HDF5 Compression Troubleshooting

Elena Pourmal

The purpose of this technical note is to help HDF5 users with troubleshooting problems with HDF5 filters, especially with compression filters. The document assumes that the reader knows HDF5 basics and is aware of the compression feature in HDF5.

The HDF Group

Contents

1. Introduction

One of the most powerful features of HDF5 is the ability to modify, or "filter," data during I/O[1]. Filters provided by the HDF5 Library, "predefined filters," include several types of data compression[2], data shuffling and checksum. Users can implement their own "user-defined filters" and use them with the HDF5 Library.

By far the most common user-defined filters are ones that perform data compression. While the programming model and usage of the compression filters are straightforward, it is easy, especially for novice users, to overlook important details when implementing compression filters and to end up with data that is not modified as they would expect.

The purpose of this document is to describe how to diagnose situations where the data in a file is not compressed as expected.

[1] I/O is short for input/output.
[2] In HDF5 documentation including this document, "compression" is often referred to as "compression filter" or "filter".

2. An Overview of HDF5 Compression Troubleshooting

Sometimes users may find that HDF5 data was not compressed in a file or that the compression ratio is very small. By themselves, these results do not mean that compression did not work or did not work well. These results suggest that something might have gone wrong when a compression filter was applied. How can users determine the true cause of the problem?

There are two major reasons why a filter did not produce the desired result: it was not applied, or it was not effective.

The filter was not applied

If a filter was not applied at all, then it was not included at compile time when the library was built or was not found at run time for dynamically loaded filters.

The absence or presence of HDF5 predefined filters can be confirmed by examining the installed HDF5 files or by using HDF5 API calls. The absence or presence of all filter types can be confirmed by running HDF5 command-line tools on the produced HDF5 files. See “If a Filter Was Not Applied” on page 6 for more information.

The filter was applied but was not effective

The effectiveness of compression filters is a complex matter and is only briefly covered this document. See “If a Compression Filter Was Not Effective” on page 18 for more information. This section gives a short overview of the problem and provides an example in which the advantages of different compression filters and their combinations are shown.

3. If a Filter Was Not Applied

This section discusses how it may happen that a compression filter is not available to an application and describes the behavior of the HDF5 Library in the absence of the filter. Then we walk through how to troubleshoot the problem by checking the HDF5 installation, by examining what an application can do at run time to see if a filter is available, and by using some HDF5 command line tools to see if a filter was applied.

Note that in addition to `gzip` and `Szip`, there are four other internal predefined filters: `shuffle`, `fletcher32`, `scaleoffset`, and `nbit`. These are enabled by default by both `configure` and `CMake` builds. While these filters can be disabled intentionally with the `configure` flag `–disable-filters`, disabling them is not recommended. The discussion and the examples in this document focus on compression filters, but everything said can be applied to other missing internal filters as well.

3.1. How the HDF5 Library Configuration May Miss a Compression Filter

The HDF5 Library uses external libraries for data compression. The two predefined compression methods are `gzip`[3] and `Szip`, and these can be requested at the HDF5 Library configuration time (compile time). User-defined compression filters and the corresponding libraries are usually linked with an application or provided as a dynamically loaded library.

`gzip` and `Szip` require the `libz.a(so)` and `libsz.a(so)` libraries, respectively, to be present on the system and to be enabled during HDF5 configuration with this `configure` command:

```
./configure –with-zlib=/path… –with-szlib=/path… <other flags>
```

There is one important difference in the behavior of `configure` between `gzip` and `Szip`.

On Unix systems, *when GNU Autotools are used to build HDF5,* `gzip` compression is enabled *automatically* if the `zlib` library is present on the system in default locations without explicitly specifying `–with-zlib=/path`. For example, if `libz.so` is installed under `/usr/lib` with the header under `/usr/include` or under `/usr/local/lib` with the header under `/usr/local/include`, the following HDF5 `configure` command will find the `gzip` library and will configure the compression filter in:

```
./configure
```

[3] `gzip` compression is called with the `H5Pset_deflate` function. `Deflate` compression and `gzip` compression refer to the same operation, and we generally use the germ `gzip` rather than `deflate`. For more information, see the "H5P: Property List Interface" page of the *HDF5 Reference Manual* for the entry for the `H5Pset_deflate` function call.

The Szip compression should *always* be *requested* with the configure flag shown above. configure will not fail if libraries supporting the requested compression method are not found, for example, because a specified path was not correct, or the library is missing.

When the library is built with CMake, gzip, and Szip compression filters are enabled by default in the source code distribution's config/cmake/cacheinit.cmake file.[4] See the CMake installation instructions for the locations of the libraries.

If compression is not requested or found at configuration time, the compression method is not registered with the library and cannot be applied when data is written or read. For example, the h5repack tool will not be able to remove an Szip compression filter from a dataset if the Szip library was not configured into the library against which the tool was built. The next section discusses the behavior of the HDF5 Library in the absence of filters.

3.2. How Does the HDF5 Library Behave in the Absence of a Filter

By design, the HDF5 Library allows applications to create and write datasets using filters that are not available at creation/write time. This feature makes it possible to create HDF5 files on one system and to write data on another system where the HDF5 Library is configured with or without the requested filter.

Let' s recall the HDF5 programming model for enabling filters.

An HDF5 application uses one or more H5Pset_<filter> calls to configure a dataset's filter pipeline at its creation time. The excerpt below shows how a gzip filter is added to a pipeline[5] with H5Pset_deflate.

```
/*
 * Create the dataset creation property list, add the gzip
 * compression filter and set the chunk size.
 */
dcpl = H5Pcreate (H5P_DATASET_CREATE);
status = H5Pset_deflate (dcpl, 9);
status = H5Pset_chunk (dcpl, 2, chunk);
dset = H5Dcreate (file, DATASET,…, dcpl,…);
```

For all internal filters (shuffle, fletcher32, scaleoffset, and nbit) and the external gzip filter, the HDF5 Library does **not** check to see if the filter is registered when the corresponding H5Pset_<filter> function is called. The only exception to this rule is H5Pset_szip which will fail if Szip was not configured in or is configured with a decoder only. Hence, in the example above,

[4] Users can overwrite the defaults by using –DHDF5_ENABLE_SZIP_SUPPORT:BOOL=OFF –DHDF5_ENABLE_Z_LIB_SUPPORT:BOOL=OFF with the "cmake –C" command. See the INSTALL_CMake.txt file under the release_docs directory in the HDF5 source distribution.

[5] See the complete set of HDF5 examples at The HDF Group website.

`H5Pset_deflate` will succeed. The specified filter will be added to the dataset's filter pipeline and will be applied to any data written to this dataset[6].

When `H5Pset_<filter>` is called, a record for the filter is added to the dataset's object header in the file, and information about the filter can be queried with the HDF5 APIs and displayed by HDF5 tools such as `h5dump`. The presence of filter information in a dataset's header does not mean that the filter was actually applied to the dataset's data, as will be explained later in this document. See "How to Use HDF5 Tools to Investigate Missing Compression Filters" on page 12 for more information on how to use `h5ls` and `h5debug` to determine if the filter was actually applied.

The success of further write operations to a dataset when filters are missing depends on the filter type.

By design, an HDF5 filter can be optional or required. This filter mode defines the behavior of the HDF5 Library during write operations. In the absence of an optional filter, `H5Dwrite` calls will succeed and data will be written to the file, bypassing the filter. A missing required filter will cause `H5Dwrite` calls to fail. Clearly, `H5Dread` calls will fail when filters that are needed to decode the data are missing.

The HDF5 Library has only one required internal filter, `Fletcher32` (checksum creation), and one required external filter, `Szip`. As mentioned earlier, only the `Szip` compression (`H5Pset_szip`) will flag the absence of the filter. If, despite the missing filter, an application goes on to create a dataset via `H5Dcreate`, the call will succeed, but the `Szip` filter will **not** be added to the filter pipeline. This behavior is different from all other filters that may not be present, but will be added to the filter pipeline and applied during I/O. See the "Using HDF5 APIs" section on page 11 for more information on how to determine if a filter is available and to avoid writing data while the filter is missing.

Developers who create their own filters should use the "`flags`" parameter in `H5Pset_filter` to declare if the filter is optional or required. The filter type can be determined by calling `H5Pget_filter` and checking the value of the "`flags`" parameter.

For more information on filter behavior in HDF5, see "Filters in HDF5".

3.3. How to Determine if the HDF5 Library was Configured with a Needed Compression Filter

The previous section described how the HDF5 Library could be configured without certain compression filters and the resulting expected library behavior.

The following subsections explain how to determine if a compression method is configured in the HDF5 Library and how to avoid accessing data if the filter is missing.

[6] In this document, we talk only about dataset compression. Compression can be also applied to HDF5 groups to compress the link names stored in local heaps. The behavior of the library when filters are missing is the same for groups as datasets.

3.3.1. Examine the hdf5lib.settings File

To see how the library was configured and built, users should examine the `hdf5lib.settings` text file found in the `lib` directory of the HDF5 installation point and search for the lines that contain the "`I/O filters`" string. The `hdf5lib.settings` file is automatically generated at configuration time when the HDF5 Library is built with `configure` on Unix or with `CMake` on Unix and Windows, and it should contain the following lines:

```
I/O filters (external): deflate(zlib),szip(encoder)
I/O filters (internal): shuffle,fletcher32,nbit,scaleoffset
```

The same lines in the file generated by `CMake` look slightly different:

```
I/O filters (external): DEFLATE ENCODE DECODE
I/O filters (internal): SHUFFLE FLETCHER32 NBIT SCALEOFFSET
```

"`ENCODE DECODE`" indicates that both the `Szip` compression encoder and decoder are present. This inconsistency between configure and `CMake` generated files will be removed in a future release.
These lines show the compression libraries configured with HDF5. Here is an example of the same output when external compression filters are absent:

```
I/O filters (external):
I/O filters (internal): shuffle,fletcher32,nbit,scaleoffset
```

Depending on the values listed on the I/O filters (external) line, users will be able to tell if their HDF5 files are compressed appropriately. If `Szip` is not included in the build, data files will not be compressed with `Szip`. If `gzip` is not included in the build and is not installed on the system, then data files will not be compressed with `gzip`.

If the `hdf5lib.settings` file is not present on the system, then users can examine a public header file or the library binary file to find out if a filter is present, as is discussed in the next two sections.

3.3.2. Examine the H5pubconf.h Header File

To see if a filter is present, users can also inspect the HDF5 public header file[7] installed under the `include` directory of the HDF5 installation point. If the compression and internal filters are present, the corresponding symbols will be defined as follows:

```
/* Define if support for deflate (zlib) filter is enabled */
#define H5_HAVE_FILTER_DEFLATE 1

/* Define if support for Fletcher32 checksum is enabled */
#define H5_HAVE_FILTER_FLETCHER32 1
```

[7] Starting with the HDF5 1.8.15 release, the internals filters will always be present; therefore, the corresponding defines will not be in the header file.

```
/* Define if support for nbit filter is enabled */
#define H5_HAVE_FILTER_NBIT 1

/* Define if support for scaleoffset filter is enabled */
#define H5_HAVE_FILTER_SCALEOFFSET 1

/* Define if support for shuffle filter is enabled */
#define H5_HAVE_FILTER_SHUFFLE 1

/* Define if support for Szip filter is enabled */
#define H5_HAVE_FILTER_SZIP 1
```

If a compression or internal filter was not configured, the corresponding lines will be commented out as follows:

```
/* Define if support for deflate (zlib) filter is enabled */
/* #undef H5_HAVE_FILTER_DEFLATE */
```

3.3.3. Check the HDF5 Library's Binary

The HDF5 Library's binary contains summary output similar to what is stored in the `hdf5lib.settings` file. Users can run the Unix "`strings`" command to get information about the configured filters:

```
% strings libhdf5.a(so) | grep "I/O filters ("
      I/O filters (external): deflate(zlib),szip(encoder)
      I/O filters (internal): shuffle,fletcher32,nbit,scaleoffset
```

When compression filters are not configured, the output of the command above will be:

```
I/O filters (external):
I/O filters (internal): shuffle,fletcher32,nbit,scaleoffset
```

On Windows one can use the `dumpbin /all` command, and then view and search the output for strings like "`DEFLATE`", "`FLETCHER32`", "`DECODE`", and "`ENCODE.`"

```
.....
  10201860: 4E 0A 20 20 20 20 20 20 20 20 20 49 2F 4F 20 66  N.         I/O f
  10201870: 69 6C 74 65 72 73 20 28 65 78 74 65 72 6E 61 6C  ilters (external
  10201880: 29 3A 20 20 44 45 46 4C 41 54 45 20 44 45 43 4F  ):  DEFLATE DECO
  10201890: 44 45 20 45 4E 43 4F 44 45 0A 20 20 20 20 20 20  DE ENCODE.
  102018A0: 20 20 20 49 2F 4F 20 66 69 6C 74 65 72 73 20 28     I/O filters (
  102018B0: 69 6E 74 65 72 6E 61 6C 29 3A 20 20 53 48 55 46  internal):  SHUF
  102018C0: 46 4C 45 20 46 4C 45 54 43 48 45 52 33 32 20 4E  FLE FLETCHER32 N
  102018D0: 42 49 54 20 53 43 41 4C 45 4F 46 46 53 45 54 0A  BIT SCALEOFFSET.
```

3.3.4. Check the Compiler Script

Developers can also use the compiler scripts such as `h5cc` to verify that a compression library is present and configured in. Use the "`- show`" option with any of the compilers scripts found in the bin subdirectory of the HDF5 installation directory. The presence of `-lsz` and `-lz` options among the linker flags will confirm that `Szip` or `gzip` were compiled with the HDF5 Library. See the sample below.

```
$ h5cc -show
gcc -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_BSD_SOURCE -
L/mnt/hdf/packages/hdf5/v1812/Linux64_2.6/standard/lib
/mnt/hdf/packages/hdf5/v1812/Linux64_2.6/standard/lib/libhdf5_hl.a
/mnt/hdf/packages/hdf5/v1812/Linux64_2.6/standard/lib/libhdf5.a -lsz -lz -lrt
-ldl -lm -Wl,-rpath -Wl,/mnt/hdf/packages/hdf5/v1812/Linux64_2.6/standard/lib
```

3.3.5. Using HDF5 APIs

Applications can check filter availability at run time. In order to check the filter's availability with the HDF5 Library, users should know the filter identifier (for example, `H5Z_FILTER_DEFLATE`) and call the `H5Zfilter_avail` function as shown in the example below. Use `H5Zget_filter_info` to determine if the filter is configured to decode data, to encode data, neither, or both.

```
/*
 * Check if gzip compression is available and can be used for both
 * compression and decompression.
 */
avail = H5Zfilter_avail(H5Z_FILTER_DEFLATE);
if (!avail) {
    printf ("gzip filter not available.\n");
    return 1;
}
status = H5Zget_filter_info (H5Z_FILTER_DEFLATE, &filter_info);
if ( !(filter_info & H5Z_FILTER_CONFIG_ENCODE_ENABLED) ||
            !(filter_info & H5Z_FILTER_CONFIG_DECODE_ENABLED) ) {
    printf ("gzip filter not available for encoding and decoding.\n");
    return 1;
}
```

`H5Zfilter_avail` can be used to find filters that are registered with the library or are available via dynamically loaded libraries. For more information, see "HDF5 Dynamically Loaded Filters."

Currently there is no HDF5 API call to retrieve a list of all of the registered or dynamically loaded filters. The default installation directories for HDF5 dynamically loaded filters are "`/usr/local/hdf5/lib/plugin`" on Unix and "`%ALLUSERSPROFILE%\hdf5\lib\plugin`" on Windows. Users can also check to see if the environment variable `HDF5_PLUGIN_PATH` is set on the system and refers to a directory with available plugins.

3.4. How to Use HDF5 Tools to Investigate Missing Compression Filters

In this section, we will use the `h5dump`, `h5ls`, and `h5debug` command-line utilities to see if a file was created with an HDF5 Library that did or did not have a compression filter configured in. For more information on these tools, see the "HDF5 Tools" page in the *HDF5 Reference Manual*.

3.4.1. How to Use h5dump to Examine Files with Compressed Data

The h5dump command-line tool can be used to see if a file uses a compression filter. The tool has two flags that will limit the output: the `-p` flag causes dataset properties including compression filters to be displayed, and the `-H` flag is used to suppress the output of data. The program provided in the "Example Program" section on page 16 creates a file called `h5ex_d_gzip.h5`. The output of `h5dump` shows that the gzip compression filter set to level 9 was added to the `DS1` dataset filter pipeline at creation time.

```
$ hdf5/bin/h5dump -p -H *.h5
HDF5 "h5ex_d_gzip.h5" {
GROUP "/" {
   DATASET "DS1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 32, 64 ) / ( 32, 64 ) }
      STORAGE_LAYOUT {
         CHUNKED ( 5, 9 )
         SIZE 5018 (1.633:1 COMPRESSION)
      }
      FILTERS {
         COMPRESSION DEFLATE { LEVEL 9 }
      }
      FILLVALUE {
         FILL_TIME H5D_FILL_TIME_IFSET
         VALUE  0
      }
      ALLOCATION_TIME {
         H5D_ALLOC_TIME_INCR
      }
   }
}
}
```

The output also shows a compression ratio defined as (original size)/(storage size). The size of the stored data is 5018 bytes vs. 8192 bytes of uncompressed data, a ratio of 1.663. This shows that the filter was successfully applied.

Now let's look at what happens when the same program is linked against an HDF5 Library that was not configured with the `gzip` library.

Notice that some chunks are only partially filled. 56 chunks (7 along the first dimension and 8 along the second dimension) are required to store the data. Since no compression was applied, each chunk has

size 5x9x4 = 180 bytes, resulting in a total storage size of 10,080 bytes. With an original size of 8192 bytes, the compression ratio is 0.813 (in other words, less than 1) and visible in the output below.

```
$ hdf5/bin/h5dump -p -H *.h5
HDF5 "h5ex_d_gzip.h5" {
GROUP "/" {
   DATASET "DS1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 32, 64 ) / ( 32, 64 ) }
      STORAGE_LAYOUT {
         CHUNKED ( 5, 9 )
         SIZE 10080 (0.813:1 COMPRESSION)
      }
      FILTERS {
         COMPRESSION DEFLATE { LEVEL 9 }
      }
      FILLVALUE {
         FILL_TIME H5D_FILL_TIME_IFSET
         VALUE  0
      }
      ALLOCATION_TIME {
         H5D_ALLOC_TIME_INCR
      }
   }
}
}
```

As discussed in the "How Does the HDF Library Behave in the Absence of a Filter" section on page 7, the presence of a filter in an object's filter pipeline **does not imply** that it will be applied unconditionally when data is written.

If the compression ratio is less than 1, compression is not applied. If it is 1, and compression is shown by `h5dump`, more investigation is needed; this will be discussed in the next section.

3.4.2. How to Use h5ls and h5debug to Find a Missing Compression Filter

Filters operate on chunked datasets. A filter may be ineffective for one chunk (for example, the compressed data is bigger than the original data), and succeed on another. How can users discern if a filter is missing or just ineffective (and as a result non-compressed data was written)? The `h5ls` and `h5debug` command-line tools can be used to investigate the issue.

First, let's take a look at what kind of information h5ls displays about the dataset `DS1` in our example file, which was written with an HDF5 library that has the `deflate` filter configured in:

```
$ h5ls -vr h5ex_d_gzip.h5
Opened "h5ex_d_gzip.h5" with sec2 driver.
/                        Group
    Location:  1:96
    Links:     1
/DS1                     Dataset {32/32, 64/64}
```

```
    Location:  1:800
    Links:     1
    Chunks:    {5, 9} 180 bytes
    Storage:   8192 logical bytes, 5018 allocated bytes, 163.25% utilization
    Filter-0:  deflate-1 OPT {9}
    Type:      native int
```

We see output similar to `h5dump` output with the compression ratio at 163%.

Now let's compare this output with another dataset `DS1`, but this time the dataset was written with a program linked against an HDF5 library without the `gzip` filter present.

```
$ h5ls -vr h5ex_d_gzip.h5
Opened "h5ex_d_gzip.h5" with sec2 driver.
/                        Group
    Location:  1:96
    Links:     1
/DS1                     Dataset {32/32, 64/64}
    Location:  1:800
    Links:     1
    Chunks:    {5, 9} 180 bytes
    Storage:   8192 logical bytes, 10080 allocated bytes, 81.27% utilization
    Filter-0:  deflate-1 OPT {9}
    Type:      native int
```

The `h5ls` output above shows that the `gzip` filter was added to the filter pipeline of the dataset `DS1`. It also shows that the compression ratio is less than 1. We can confirm by using h5debug that the filter was not applied at all, and, as a result of the missing filter, the individual chunks were not compressed.

From the `h5ls` output we know that the dataset object header is located at address 800. We retrieve the dataset object header at address 800 and search the layout message for the address of the chunk index B-tree as shown in the excerpt of the `h5debug` output below:

```
$ h5debug h5ex_d_gzip.h5 800
Reading signature at address 800 (rel)
Object Header...
…..
Message 4...
   Message ID (sequence number):                  0x0008 `layout' (0)
   Dirty:                                         FALSE
   Message flags:                                 <C>
   Chunk number:                                  0
   Raw message data (offset, size) in chunk:      (144, 24) bytes
   Message Information:
      Version:                                    3
      Type:                                       Chunked
      Number of dimensions:                       3
      Size:                                       {5, 9, 4}
      Index Type:                                 v1 B-tree
      B-tree address:                             1400
```

Now we can retrieve the B-tree information:

```
$ h5debug h5ex_d_gzip.h5 1400 3 [8]
Reading signature at address 1400 (rel)
Tree type ID:                                   H5B_CHUNK_ID
Size of node:                                   2616
Size of raw (disk) key:                         32
Dirty flag:                                     False
Level:                                          0
Address of left sibling:                        UNDEF
Address of right sibling:                       UNDEF
Number of children (max):                       56 (64)
Child 0...
   Address:                                     4016
   Left Key:
      Chunk size:                               180 bytes
      Filter mask:                              0x00000001
      Logical offset:                           {0, 0, 0}
   Right Key:
      Chunk size:                               180 bytes
      Filter mask:                              0x00000001
      Logical offset:                           {0, 9, 0}
Child 1...
   Address:                                     4196
   Left Key:
      Chunk size:                               180 bytes
```

We see that the size of each chunk is 180 bytes: in other words, compression was not successful. The filter mask value `0x00000001` indicates that filter was not applied. For more information on the filter mask, see the "Version 1 B-trees (B-link trees)" section in the *HDF5 File Format Specification*.

[8] Users have to supply the chunk rank. According to the *HDF5 File Format Specification*, this is the dataset rank plus 1; in other words, 3.

3.5. Example Program

The example program used to create the file discussed in this document is a modified version of the program available at http://www.hdfgroup.org/ftp/HDF5/examples/examples-by-api/hdf5-examples/1_8/C/H5D/h5ex_d_gzip.c. It was modified to have chunk dimensions not be factors of the dataset dimensions. Chunk dimensions were chosen for demonstration purposes only and are not recommended for real applications.

```
#include <stdlib.h>

#define FILE            "h5ex_d_gzip.h5"
#define DATASET         "DS1"
#define DIM0            32
#define DIM1            64
#define CHUNK0          5
#define CHUNK1          9

int
main (void)
{
    hid_t           file, space, dset, dcpl;    /* Handles */
    herr_t          status;
    htri_t          avail;
    H5Z_filter_t    filter_type;
    hsize_t         dims[2] = {DIM0, DIM1},
                    chunk[2] = {CHUNK0, CHUNK1};
    size_t          nelmts;
    unsigned int    flags,
                    filter_info;
    int             wdata[DIM0][DIM1],          /* Write buffer */
                    rdata[DIM0][DIM1],          /* Read buffer */
                    max,
                    i, j;


    /*
     * Initialize data.
     */
    for (i=0; i<DIM0; i++)
        for (j=0; j<DIM1; j++)
            wdata[i][j] = i * j - j;

    /*
     * Create a new file using the default properties.
     */
    file = H5Fcreate (FILE, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

    /*
     * Create dataspace.  Setting maximum size to NULL sets the maximum
     * size to be the current size.
     */
    space = H5Screate_simple (2, dims, NULL);

    /*
     * Create the dataset creation property list, add the gzip
     * compression filter and set the chunk size.
```

```
     */
    dcpl = H5Pcreate (H5P_DATASET_CREATE);
    status = H5Pset_deflate (dcpl, 9);
    status = H5Pset_chunk (dcpl, 2, chunk);

    /*
     * Create the dataset.
     */
    dset = H5Dcreate (file, DATASET, H5T_STD_I32LE, space, H5P_DEFAULT, dcpl,
                H5P_DEFAULT);

    /*
     * Write the data to the dataset.
     */
    status = H5Dwrite (dset, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
                wdata[0]);

    /*
     * Close and release resources.
     */
    status = H5Pclose (dcpl);
    status = H5Dclose (dset);
    status = H5Sclose (space);
    status = H5Fclose (file);

    return 0;
}
```

4. If a Compression Filter Was Not Effective

There is no "one size fits all" compression filter solution. Users have to consider a number of characteristics such as the type of data, the desired compression ratio, the encoding/decoding speed, the general availability of a compression filter, and licensing among other issues before committing to a compression filter. This is especially true for data producers. The way data is written will affect how much bandwidth consumers will need to download data products, how much system memory and time will be required to read the data, and how many data products can be stored on the users' system to name a few issues. Users should plan on experimenting with various compression filters and settings. The following are some suggestions for where to start finding the best compression filter for your data:

1. Find the compression filter which operates best on the type of data in the file and for the objectives of the file users. Different applications, data providers, and data consumers will work differently with different compression filters. For example, `Szip` compression is fast but typically achieves smaller compression ratios for floating point data than `gzip`. For more information on `Szip`, see the "Szip Compression in HDF Products" page.
2. Once you have the right compression method, find the right parameters. For example, `gzip` compression at level 6 usually achieves a compression ratio comparable to level 9 in less time. For more information on compression levels, see the `H5Pset_deflate` entry on the "H5P: Property List Interface" page of the *HDF5 Reference Manual*.
3. Data preprocessing using a filter such as `shuffling` in combination with compression can drastically improve the compression ratio. See the "Compression Comparisons" section below for more information.

The `h5repack` tool can be used to experiment with the data to address the items above.

Users should also look beyond compression. An HDF5 file may contain a substantial amount of unused space. The `h5stat` tool can be used to determine if space is used efficiently in an HDF5 file, and the `h5repack` tool can be used to reduce the amount of unused space in an HDF5 file. See "An Alternative to Compression" on page 20 for more information.

4.1. Compression Comparisons

An extensive comparison of different compression filters is outside the scope of this document. However, it is easy to show that, unless a suitable compression method or an advantageous filter combination is chosen, applying the same compression filter to different types of data may not reduce HDF5 file size as much as possible.

For example, we looked at a NASA weather data product file packaged with it its geolocation information (the file name is `GCRIO-REDRO_npp_d20030125_t0702533_e0711257_b00993_c20140501163427060570_XXXX_XXX.h5`) and used `h5repack` to apply three different compressions to the original file:

1. `gzip` with compression level 7
2. `Szip` compression using `NN` mode and 32-bit block size
3. `Shuffle` in combination with `gzip` compression level 7

Then we compared the sizes of the 32-bit floating dataset `/All_Data/CrIMSS-EDR-GEO-TC_All/Height` when different types of compression were used and compared for the sizes of the 32-bit integer dataset `/All_Data/CrIMSS-EDR_All/FORnum`. The results are shown in the table below.

Data	**Original**	**`gzip` Level 7**	**`Szip` Using NN Mode and Block-size 32**	**`Shuffle` and `gzip` Level 7**
32-bit Floats	1	2.087	1.628	2.56
32-bit Integers	1	3.642	10.832	38.20

Table 1: Compression ratio for different types of compressions when using `h5repack`

The combination of the `shuffle` filter and `gzip` compression level 7 worked well on both floating point and integer datasets, as shown in the fifth column of the table above. `gzip` compression worked better than `Szip` on the floating point dataset, but not on the integer dataset as shown by the results in columns three and four. Clearly, if the objective is to minimize the size of the file, datasets with different types of data have to be compressed with different compression methods.

For more information on the `shuffle` filter, see the "Data Pipeline Filters" section (5.4.2) in the "HDF5 Datasets" chapter of the *HDF5 User's Guide*. See also the "H5P: Property List Interface" page of the *HDF5 Reference Manual* for the `H5Pset_shuffle` function call entry.

4.2. An Alternative to Compression

Sometimes HDF5 files contain unused space. The `h5repack` command-line tool can be used to reduce the amount of unused space in a file without changing any storage parameters of the data. For example, running `h5stat` on the file `GCRIO-REDRO_npp_d20030125_t0702533_e0711257_b00993_c20140501163524579819_XXXX_XXX.h5` shows:

```
Summary of file space information:
  File metadata: 425632 bytes
  Raw data: 328202 bytes
  Unaccounted space: 449322 bytes
Total space: 1203156 bytes
```

After running `h5repack`, the file shows a 10-fold reduction in unaccounted space:

```
Summary of file space information:
  File metadata: 425176 bytes
  Raw data: 328202 bytes
  Unaccounted space: 45846 bytes
Total space: 799224 bytes
```

There is also a small reduction in file metadata space.

For more information on `h5repack` and `h5stat`, see the "HDF5 Tools" page in the *HDF5 Reference Manual*.

5. Other Resources

See the following documents published by The HDF Group for more information.

- See the "Creating a Compressed Dataset" tutorial on The HDF Group website.
- The "Filter Behavior in HDF5" sub-section is part of the `H5Pset_filter` function call entry. See the "H5P: Property List Interface" page of the *HDF5 Reference Manual*.

6. Revision History

April 24, 2015: *Initial published version.*