RFC: SZIP in HDF5: Dynamic Discovery of Encoder

October 12, 2004

Robert E. McGrath

1. Introduction

With SZIP version 1.2, the SZIP library can compiled with or without the encoder. The version without the encoder is license free, while the encoder requires a license for commercial use. The library has a new function, *SZ_encoder_enabled(),* which returns 0 if the encoder is disabled, and 1 if enabled.[1]

In HDF5 version 1.6.3, the HDF software implemented static linking to SZIP: the HDF library is configured at compile time without SZIP, or with SZIP decode only, or with SZIP decode and encode. Statically compiling the encoder has the undesirable effect that we must build two versions of HDF, compiled with and without SZIP encoding.

Static detection of the SZIP encoder is not, in fact, necessary, since the HDF5 library and SZIP are designed to dynamically detect the encoder at run time. The HDF5 library should simply ask the SZIP library if encoding is available in the current run.

This note describes the changes that must be made to HDF5 to support run-time detection of the SZIP encoder. These changes are relatively minor, do not change the compilation, testing, or use of the HDF5 library, and are not visible to user software.

These changes will be implemented as soon a possible in HDF5-1.6.4 and the HDF5-1.7 branch. (The changes are the same in both branches.)

2. Summary of Changes

The fundamental change is to completely eliminate the compile time constant 'H5_SZIP_CAN_ENCODE' (in H5pubconf.h). This constant is replaced by calls to *H5Zget_filter_info()* when needed. Note that all changes apply to code that is only relevant when SZIP is present, so all changes are conditionally compiled inside the '#ifdef H5_HAVE_FILTER_SZIP'.

Table 1 lists the files affected by this change. (autogenerated files in parens). The details of the changes are given in the follow sections.

---

[1] In the officially released SZIP library, when *SZ_encoder_enabled()* reports the encoder is disabled, the code is legally license free.

Table 1. Files Affected

(./src/H5config.h)
(./src/H5pubconf.h)
./src/H5config.h.in
./src/H5Z.c
./src/H5Zszip.c
./test/dsets.c
./test/tmisc.c
./tools/lib/h5tools_filters.c
./tools/h5dump/h5dumpgentest.c
./tools/h5repack/testh5repack_main.c
./tools/h5repack/testh5repack_make.c

2.1. Eliminate the "H5_SZIP_CAN_ENCODE" Variable

In the current implementation, the *configure* step detects the presence of the SZIP library and the presence of the decoder. The latter is recorded in *H5pubconf.h* by defining or not defining the variable, **H5_SZIP_CAN_ENCODE**.

This variable is no longer needed, and will be removed from *H5pubconf.h.in* and all places it appears in the HDF5 source.

2.2. Detect the Encoder Before Registering the SZIP Filter

The only change to the HDF5 library is in the registration of the SZIP encoder (when present). Instead of statically compiling in the configuration, the HDF5 library will probe the SZIP library, to discover the status of the encoder. Once registered, the filter works completely as before.

Each filter has a *H5Z_class_t* structure which has information fields indicating whether encoding and/or decoding are enabled (Figure 1). In the current code, this structure is statically defined in *H5Zszip.c*, with an "ifdef H5_SZIP_CAN_ENCODE" to determine whether the encoder is enabled or not. The declaration of the structure will be changed to statically define a default structure (Figure 2).

```
   H5Z_class_t H5Z_SZIP[1] = {{
     H5Z_CLASS_T_VERS,      /* H5Z_class_t version */
     H5Z_FILTER_SZIP,              /* Filter id number                    */
#ifdef H5_SZIP_CAN_ENCODE
     1,             /* Encoder present */
#else
     0,             /* Encoder disabled */
#endif
      1,               /* decoder_present flag (set to true) */
     "szip",                                  /* Filter name for debugging     */
     H5Z_can_apply_szip,             /* The "can apply" callback     */
   …
```

Figure 1

```
H5Z_class_t H5Z_SZIP[1] = {{
    H5Z_CLASS_T_VERS,       /* H5Z_class_t version */
    H5Z_FILTER_SZIP,        /* Filter id number                    */
    1,          /* Assume encoder present: check before registering */
    1,          /* decoder_present flag (set to true) */
    "szip",                 /* Filter name for debugging           */
    H5Z_can_apply_szip,     /* The "can apply" callback            */
  …
```

Figure 2

When the library starts up, the filters are registered with calls to *H5Zregister()*. If the SZIP filter is present it is registered with a call to *H5Zregister()*, passing the *H5Z_class_t* structure discussed above (Figure 3). In order to set the correct value for this structure, *SZ_encoder_enabled()* will be called (this subroutine is in the SZIP library), and the value of *H5Z_class_t->encoder_present* set accordingly. Then the filter will be registered, as before. (Figure 4)

```
#ifdef H5_HAVE_FILTER_SZIP
    if (H5Z_register (H5Z_SZIP)<0)
        HGOTO_ERROR (H5E_PLINE, H5E_CANTINIT, FAIL, "unable to register szip filter")
#endif /* H5_HAVE_FILTER_SZIP */
```

Figure 3

```
#ifdef H5_HAVE_FILTER_SZIP
    H5Z_SZIP->encoder_present = SZ_encoder_enabled();
    if (H5Z_register (H5Z_SZIP)<0)
        HGOTO_ERROR (H5E_PLINE, H5E_CANTINIT, FAIL, "unable to register szip filter")
#endif /* H5_HAVE_FILTER_SZIP */
```

Figure 4

After this point, the filter has been registered and is used exactly as before. All other parts of the library will retrieve the information from this field through a call to the API function, *H5Zget_filter_info(H5Z_FIZTER_SZIP).*

No other changes to the library are required.

2.3. Library Tests

Library tests involving SZIP are contingent on whether the encoder is enabled or not. In the current implementation, alternative versions of the test are controlled by the H5_SZIP_CAN_ENCODE variable. This will be changed to check the filter configuration at run time.

Figure 5 shows a subroutine to check the status of the SZIP encoder, and return 1 if encoder is enabled.[2] In this subroutine, *H5Zget_filter_info()* is called, and the value of *filter_config_flags* checked. This flag is set from the *H5Z_class_t* structure discussed above.

Essentially, this subroutine should be called wherever **H5_SZIP_CAN_ENCODE** appears in the test code.

```
#if defined H5_HAVE_FILTER_SZIP

int szip_can_encode( )
{

 herr_t       status;
 unsigned int filter_config_flags;

   status =H5Zget_filter_info(H5Z_FILTER_SZIP, & filter_config_flags);
   if ((filter_config_flags &
         (H5Z_FILTER_CONFIG_ENCODE_ENABLED|H5Z_FILTER_CONFIG_DECODE_ENABLED))
             == 0) {
    /* filter present but neither encode nor decode is supported??? */
    return -1;
   } else if ((filter_config_flags &
         (H5Z_FILTER_CONFIG_ENCODE_ENABLED|H5Z_FILTER_CONFIG_DECODE_ENABLED))
           ==
           H5Z_FILTER_CONFIG_DECODE_ENABLED) {
     /* decoder only: read but not write */
    return 0;
   } else if ((filter_config_flags &
         (H5Z_FILTER_CONFIG_ENCODE_ENABLED|H5Z_FILTER_CONFIG_DECODE_ENABLED))
           ==
         H5Z_FILTER_CONFIG_ENCODE_ENABLED) {
     /* encoder only: read but not write ??? */
     return -1;
   } else if ((filter_config_flags &
         (H5Z_FILTER_CONFIG_ENCODE_ENABLED|H5Z_FILTER_CONFIG_DECODE_ENABLED))
           ==
         (H5Z_FILTER_CONFIG_ENCODE_ENABLED|H5Z_FILTER_CONFIG_DECODE_ENABLED))
     {
    return 1;
   }
}
```

Figure 5

There are two library test files affected, *test/dsets.c* and *test/tmisc.c*. The changes to *tmisc.c* are simple: the two SZIP related tests are changed to check if the encoder is present, and return silently if not. (Figure 6)

---

[2] This subroutine specifically checks SZIP. It is possible to make this a generic function that would work for any filter. See section 2.4 below for an example of a more general test.

```
#ifdef  H5_SZIP_CAN_ENCODE
static void
test_misc21(void) {
 /* … */
    if (szip_can_encode() != 1) return;

     /* Output message about test being performed */
     MESSAGE(5, ("Testing late allocation time w/chunks & filters\n"));
     … /* rest of test is unchanged */
```

Figure 6. A new test at the first statement in *test_misc21()* (and similarly *testmisc_22()*).

The changes to the tests in *dsets.c* are more complex: depending on whether SZIP encoding is enabled, certain tests are performed and others are skipped. The current code controls this contingency by conditionally compiling alternative code depending on "ifdef H5_SZIP_CAN_ENCODE". This is changed to call *szip_can_encode()* instead.

Figure 7 shows an example of the code using the ifdef H5_SZIP_CAN_ENCODE. This is revised to call *szip_can_encode()* instead. Figure 8 shows the same code with the dynamic detection.

```
#ifdef H5_HAVE_FILTER_SZIP
   TESTING("szip filter (with encoder)");
   if((dc = H5Pcreate(H5P_DATASET_CREATE))<0) goto error;
   if (H5Pset_chunk (dc, 2, chunk_size)<0) goto error;
#ifdef H5_SZIP_CAN_ENCODE
           puts("");
           if (H5Pset_szip(dc, szip_options_mask, szip_pixels_per_block)<0) goto error;
           if(test_filter_internal(file,DSET_SZIP_NAME,dc,DISABLE_FLETCHER32,
                 DATA_NOT_CORRUPTED,  &szip_size)<0) goto error;
#else
           SKIPPED();
#endif
           TESTING("szip filter (without encoder)");
#ifndef H5_SZIP_CAN_ENCODE
           puts("");
           if(test_filter_noencoder(NOENCODER_SZIP_DATASET) < 0) goto error;
#else
           SKIPPED();
#endif
   if (H5Pclose (dc)<0) goto error;
#else /* H5_HAVE_FILTER_SZIP */
   TESTING("szip filter");
   SKIPPED();
   puts("    Szip filter not enabled");
#endif /* H5_HAVE_FILTER_SZIP */
```

Figure 7

```
#ifdef H5_HAVE_FILTER_SZIP
   TESTING("szip filter (with encoder)");
   szip_can_encode = szip_encoder_enabled();
   if (szip_can_encode == 1) {
      if((dc = H5Pcreate(H5P_DATASET_CREATE))<0) goto error;
      if (H5Pset_chunk (dc, 2, chunk_size)<0) goto error;
           puts("");
           if (H5Pset_szip(dc, szip_options_mask, szip_pixels_per_block)<0) goto error;
           if(test_filter_internal(file,DSET_SZIP_NAME,dc,DISABLE_FLETCHER32,
                DATA_NOT_CORRUPTED,  &szip_size)<0) goto error;
      if (H5Pclose (dc)<0) goto error;
   } else {
           SKIPPED();
   }
   TESTING("szip filter (without encoder)");
   if (szip_can_encode != 1) {
           puts("");
           if(test_filter_noencoder(NOENCODER_SZIP_DATASET) < 0) goto error;
   } else {
           SKIPPED();
   }
#else /* H5_HAVE_FILTER_SZIP */
   TESTING("szip filter");
   SKIPPED();
   puts("    Szip filter not enabled");
#endif /* H5_HAVE_FILTER_SZIP */
```

Figure 8

Note that these tests behave exactly as before (indeed, they would work fine with static detection.) The main change is that there is one test that works correctly for either encoder enabled or disabled.

2.4. Tools Tests

Several *h5dump* and *h5repack* tests involve SZIP, and therefore are contingent on whether the encoder is enabled or not. This dependency occurs where test files are generated (when it is necessary to encode the test data using SZIP), and in the case of *h5repack*, when the output would require writing the data with SZIP.

These tests need to be changed along the same lines as already described. Instead of conditionally compiling the code, the encoder should be detected at run time.

The tools library has a function called *h5tools_canreadf()* (in *tools/lib/h5tools_filter.c*), which checks the availability of filters. A new function will be added to this file, *h5tools_canwritef()*. This function checks the availability of encoding for a given filter (not just SZIP). Figure 9 shows this subroutine.

```
int h5tools_can_encode( H5Z_filter_t filtn)
{
 int         have_deflate=0; /* assume initially we do not have filters */
 int         have_szip=0;
 int         have_shuffle=0;
 int         have_fletcher=0;
 herr_t      status;
 unsigned int filter_config_flags;

#ifdef H5_HAVE_FILTER_DEFLATE
 have_deflate=1;
#endif
#ifdef H5_HAVE_FILTER_SZIP
 have_szip=1;
#endif
/* … */
  switch (filtn)
  {
    /*  user defined or unrecognized filter   */
  default:
   return 0;
  break;
  case H5Z_FILTER_DEFLATE:
  if (!have_deflate)
  {    return 0;  }
  break;
  case H5Z_FILTER_SZIP:
  if (!have_szip)
  {   return 0;
  } else {
    status =H5Zget_filter_info(filtn, &filter_config_flags);
    if ((filter_config_flags &
        (H5Z_FILTER_CONFIG_ENCODE_ENABLED|H5Z_FILTER_CONFIG_DECODE_ENABLED))
             == 0) {
         /* filter present but neither encode nor decode is supported??? */
      return -1;
  } else if ((filter_config_flags &
        (H5Z_FILTER_CONFIG_ENCODE_ENABLED|H5Z_FILTER_CONFIG_DECODE_ENABLED))
           ==   H5Z_FILTER_CONFIG_DECODE_ENABLED) {
   /* decoder only: read but not write */
   return 0;
  } else if ((filter_config_flags &
        (H5Z_FILTER_CONFIG_ENCODE_ENABLED|H5Z_FILTER_CONFIG_DECODE_ENABLED))
              ==  H5Z_FILTER_CONFIG_ENCODE_ENABLED) {
       /* encoder only: read but not write ??? */
   return -1;
  } else if ((filter_config_flags &
        (H5Z_FILTER_CONFIG_ENCODE_ENABLED|H5Z_FILTER_CONFIG_DECODE_ENABLED))
         ==
        (H5Z_FILTER_CONFIG_ENCODE_ENABLED|H5Z_FILTER_CONFIG_DECODE_ENABLED))
    {
      return 1;
    }
  break;
 /* … */
```

Figure 9. Sketch of *h5tools_canwritef()*. GZIP and SZIP filters shown, other filters omitted for space.

This function is called in the test programs, replacing all references to H5_SZIP_CAN_ENCODE. There are several places in *tools/h5dump/h5dumpgentest.c*, *tools/h5repack/testh5repack_main.c*, and *tools/h5repack/testh5repack_make.c*.

Figure 10 shows an example from *testh5repack_make.c*. The current code is conditional on the test "#if defined (H5_HAVE_FILTER_SZIP) && defined (H5_SZIP_CAN_ENCODE)". Inside this test, a dataset is created using SZIP. This code is changed to call *h5tools_can_encode()*. (Figure 11) Note that the test is still conditional on the presence of the SZIP filter, but detects whether the encoder can be used when the test is run (rather than when compiled).

```
#if defined (H5_HAVE_FILTER_SZIP) && defined (H5_SZIP_CAN_ENCODE)
 /* remove the filters from the dcpl */
 ret=H5Premove_filter(dcpl,H5Z_FILTER_ALL);
 assert(ret>=0);

 /* set szip data */
 ret=H5Pset_szip (dcpl,szip_options_mask,szip_pixels_per_block);
 assert(ret>=0);

 ret=make_dset(fid,"szip",sid,H5T_NATIVE_INT,dcpl,buf1);
 assert(ret>=0);
#endif
```

Figure 10

```
#if defined (H5_HAVE_FILTER_SZIP)
if (h5tools_can_encode(H5Z_FILTER_SZIP) == 1) {
 encoder_enabled=1;
 /* remove the filters from the dcpl */
 ret=H5Premove_filter(dcpl,H5Z_FILTER_ALL);
 assert(ret>=0);

 /* set szip data */
 ret=H5Pset_szip (dcpl,szip_options_mask,szip_pixels_per_block);
 assert(ret>=0);

 ret=make_dset(fid,"szip",sid,H5T_NATIVE_INT,dcpl,buf1);
 assert(ret>=0);
}
#endif
```

Figure 11

Again, these tests work as before. The only difference is that the code is the same whether encoding is enabled or not.

3. Overall Behavior of the Revised Library

3.1. Compiling and Testing the Library

There is no change in configuring, compiling, or testing the library.

3.2. Deploying the Library

With this change, the HDF5 library can be built with SZIP, with or without the encoder. The same HDF5 binary can be deployed with SZIP with or without the encoder, no matter which was used for compiling. When using shared libraries, the behavior of the library will automatically detect whether the encoder is available at run time.

To illustrate how this works, consider the example program in Figure 12. When SZIP encoding is enabled, the file and dataset are created. When SZIP encoding is disabled, the program detects this and exits with a message. (Note that the call to *H5Dcreate()* would fail as well, when encoding is not enabled.)

Using dynamic linking (e.g., on Linux), this program is compiled once, using HDF5 and SZIP 1.2. The program can be *compiled* with *either* version of SZIP. When the program runs, it dynamically links to HDF5 and to libsz. The result of the program depends on the libsz that is linked at run time, not what was compiled.

To illustrate this process, consider the following configuration on a Linux system.

```
/usr/test/lib/libhdf5.so                    # the hdf library
/usr/test/szip_with_encoder/lib/libsz.so    #  the same vers. of szip, w and wo encoder
/usr/test/szip_wo_encoder/lib/libsz.so
```

The *try_szip* program from Figure 12 is compiled using either version of SZIP. The program would usually be compiled using ‘h5cc’. E.g.,

h5cc try_szip.c –o try_szip

After it is compiled, the dynamic library path is set to:

setenv LD_LIBRARY_PATH “/usr/test/lib/:/usr/test/szip_with_encoder/lib”

With this librry path, run the program: *try_szip*. The program succeeds, the file is created and the dataset created.

Reset the dynamic library path to:

setenv LD_LIBRARY_PATH “/usr/test/lib/:/usr/test/szip_wo_encoder/lib”

```
#include "hdf5.h"

#define H5FILE_NAME    "tryszip.h5"
#define RANK    2

int main(void)
{
    hid_t    file;
    hid_t    dataset, dataspace;
    hid_t    plist;

    herr_t   status;
    hsize_t  dims[2];
    hsize_t  cdims[2];
    unsigned int filter_config_flags;


   status =H5Zget_filter_info(H5Z_FILTER_SZIP, &filter_config_flags);
   if ((filter_config_flags &
          H5Z_FILTER_CONFIG_ENCODE_ENABLED) == 0) {
     printf("Szip: encoding disabled, exiting\n");
    return 1;
   }

    file = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT,
H5P_DEFAULT);

    dims[0] = 1000;
    dims[1] = 20;
    cdims[0] = 20;
    cdims[1] = 20;
    dataspace = H5Screate_simple(RANK, dims, NULL);
    plist     = H5Pcreate(H5P_DATASET_CREATE);
                H5Pset_chunk(plist, 2, cdims);
                H5Pset_szip( plist,H5_SZIP_NN_OPTION_MASK , 32 );
    dataset = H5Dcreate(file, "/Compressed_Data", H5T_NATIVE_INT,
                        dataspace, plist);

    if (dataset < 0) {
      printf("dataset create failed?\n");
        H5Sclose(dataspace);
        H5Dclose(dataset);
        H5Pclose(plist);
        H5Fclose(file);
        return 1;
    } else {
      printf("dataset created OK\n");
      H5Sclose(dataspace);
        H5Dclose(dataset);
        H5Pclose(plist);
        H5Fclose(file);
        return 0;
    }
}
```

Figure 12. An example program, try_szip. The program dynamically detects whether the encoder is enabled.

Run the same binary program with this path: *try_szip*. In this case, the program detects the encoder is disabled, prints the message and exits.

Thus, a single binary does the right thing, depending on the system configuration. This is the desired result.

3.3. Distribution of Pre-built Binaries

With this implementation, we can distribute one copy of HDF5, and two copies of the SZIP library. The user can download HDF5, and select which version of SZIP to use. In fact, the user can install *both* versions of SZIP, and restrict access to the licensed version.

This also lets tools builders, such as IDL and Matlab, ship their product with a single copy of HDF5, and include whichever version of SZIP (or both) they require.

4. Notes and Other Changes

4.1. Dynamic Libraries for SZIP

The changes described here depend on the completion of the automake system for SZIP. The current release of SZIP (1.2) does not support dynamic libraries, but will do so in the near future.

4.2. Consolidating Detection Subroutines

In this approach, the library tests and tools test repeat similar code to detect the presence of the encoder. It would be nice to have a single copy of this code somewhere, e.g., a test library. This is left to the future.

4.3. Windows Test Scripts

The windows tests use H5_SZIP_CAN_ENCODE. This needs to be changed along the lines described here. The details of these changes are TBD.

4.4. Changes to Utilities and Tools

The *h5repack* and *h4toh5* tools should be modified to dynamically detect the SZIP encoder, and give the user clear diagnostic messages in the case where SZIP is available but cannot encode.

The HDFView tools has partially implemented this feature, but may need further changes.

4.5. HDF4 Support

Unfortunately, this feature cannot be added to HDF4 unless and until HDF4 supports dynamic libraries. If dynamic libraries are implemented for HDF4, then similar changes to dynamically detect SZIP encoder can be implemented.

Related Links

1. “Szip Compression in HDF Products”, http://hdf.ncsa.uiuc.edu/doc_resource/SZIP/

2. “HDF5: API Specification Reference Manual”,
http://hdf.ncsa.uiuc.edu/HDF5/doc/RM_H5Front.html

3. “SZIP Support-- Proposals for Handling "Read Only" Libraries”,
http://hdf.ncsa.uiuc.edu/RFC/SZIP/Szip_support.html