# HDF4.2r1 SZIP Release Notes

SZIP compression was first available in HDF4.2r0. HDF4.2r1 has substantial bug fixes and changes in the use of SZIP. This document summarizes the important changes in HDF4.2r1 support for SZIP compression.

SZIP compression was nearly completely broken in HDF4.2r0.[1] Users should upgrade to HDF4.2r1 as soon as possible. Any data compressed with SZIP using HDF4.2r0 may or may not be accessible.

Szip has been extensively tested with HDF4.2r1, but has not been user-tested. Users are encouraged to use Szip and report any problems to NCSA as soon as possible.

## 1. Capability

SZIP compression can be used to as an option to compress data in an SDS under certain circumstances. SZIP compression is *not available for GR images or any other type of object in HDF4.*

SZIP compression is optional. The HDF4 library can be built without SZIP, with SZIP decoder only (read only), or with SZIP encoder and decoder.

SZIP can be used for data of any SDS data type, and for fixed-dimension arrays of any size or dimensions. However, a dataset with an unlimited dimension cannot be compressed with SZIP or any other compression method.

SZIP can be used for regular and chunked datasets, with the following limitation. When a regular (non-chunked) SDS is compressed, it can be written once and then read; but cannot be updated. A chunked dataset can be updated. (This restriction is similar to the behavior of Deflate (GZIP) compression.)

The SZIP compression algorithm requires several parameters. When used with HDF4 only two parameters need to be set by the calling program:

- Entropy or Nearest Neighbor coding
- Pixels per block

All other parameters are set by the HDF4 library, as explained below.

## 2. Usage

SZIP is used the same as other compression methods in HDF4.

[1] It is important to note that the problems were in the integration of SZIP support in the HDF4 library. The SZIP library did not have significant problems, and the HDF5 integration of SZIP does not have problems.

## 2.1. Applying SZIP compression

SZIP compression is applied to an SDS by calling `SDsetcompress` or `SDsetchunk` functions with the appropriate settings. The compression type is set to `COMP_CODE_SZIP` and set the two SZIP parameters, `pixels_per_block`, and the encoding option in the `comp_info` structure.

Figure 1 shows the SZIP parameters of the `comp_info` union. Table 1 defines the parameters.

```
typedef union tag_comp_info
  {
      struct
        {
            int32 options_mask;   /* IN */
            int32 pixels_per_block;   /* IN */
            int32 pixels_per_scanline; /* OUT: computed */
            int32 bits_per_pixel; /* OUT: size of NT */
            int32 pixels; /* OUT: size of dataset or chunk */
         }
       szip;  /* for szip encoding */

  } comp_info;
```

**Figure 1. The comp_info for SZIP.**

**Table 1. SZIP parameters (* = automatically set by the HDF library during `SDsetcompress` call.)**

| Parameter | IN/OUT | Definition |
|---|---|---|
| `pixels_per_block` | IN | Number of data elements in SZIP block (2-32) |
| `options_mask` | IN | Szip encoding scheme and other options. |
| `bits_per_pixel` | OUT* | Number of bits in the HDF data type, e.g., `DFNT_FLOAT` will be 32 bits per pixel |
| `pixels` | OUT* | Number of elements in the SDS, dim[0] * dim[1] * … |
| `pixels_per_scanline` | OUT* | Set according to heuristic. |

Figure 2 shows a sketch of the code to enable SZIP compression for an SDS. In the example, the SZIP parameters are set to use Entropy Coding and 2 pixels per block. The `SDsetcompress` call checks that SZIP is available, and makes sure that the parameters are correct. Other parameters to SZIP are set automatically by the HDF library.

Similar calls are made to set SZIP compression and chunking for a chunked dataset.

```
#include “hdf.h”
#include “szlib.h”

comp_info   c_info;

    /* open file, create SDS, etc. */

    /* Initialize SZIP Compression:
      in this case, select entropy coding */
    c_info.szip.pixels_per_block = 2;
    c_info.szip.options_mask |= SZ_EC_OPTION_MASK;

    /* enable compression */
    status = SDsetcompress (sds_id, COMP_CODE_SZIP, &c_info);
```

**Figure 2. Sketch of code to enable SZIP compression for an SDS dataset.**

## 2.2. Writing and Reading Data

A compressed dataset is written with `SDwritedata` (or `SDwritechunk`), and read with `SDreaddata` (or `SDreadchunk`). These calls are the same for compressed or uncompressed data.

The compression parameters can be discovered with `Sdgetcompress` (or `SDgetchunk`), which returns the `comp_type`, `COMP_CODE_SZIP`, and the `comp_info` union (Figure 3). The compression information includes the parameters automatically set by the library. Table 1 above describes the fields in the `comp_info` returned by `SDgetcompress`.

```
        /* Sdstart, Sdselect, etc. */

        status = SDgetcompress (sds_id, &c_type, &c_info);

        if (c_type == COMP_CODE_SZIP) {
             printf("SZIP coder params:\n");
         printf("  pixels_per_block = %d\n",  c_info.szip.pixels_per_block );
           if ( c_info.szip.options_mask & SZ_NN_OPTION_MASK) {
             printf("NN option\n");
           } else if ( c_info.szip.options_mask & SZ_EC_OPTION_MASK) {
             printf("EC option\n");
           }
         printf("  pixels_per_scanline = %d\n", c_info.szip.pixels_per_scanline
);
         printf("  bits_per_pixel = %d\n", c_info.szip.bits_per_pixel );
         printf("  pixels = %d\n",c_info.szip.pixels );
       }
```

**Figure 3. Reading SZIP compression Information.**

## 2.3. Configurations of SZIP

As discussed above, the SZIP library can be configured with encoding disabled or enabled. A program can discover if the SZIP encoder is enabled by calling the HDF4 function `HCget_config_info`. This function returns an integer, with bits set to indicate if the decoder and encoder are present. (If SZIP is not available at all, both the decoder and encoder will be reported to be disabled.) This function works for any compression method, although SZIP is the only compression method to date that has an optional encoder. Figure 4 shows a sketch of how this function can be used.

```
uint32    comp_config;

    status = HCget_config_info( COMP_CODE_SZIP , &comp_config);
    if ((comp_config & COMP_DECODER_ENABLED|COMP_ENCODER_ENABLED) == 0)
    {
      /* coder not present?? */
      printf("SZIP not configured\n");
      exit(1);
    }
    if ((comp_config & COMP_DECODER_ENABLED) == 0) {
      /* decoder not present?? */
      printf("SZIP encoding not allowed\n");
      exit(1);
    }

    /* SZIP is present and encoder works */
```

**Figure 4. Sketch of code to detect the presence of SZIP, and the SZIP encoder.**

The HDF4 library checks for the presence of SZIP and the availability of SZIP encoding, and returns failure if the operation cannot be performed. There are three possibilities: no SZIP library at all, SZIP decoder only, and SZIP encoder and decoder. Table 2 summarizes the behavior of the HDF functions in these cases.

It is not possible to create a new dataset with SZIP compression (`SDsetcompress`) unless SZIP encoding is available. When SZIP is not available, or encoding is disabled, `SDsetcompress` will fail, and the dataset will not be compressed with SZIP.

Even when a program is compiled without SZIP or with decode only, it still may open files that contain SDS datasets compressed with SZIP. When the SZIP encoder is disabled, data can be read and decompressed.[2] Also, the SZIP parameters used to compress the dataset can be read from the file.

A dataset cannot be written (updated) unless SZIP encoding is enabled. If a program opens a dataset that was compressed with SZIP, and then tries to write data into it, the write will fail and no data will be written if encoding is disabled.

[2] The file must be opened read only, i.e., `SDstart(FILE, DFACC_RDONLY)`. If the file is opened with `DFACC_RDWR` the `SDreaddata` will fail.

While compression is only available through the SD interface, the data in the datasets may also be read and updated through the old DFSD interface. *It is not certain that the DFSD interface will always work correctly with SZIP, so users are advised not to use the DFSD interface.*

**Table 2. Behavior of HDF functions, depending on SZIP configuration.**

| **SD function** | **SZIP available (compress and decompress)** | **SZIP encoder not available (decompress only)** | **No SZIP available** |
|---|---|---|---|
| `SDsetcompress` | Checks the input parameters, and sets:<br>• bits per pixel to the size of the HDF data type<br>• pixels to the number of data elements, i.e., the dimensions of the SDS<br>• pixels_per_block to an appropriate value.<br>Succeed. | Fail | Fail |
| `SDreaddata`, `SDreadchunk` | Succeed | Succeed (if `DFACC_RDONLY`) | Fail |
| `SDgetcompress`, `SDgetchunk` | Succeed | Succeed | Succeed |
| `SDwritedata` | If not chunked, partial write fails.<br>Otherwise, succeed. | Fail (This can occur if a program opens a file created by another program.) | Fail |
| `SDwritechunk` | Succeed. | Fail | Fail |
| Other functions | Succeed | Succeed | Succeed |

## 3. Compatibility Issues

There have been several important fixes and improvements, so users should upgrade to SZIP 2.0 and HDF4.2r1 as soon as possible.

### 3.1. File Format Changes between HDF4.2r0 and HDF4.2r1

The format changes are documented in the File Format Specification.

Briefly, in the compressed data element, the compressed data is prefixed by two fields:
1) 1 byte, 0 == compressed, 1== not compressed.
2) 4 bytes, size of the compressed data

In some unfavorable cases, the compressed data may be larger than the original data. When this happens, the original data is written and the first byte is set to '1'. The next 4 bytes indicate the number of bytes of good data, which may be smaller than the allocated space. When data is updated, the compressed data may be smaller than the previous compressed data.

In addition, one byte of the SZIP extended tag `options_mask` is reserved for HDF. One bit of this byte is set to indicate that the new format (HDF4.2r1) is used in the data.

These changes are transparent to user programs.

## 3.2. Changes to Program Source and Linking

HDF4.2r1 requires SZIP2.0 and will not link correctly with earlier versions of SZIP.

In HDF4.2r1 the `comp_info` structure for SZIP parameters was modified. Some source code may need to be modified to remove references to the field `c_info.szip.compression_mode.`
In HDF4.2r1 SZIP parameters had to be coded into the application. Source code that sets SZIP compression should be changed to include "szlib.h", and use the SZIP parameters defined in that file (e.g., see Figure 2, above).

In 4.2r0 all the SZIP parameters needed to be set before calling `SDsetcompress`. This is not necessary in HDF4.2r1, so program source should be changed to set only the `pixels_per_block` and `options_mask`, as in Figure 2.

## 3.3. Data compatibility

There were serious bugs and limitations in HDF4.2r0. In many cases, data could not be successfully written with SZIP, or could not be read after writing with SZIP. *Any data that was written using SZIP in HDF4.2r0 should be examined very carefully to make sure it is valid.*

HDF4.2r1 fixed many of the bugs from HDF4.2.0, and corrected many limitations. These fixes required changes to the file format, which are incompatible with HDF4.2r0.

*Programs compiled with HDF4.2r0 cannot decompress SDS datasets compressed with SZIP created by HDF4.2r1. Programs compiled with HDF4.2r1 should be able to decompress data compressed with SZIP by HDF4.2r0.* However, given the severe problems in the earlier code, data written with HDF4.2r0 might or might not be readable. Table 3 summarizes the compatibility of data between the old and new library.

**Table 3. Compatibility of Data between HDF4.1r0 and HDF4.2r1**

| | **Written with HDF4.2r0 (old)** | **Written with HDF4.2r1 (new)** |
|---|---|---|
| **Read with HDF4.2r0 (old)** | ? (bugs) | No |
| **Read with HDF4.2r1 (new)** | Yes (if no bug when written) | Yes |
| **Update with HDF4.2r1 (new) (chunked data)** | Yes (if no bug when written, writes in new format) | Yes |