part file of the same format:

part type	type attribute	Notes
scan line	indicated by a type attribute of “scanlineimage”	Each chunk stores a scan line block, with the minimum y coordinate of the scan line(s) within the chunk. See *Regular scan line image block layout*, on page 12.
tiled	indicated by a type attribute of “tiledimage”	See *Regular image tile layout*, on page 13.
deep scan line	indicated by a type attribute of “deepscanline”	See *Deep scan line layout*, on page 13.
deep tile	indicated by a type attribute of “deeptile”	See *Deep tiled layout*, on page 13.

For more information about data types, see page Error: no se encontró el origen de la referencia.

Regular Scan Line Blocks

For scan line images and deep scan line images, one or more scan lines may be stored together as a scan line block. The number of scan lines per block depends on how the pixel data are compressed:

compression method	number of scan lines per block
NO_COMPRESSION	1
RLE_COMPRESSION	1
ZIPS_COMPRESSION	1
ZIP_COMPRESSION	16
PIZ_COMPRESSION	32
PXR24_COMPRESSION	16
B44_COMPRESSION	32
B44A_COMPRESSION	32

Each scan line block has a y coordinate of type `int`. The block's y coordinate is equal to the pixel space y coordinate of the top scan line in the block. The top scan line block in the image is aligned with the top edge of the data window (that is, the y coordinate of the top scan line block is equal to the data window's minimum y).

If the height of the image's data window is not a multiple of the number of scan lines per block, then the block that contains the bottom scan line contains fewer scan lines than the other blocks.

Regular scan line image block layout

The layout of a regular image scan line block is as follows:

[part number] (if multipart bit is set)

y coordinate

pixel data size

pixel data

The **pixel data size**, of type `int`, indicates the number of bytes occupied by the actual pixel data.

Within the **pixel data**, scan lines are stored top to bottom. Each scan line is contiguous, and within a scan line the data for each channel are contiguous. Channels are stored in alphabetical order, according to channel names. Within a channel, pixels are stored left to right.

Compressed data

If the file's compression method is `NO_COMPRESSION`, then the original, uncompressed pixel data are stored directly in the file. Otherwise, the uncompressed pixels are fed to the appropriate compressor, and either the compressed or the uncompressed data are stored in the file, whichever is smaller.

The layout of the compressed data depends on which compression method was applied. The compressed formats are not described here. For information on the compressed data formats, see the source code for the IlmImf library.

Regular ImageTiles

Regular image tile layout

The layout of a regular image tile is as follows:

> [part number] (if multi-part bit is set)
>
> tile coordinates
>
> pixel data size
>
> pixel data

The **tile coordinates**, a sequence of four `int`s (tileX, tileY, levelX, levelY) indicates the tile's position and resolution level. The **pixel data size**, of type `int`, indicates the number of bytes occupied by the pixel data.

The **pixel data** in a tile are laid out in the same way as in a scan line block, but the length of the scan lines is equal to the width of the tile, and the number of scan lines is equal to the height of the tile.

If the width of a resolution level is not a multiple of the file's tile width, then the tiles at the right edge of that resolution level have shorter scan lines. Similarly, if the height of a resolution level is not a multiple of the file's tile height, then tiles at the bottom edge of the resolution level have fewer scan lines.

Deep Data (New in 2.0)

Deep images store an arbitrarily long list of data at each pixel location (each pixel contains a list of samples, and each sample contains a fixed number of channels).

Deep scan line layout

Deep scan line images are indicated by a type attribute of “deepscanline”. Each chunk of deep scan line data is a single scan line of data. The data in each chunk is laid out as follows:

> [part number] (if multipart bit is set)
>
> y coordinate
>
> packed size of pixel offset table
>
> packed size of sample data
>
> unpacked size of sample data
>
> compressed pixel offset table
>
> compressed sample data

The **unpacked size of the sample data** (an `unsigned long`) is the size of the deep sample data once it is unpacked. It is necessary to specify the unpacked size since the data may be arbitrarily large (so generally cannot otherwise be determined without decompressing the data first).

Deep tiled layout

Tiled images are indicated by a type attribute of “deeptile”. Each chunk of deep tile data is a single tile. The data in each chunk is laid out as follows:

> [part number] (if multipart bit is set)
>
> tile coordinates
>
> packed size of pixel offset table

packed size of sample data

unpacked size of sample data

compressed pixel offset table

compressed sample data

The **unpacked size of the sample data** (an `unsigned long`) is the size of the deep data once it is unpacked. It is necessary to specify the unpacked size since the data may be arbitrarily large (so generally cannot otherwise be determined without decompressing the data first).

The **pixel offset table** is a list of `int`s, one for each column within the dataWindow. Each entry *n* in the table indicates the total number of samples required to store the pixel in *n* as well as all pixels to the left of it. Thus, the first samples stored in each channel of the pixel data are for the pixel in column 0, which contains table[1] samples. Each channel contains table[width-1] samples in total.

Unpacked deep data chunks

When decompressed, the unpacked chunk consists of the channel data stored in a non-interleaved fashion:

```
pixel sample data for channel 0
pixel sample data for channel 1
pixel sample data for channel ...
pixel sample data for channel n
```

Exception: For ZIP_COMPRESSION only there will be up to 16 scanlines in the packed sample data block:

```
pixel sample data for channel 0 for scanline 0
pixel sample data for channel 1 for scanline 0
pixel sample data for channel ... for scanline 0
pixel sample data for channel n for scanline 0

pixel sample data for channel 0 for scanline 1
pixel sample data for channel 1 for scanline 1
pixel sample data for channel ... for scanline 1
pixel sample data for channel n for scanline 1

...
```

Deep data compression

The following compression schemes are the only ones permitted for deep data:

compression method	**number of scan lines per block**
`NO_COMPRESSION`	1
`RLE_COMPRESSION`	1
`ZIPS_COMPRESSION`	1
`ZIP_COMPRESSION`	16

Predefined Attribute Types

The IlmImf library predefines the following attribute types:

type name	data
box2i	Four `int`s: xMin, yMin, xMax, yMax
box2f	Four `float`s: xMin, yMin, xMax, yMax
chlist	A sequence of channels followed by a null byte (0x00). Channel layout: name — zero-terminated string, from 1 to 255 bytes long pixel type — `int`, possible values are: `UINT` = 0 `HALF` = 1 `FLOAT` = 2 pLinear — `unsigned char`, possible values are 0 and 1 reserved — three `char`s, should be zero xSampling — `int` ySampling — `int`
chromaticities	Eight `float`s: redX, redY, greenX, greenY, blueX, blueY, whiteX, whiteY
compression	`unsigned char`, possible values are `NO_COMPRESSION` = 0 `RLE_COMPRESSION` = 1 `ZIPS_COMPRESSION` = 2 `ZIP_COMPRESSION` = 3 `PIZ_COMPRESSION` = 4 `PXR24_COMPRESSION` = 5 `B44_COMPRESSION` = 6 `B44A_COMPRESSION` = 7
double	`double`
envmap	`unsigned char`, possible values are: `ENVMAP_LATLONG` = 0 `ENVMAP_CUBE` = 1
float	`float`
int	`int`
keycode	Seven `int`s: filmMfcCode, filmType, prefix, count, perfOffset, perfsPerFrame, perfsPerCount

type name	data
lineOrder	`unsigned char`, possible values are: `INCREASING_Y` = 0 `DECREASING_Y` = 1 `RANDOM_Y` = 2
m33f	9 `float`s
m44f	16 `float`s
preview	Two `unsigned int`s, width and height, followed by 4×width×height `unsigned char`s of pixel data. Scan lines are stored top to bottom; within a scan line pixels are stored from left to right. A pixel consists of four `unsigned char`s, R, G, B, A.
rational	An `int`, followed by an `unsigned int`.
string	String length, of type `int`, followed by a sequence of `char`s.
stringvector	A sequence of zero or more text strings. Each string is represented as a string length, of type `int`, followed by a sequence of `char`s. The number of strings can be inferred from the total attribute size (see the *Attribute Layout* section, on page 8).
tiledesc	Two `unsigned int`s: xSize, ySize, followed by mode, of type `unsigned char`, where mode = levelMode + roundingMode×16 Possible values for levelMode: `ONE_LEVEL` = 0 `MIPMAP_LEVELS` = 1 `RIPMAP_LEVELS` = 2 Possible values for roundingMode: `ROUND_DOWN` = 0 `ROUND_UP` = 1
timecode	Two `unsigned int`s: timeAndFlags, userData.
v2i	Two `int`s
v2f	Two `float`s
v3i	Three `int`s.
v3f	Three `float`s.

Sample File

The following is an annotated byte-by-byte listing of a complete OpenEXR file. The file contains a scan-line based image with four by three pixels. The image has two channels: G, of type `HALF`, and Z, of type `FLOAT`. The pixel data are not compressed. The entire file is 415 bytes long.

The first line of text in each of the gray boxes below lists up to 16 bytes of the file in hexadecimal notation. The second line in each box shows how the bytes are grouped into integers, floating-point numbers and text strings. The third and fourth lines indicate how those basic objects form compound objects such as attributes or the line offset table.

```
 76   2f   31   01   02   00   00   00   63   68   61   6e   6e   65   6c   73
      20000630     |         2         |  c    h    a    n    n    e    l    s
   magic number    |  version, flags   | attribute name
                   |                   | start of header
```

```
 00   63   68   6c   69   73   74   00   25   00   00   00   47   00   01   00
 \0 |  c    h    l    i    s    t   \0 |        37         |  G   \0 | HALF
    | attribute type                   | attribute size    | attribute value
```

```
 00   00   00   00   00   00   01   00   00   00   01   00   00   00   5a   00
         |  0 |       0       |         1         |         1         |  Z   \0 |
```

```
 02   00   00   00   00   00   00   00   01   00   00   00   01   00   00   00
 FLOAT              |  0 |       0       |         1         |         1         |
                                                                                 |
```

```
 00   63   6f   6d   70   72   65   73   73   69   6f   6e   00   63   6f   6d
 \0 |  c    o    m    p    r    e    s    s    i    o    n   \0 |  c    o    m
    | attribute name                                          | attribute type
```

```
 70   72   65   73   73   69   6f   6e   00   01   00   00   00   00   64   61
  p    r    e    s    s    i    o    n   \0 |         1         | NONE| d    a
                                           | attribute size    |value|
```

```
 74   61   57   69   6e   64   6f   77   00   62   6f   78   32   69   00   10
  t    a    W    i    n    d    o    w   \0 |  b    o    x    2    i   \0 |
 attribute name                             | attribute type              |

 00   00   00   00   00   00   00   00   00   00   00   03   00   00   00   02
    16        |         0         |         0         |         3         |
attribute size| attribute value

 00   00   00   64   69   73   70   6c   61   79   57   69   6e   64   6f   77
    2         |  d    i    s    p    l    a    y    W    i    n    d    o    w
              | attribute name

 00   62   6f   78   32   69   00   10   00   00   00   00   00   00   00   00
\0 |  b    o    x    2    i   \0 |        16         |         0         |
   | attribute type              | attribute size    | attribute value

 00   00   00   03   00   00   00   02   00   00   00   6c   69   6e   65   4f
    0         |         3         |         2         |  l    i    n    e    O
                                                     | attribute name

 72   64   65   72   00   6c   69   6e   65   4f   72   64   65   72   00   01
  r    d    e    r   \0 |  l    i    n    e    O    r    d    e    r   \0 |
                        | attribute type                                  |

 00   00   00   00   70   69   78   65   6c   41   73   70   65   63   74   52
    1         |INCY | p    i    x    e    l    A    s    p    e    c    t    R
attribute size|value| attribute name

 61   74   69   6f   00   66   6c   6f   61   74   00   04   00   00   00   00
  a    t    i    o   \0 |  f    l    o    a    t   \0 |         4         |
                        | attribute type              | attribute size    |
```

```
 00   80   3f   73   63   72   65   65   6e   57   69   6e   64   6f   77   43
   1.0         | s    c    r    e    e    n    W    i    n    d    o    w    C
attribute value| attribute name
```

```
 65   6e   74   65   72   00   76   32   66   00   08   00   00   00   00   00
  e    n    t    e    r   \0 | v    2    f   \0 |         8         |
                             | attribute type    | attribute size    |
```

```
 00   00   00   00   00   00   73   63   72   65   65   6e   57   69   6e   64
0.0      |         0.0       | s    c    r    e    e    n    W    i    n    d
attribute value              | attribute name
```

```
 6f   77   57   69   64   74   68   00   66   6c   6f   61   74   00   04   00
  o    w    W    i    d    t    h   \0 | f    l    o    a    t   \0 |
                                       | attribute type              |
```

```
 00   00   00   00   80   3f   00   3f   01   00   00   00   00   00   00   5f
4        |        1.0       | \0  |                 319                  |
size     | attribute value  |     |        offset of scan line 0         |
                    end of header | start of scan line offset table
```

```
 01   00   00   00   00   00   00   7f   01   00   00   00   00   00   00   00
            351                  |                383                  |
   offset of scan line 1          |       offset of scan line 2          |
                                    end of scan line offset table |
```

```
 00   00   00   18   00   00   00   00   00   54   29   d5   35   e8   2d   5c
    0         |         24        |  0.000  |  0.042  |  0.365  |  0.092  |
    y         |  pixel data size  |  pixel data for G channel             |
  scan line 0
```

```
 28   81   3a   cf   e1   34   3e   8b   0b   bb   3d   89   74   f9   3e   01
0.000985395   |   0.176643        |   0.0913306       |   0.487217        |
pixel data for Z channel                                                  |
                                                                          |
```

```
 00   00   00   18   00   00   00   37   38   76   33   74   3b   73   38   7f
    1         |         24        |  0.527  |  0.233  |  0.932  |  0.556  |
    y         |  pixel data size  |  pixel data for G channel             |
  scan line 1
```

```
 ab   e8   3e   8a   cf   54   3f   5b   6c   11   3f   20   35   50   3d   02
0.454433      |   0.831292        |   0.56806         |   0.0508319       |
pixel data for Z channel                                                  |
                                                                          |
```

```
 00   00   00   18   00   00   00   23   3a   0a   34   02   3b   5d   3b   38
    2         |         24        |  0.767  |  0.252  |  0.876  |  0.920  |
    y         |  pixel data size  |  pixel data for G channel             |
  scan line 2
```

```
 f3   9a   3c   4d   ad   98   3e   1c   14   08   3f   4c   f3   03   3f
0.0189148     |   0.298197        |   0.531557        |   0.515431
pixel data for Z channel
                                                          end of file
```