* Extended Tags and Special Elements

# Chapter Overview

This chapter provides detailed information regarding HDF-supported HDF extended tags and the special elements they define. General information about tags and detailed specifications of basic tags are presented in Chapter 9, “Tag Specifications."

## Extended Tags and Alternate Physical Storage Methods

Prior to HDF Version 3.2, each data element had to be stored in one contiguous block in the basic HDF file. Version 3.2 introduced *extended tags*, a mechanism supporting alternate physical data element storage structures. All HDF-supported tags with variable-sized data elements can take advantage of the extended tag features.

### Extended Tag Implementation

Extended tags are automatically recognized by current versions of the HDF library and interpreted according to a description record. The description record, a complete data element, identifies the type of extended element and provides the relevant parameters for data retrieval.

Extended tags currently support four styles of alternate physical storage:

* Linked block elements are stored in several non-contiguous blocks within the basic HDF file.
* External elements are stored in a separate file, external to the basic HDF file.
* Chunked elements are stored in blocks within the basic HDF file to facilitate selective I/O.
* Compressed elements are stored in a configurable compressed mode within the basic HDF file to save storage space and to speed I/O and data transfer.

Every HDF-supported tag is represented in HDF libraries and files by a tag number. HDF-supported tags that take advantage of alternative physical storage features have an alternative tag number, called an *extended tag number*, that appears instead of the original tag number when an alternative physical storage method is in use.

When The HDF Group determines that an extended tag should be defined for a given tag, the extended tag number is determined by performing an arithmetic OR with the original tag number and the hexadecimal number 0x4000. Since all basic tags are numbered 0x0001 through 0x3FFF, this arithmetic OR effectively adds 0x4000, or a decimal value of 16384, to derive the extended tag value.

For example, the tag DFTAG\_RI points to a data element containing a raster image. If the data element is stored contiguously in the same HDF file, the DD contains the tag number 302; if the data element is stored either in linked blocks or in an external file, the DD contains the extended tag number 16686.

If a data object uses a regular tag number, its storage structure will be exactly as described in the "Section 9.3, "Tag Specifications." Figure 10a illustrates this general structure with the DD pointing directly to a single, contiguous data block.

* Regular Data Object

*regular\_tag* Tag number



*ref\_no* Reference number

*data\_element* The data element

If a data object uses an extended tag, the storage structure will appear generally as illustrated in Figure 10b. The DD will point to an extended tag description record which in turn will point to the data.

* Data Object with Extended Tag



*extended\_tag* Extended tag number

*ref\_no* Reference number

*ext\_tag\_desc* A 32-bit constant defined in Hdfi.h that identifies the type of alternative storage involved. Current definitions include EXT\_LINKED for linked block elements or EXT\_EXTERN for external elements.

*data\_location\_information*Information identifying and describing the linked blocks or external file

*data* The data, stored either in linked blocks or in an external file

Since the HDF tools were modified for HDF Version 3.2 to handle extended tags automatically, the only thing the user ever has to do is specify the use of either the linked blocks mechanism or an external file. Once that has been specified, the user can forget about extended tags entirely; the HDF library will manage everything correctly.

There is only one circumstance under which an HDF user will need to be concerned with the difference between regular tag numbers and extended tag numbers. If a user bypasses the regular HDF interface to examine a raw HDF file, that user will have to know the extended tag numbers, their significance, and the alternative storage structures.

## Linked Block Elements

As mentioned above, data elements had to be stored as single contiguous blocks within the basic HDF file prior to HDF Version 3.2. This meant that if a data element grew larger than the allotted space, the file had to be erased from its current location and rewritten at the end of the file.

Linked blocks provide a convenient means of addressing this problem by linking new data blocks to a pre-existing data element. Linked block elements consist of a series of data blocks chained together in a linked list (similar to the DD list). The data blocks must be of uniform size, except for the first block, which is considered a special case.

The linked block data element is a description record beginning with the constant EXT\_LINKED, which identifies the linked block storage method. The rest of the record describes the organization of the data element stored as linked blocks. Figure 10c illustrates a linked block description record.

* Linked Block Description Record



*extended\_tag* The extended tag counterpart of any HDF standard tag (16-bit integer)

*ref\_no* Reference number (16-bit integer)

EXT\_LINKED Constant identifying this as a linked block description record (32-bit integer)

*length* Length of entire element (32-bit integer)

*first\_len* Length of the first data block (32-bit integer)

*blk\_len* Length of successive data blocks (32-bit integer)

*num\_blk*  Number of blocks per block table (32-bit integer)

*link\_ref*  Reference number of first block table (16-bit integer)

The *link\_ref* field of the description record gives the reference number of the first linked block table for the element. This table is identified by the tag/ref DFTAG\_LINKED/*link\_ref*  and contains *num\_blk* entries. There may be any number of linked block tables chained together to describe a linked block element. Figure 10d illustrates a linked block table.

* A Linked Block Table



*link\_ref* Reference number for this table (16-bit integer)

*next\_ref* Reference number for next table (16-bit integer)

*blk\_ref\_n* Reference number for data block (16-bit integer)

The *next\_ref* field contains the reference number of the next linked block table. A value of zero (0) in this field indicates that there are no additional linked block tables associated with this element.

The *blk\_ref\_n* fields of each linked block table contain reference numbers for the individual data blocks that make up the data portion of the linked block element. These data blocks are identified by the tag/ref DFTAG\_LINKED/*blk\_ref\_n* as illustrated in Figure 10e. Although it may seem ambiguous to use the same tag to refer to two different objects, this ambiguity is resolved by the context in which the tags appear.

* A Data Block



*blk\_ref\_n* Reference number for this data block (16-bit integer)

*data\_block* Block of actual data (size specified by *first\_len*  or *blk\_len* in the description record)

Linked block elements can be created using the function HLcreate(), which is discussed in Chapter , “Low-level Interface.”

## External Elements

External elements allow the data portion of an HDF element to reside in a separate file. The potential of external data elements is largely unexplored in the HDF context, although other file formats (most notably the Common Data Format, CDF, from NASA) have used external data elements to great advantage.

Because there has been little discussion of external elements within the HDF user community, the structure of these elements is still not completely defined. Figure 10f shows a diagram of the suggested structure for an external element.

* External Element Description Record



extended\_tag The extended tag counterpart of any HDF standard tag (16-bit integer)

ref\_no Reference number (16-bit integer)

SPECIAL\_EXT Constant identifying this as an external element description record (16-bit integer)

length Length in bytes of the data in the external file (32-bit integer)

offset Location of the data within the external file (32-bit integer)

filename Non-null terminated ASCII string naming the external file (any length)

An external element description record begins with the constant SPECIAL\_EXT, which identifies the data object as having an externally stored data element. The rest of the description record consists of the specific information required to retrieve the data.

External elements can be created using the function HXcreate(), which is discussed in Chapter , “Low-level Interface.”

## Chunked Data Storage

### Chunked Element Description Record

The file format, or layout, of a chunked data element is specified in a chunked element description record. Figure 10g, "DD for a chunked element (12 bytes) pointing to a chunked element description record (>52 bytes)," provides a complete description, via illustration, of this record.

The fields that define a chunked element, as illustrated in Figure 10g, are as follows:

sp\_tag\_desc SPECIAL\_CHUNKED (a 16-bit constant) identifies this as a chunked element description record.

sp\_tag\_head\_len Length of this special element header only (4 bytes). Does not include length of header with additional specialness headers. Note: This is done to make this header layout similar to the multiple specialness layout.

version Version information (8-bit field).

flag Bit field to set additional specialness (32-bit field). Only the bottom 8 bits are currently used.

elem\_tot\_len Valid logical length of the entire element (4 bytes). The logical physical length is this value multiplied by nt\_size. The actual physical length used for storage can be greater than the dataset size due to the presence of ghost areas in chunks. Partial chunks are not distinguished from regular chunks.

chunk\_size Logical size of data chunks (4 bytes).

nt\_size Number type size, i.e the size of the data type (4 bytes).

chk\_tbl\_tag Tag for the chunk table, i.e. the Vdata (2 bytes).

chk\_tbl\_ref Reference number for the chunk table, i.e. the Vdata (2 bytes).

sp\_tag For future use. Special table for 'ghost' chunks (2 bytes).

sp\_ref For future use (2 bytes).

ndims Number of dimensions of the chunked element.(4 bytes).

file\_val\_num\_bytes Number of bytes in fill value (4 bytes).

fill value Fill value (variable bytes).

.

|  |
| --- |
| * DD for a chunked element (12 bytes) pointing to a chunked element description record  (>52 bytes) |

In addition to the above fields, each chunked element dimension requires a set of the following fields:

flag (32-bit field) This field is divided as follows:  
| High, 8 bits | Medium High, 8 bits | Medium Low, 8 bits | Low, 8 bits |

* distrib\_type (Low 8 bits, bits 0-7)  
  Type of data distribution along this dimension   
   0x00 -> None  
   0x01 -> Block  
  Currently only block distribution is supported but this is not currently checked or verified.
* Other (Medium Low 8 bits, bits 7-15)   
   0x00 -> Regular dimension  
   0x01 -> UNLIMITED dimension

dim\_length Current length of this dimension (4 bytes).

chunk\_length Length of the chunk along this dimension (4 bytes).

Further, additional specialnesses may be used. Each additional specialness requires a set of the following fields:

sp\_tag\_desc SPECIAL\_xxx (16-bit constant) identifies this as an xxx element description record (16-bit field).

sp\_tag\_header\_len Length of special element header (4 bytes).

sp\_tag\_header Special header (variable bytes).

### Chunk Table

Information regarding a chunked data set is stored in the chunk table, described in Figure 10h on page 154.

The chunk table fields are defined as follows:

origin Specifies the coordinates of the chunk in the overall chunk array. This is a variable-size field, depending on the number of dimensions of the chunked element.

chunk\_tag Currently DFTAG\_CHUNK. Could be another chunked element to allow recursive chunked elements (DFTAG\_CHUNKED). (16-bit field)

chunk\_ref Reference number of the chunk itself. (16-bit field)

* Chunk table



## Data Compression

The HDF library supprts the following compression formats for scientific data sets.

* Skipping-Huffman
* GNU ZIP deflation (Lempel/Ziv-77 dictionary coder)
* N-bit run-length encoding
* SZIP

The compression format of a data set is specified in an extended tag description known as a compressed element description record. Figure 10i, "Compression header extended tag description," describes the common elements of this record. Subsequent figures describe the remainder of the record, which varies for each type of compression.

### Compression Header: The Common Elements of Compressed Element Description Records

The compression header comprises the common elements of all compressed element description records and is contained in the first ten fields of the record. As illustrated in Figure 10i, the compresion header is made up of the following fileds.

The first four fields of the compression header are common among all special element headers:

Extended tag

Reference # These two fields contain the tag/ref pair that identifies any HDF object.

Offset This is the offset, in bytes, to the location of the fifth field, or the sp\_tag\_desc  field, of the compression header. This field always contains the value SPECIAL\_COMP in a compressed element description record.

Length This field specifies the space requirement, in bytes, of the fifth through last fields of the compressed element description record.

The fifth through tenth fields are particular to the compression header:

sp\_tag\_desc SPECIAL\_COMP (a 16-bit constant) identifies this as a compressed element description record.

Version Version information (16-bit field).

Length of uncompressed data   
Length, in bytes of the uncompressed data.

Ref # of compressed data   
As illustrated in Figure 10j, "Compressed element reference number," this field contains a pointer to a DFTAG\_COMPRESSED structure which, in turn, provides the offset location and size, both in bytes, of the actual compressed data.

Model type Currently only streaming I/O.

Compression type   
A string identifying the type of compression in use.

The remainder of the compressed element description record is different for each type of compression. The following sections discuss each of those types of records in turn.

* Compression header extended tag description



* Compressed element reference number



### Compressed Element Description Record: NBIT Run-length Encoding

* Extended tag description for NBIT run-length encoding compression



### Compressed Element Description Record: Skipping-Huffman

* Extended tag description for Skipping-Huffman compression



### Compressed Element Description Record: GNU ZIP (Deflate)

* Extended tag description for GNU ZIP (deflate) compression



### Compressed Element Description Record: SZIP

* Compression header extended tag description



The following parameters are used in SZIP compression.

Pixels: Number of pixels, or data elements, in the SDS to be compressed and must be greater than 0. It is computed by dim[0]\*dim[1]\*...\*dim[n], where n is the number of dimensions.

Pixels per scanline: Number of pixels per scan line. This value must be greater than or equal to pixels per block, and smaller than or equal to SZ\_MAX\_PIXELS\_PER\_SCANLINE. SZ\_MAX\_PIXELS\_PER\_SCANLINE is defined as:

SZ\_MAX\_PIXELS\_PER\_SCANLINE =

SZ\_MAX\_BLOCKS\_PER\_SCANLINE \* SZ\_MAX\_PIXELS\_PER\_BLOCK,

where:

SZ\_MAX\_BLOCKS\_PER\_SCANLINE = 128 and SZ\_MAX\_PIXELS\_PER\_BLOCK = 32

Options mask Szip encoding scheme and other options. This parameter combines a bitwise or of any of the following values:

SZ\_ALLOW\_K13\_OPTION\_MASK (or 1)

SZ\_CHIP\_OPTION\_MASK (or 2)

SZ\_EC\_OPTION\_MASK (or 4)

SZ\_LSB\_OPTION\_MASK (or 8)

SZ\_MSB\_OPTION\_MASK (or 16)

SZ\_NN\_OPTION\_MASK (or 32)

SZ\_RAW\_OPTION\_MASK (or 128)

Bits per pixel: The number of bits in the SDS number type, e.g., if the SDS’ number type is DFNT\_FLOAT, the bits per pixel of this SDS will be 32. This parameter must be either 8, 16, 32, or 64.

Pixels per block: Number of data elements in an szip block. Must be even and smaller than or equal to pixels per scanline and smaller than and equal to SZ\_MAX\_PIXELS\_PER\_BLOCK (32.)

The two parameters Options mask and Pixels per block are required when setting compression for SZIP. If any of the other parameters are not provided, they will be computed by HCPsetup\_szip\_parms.

The SZIP source code can be found at https://support.hdfgroup.org/doc\_resource/SZIP/ for further reference.