HDF5 Dynamically Loaded Filters

The HDF Group

This document describes the HDF5 dynamically loaded filter feature (HDF5 filter plugin) that will be released in the HDF5 version 1.8.11 in May 2013.

This feature allows an HDF5 application to apply a non-native HDF5 filter during I/O operations at runtime. The application does not need to register the filter with the HDF5 Library. There is no need for adding the filter to the HDF5 Library and then linking the application with the modified library.

For example, the HDF5 1.8.11 command-line tools such as h5dump or h5ls downloaded from The HDF Group website, or Java-based browsing tool HDFView that uses 1.8.11 will be able to read LZF or bzip2 compressed datasets created by H5Py or PyTables software.

The dynamically loaded filter feature was sponsored by ***Deutsches Elektronen-Synchrotron (DESY)***, Hamburg, Germany.

**Contents**

1 Introduction 3

2 Requirements 5

2.1 Functional Requirements 5

2.2 Non-functional Requirements 5

3 Programming Model for Applications 6

3.1 Applying a Third-party Filter When Creating and Writing a Dataset 6

3.2 Reading Data with an Applied Third-party Filter 6

3.3 A Word of Caution When Using Custom Filters 8

4 Programming Model for HDF5 Filter Plugins 10

4.1 Writing a Filter Function 10

4.2 Registering a Filter with The HDF Group 10

4.3 Creating an HDF5 Filter Plugin 11

4.4 Installing an HDF5 Filter Plugin 12

5 Design 13

5.1 Data Writing 13

5.2 Data Reading 14

5.3 New HDF5 Source Files for Dynamically Loaded Filters 15

5.4 Testing Dynamically Loaded Filters 15

6 Proposed Changes to the HDF5 Command-line Tools 16

6.1 h5dump, h5ls, and h5copy 16

6.2 h5repack 16

7 Building an HDF5 bzip2 Plugin Example 17

8 Example: Writing and Reading Compressed Data 18

9 HDF5 bzip2 Filter Plugin 22

10 Example of Makefile 26

11 Revision History 27

12 References 28

# Introduction

The HDF5 Library provides an internal mechanism to compress an HDF5 dataset’s raw data and the links stored in an HDF5 group object. Readers who are unfamiliar with HDF5 compression are encouraged to view the HDF5 tutorials, the *HDF5 User’s Guide*, the *HDF5 Reference Manual* and other documentation available from the <http://www.hdfgroup.org> website.

The HDF5 compression mechanism is a part of the HDF5 “filter pipeline” that modifies an application’s data during the I/O operations. The pipeline was designed to be extensible. New filters could be easily added to the pipeline by an application, or the HDF5 Library code can be modified with a new filter to be later used by an application.

The HDF5 Library supports several “internal” filters (see in the *HDF5 User’s Guide*, section 5.4.2, “Data Pipeline Filters”). Two filters, *deflate* and *szip*, depend on third-party compression libraries. All internal filters are configurable. They can be added or removed during the configuration step when the HDF5 Library is built.

While currently available HDF5 “internal” compression methods work reasonably well on users’ datasets, there are certain drawbacks to the current implementation. First, the “internal” compression methods may not provide the optimal compression ratio, as do some newly developed or specialized compression methods. Secondly, if a data provider wants to use a “non-internal” compression for storing the data in HDF5, he/she has to write a filter function that uses the new compression method and then register it with the library. Data consumers of such HDF5 files will need to have the new filter function and use it with their applications to read the data, or they will need a modified version of the HDF5 Library that has the new filter as a part of the library.

If a user of such data does not have a modified HDF5 Library installed on his system, command-line tools such as h5dump or h5ls will not be able to display the compressed data. Furthermore, it would be practically impossible to determine the compression method used, making the data stored in HDF5 useless.

It is clear that the current HDF5 filter mechanism, while extensible, does not work well with third-party filters. It would be a maintenance nightmare to keep adding and supporting new compression methods in HDF5. For any set of HDF5 “internal” filters, there always will be data with which the “internal” filters will not achieve the optimal performance needed to address data I/O and storage problems. Thus the current HDF5 filter mechanism should be enhanced to address the issues discussed above.

We have added a new feature to HDF5 called “dynamically loaded filters in HDF5.” This feature will make the HDF5 third-party filters available to an application at runtime. The third-party HDF5 filter function has to be a part of the HDF5 filter plugin installed on the system as a shared library or DLL.

To use a third-party filter an HDF5 application should call the H5Pset\_filter function when setting the filter pipeline for a dataset creation property. The HDF5 Library will register the filter with the library and the filter will be applied when data is written to the file.

When an application reads data compressed with a third-party HDF5 filter, the HDF5 Library will search for the required filter plugin, register the filter with the library (if the filter function is not registered) and apply it to the data on the read operation.

This document describes this new feature and provides details in the following sections.

# Requirements

The current design for the HDF5 dynamically loaded filters feature addresses the requirements listed in the sections below.

## Functional Requirements

1. There are **no changes to the HDF5 file format**.
2. There are **no changes** **to the HDF5 interface and the programming model** (see the “Programming Model for Applications” section on page 6) in order to use the feature.
3. The HDF5 third-party filters are available as **shared libraries or DLLs** on the user’s system. The minimum content of the shared library satisfies the requirements as documented in the “Programming Model for HDF5 Filter Plugins” section on page 10.
4. There are predefined **default locations** where the HDF5 Library searches the shared libraries or DLLs with the HDF5 filter functions.
5. The **default location may be overwritten** by an environment variable.
6. Once a filter plugin library is loaded, it stays loaded until the HDF5 Library is closed.

Future enhancements:

* The default search locations can be specified in a configuration file.

## Non-functional Requirements

1. The HDF Group maintains a list of the third-party filters registered with HDF5 to facilitate tracking of a filter provenance. The current procedure is outlined at <http://www.hdfgroup.org/services/contributions.html>.
2. The HDF group publishes a specification and provides examples of dynamic loaded filters libraries.

Future enhancements:

* The HDF Group provides a repository such as SVN or GIT for the third-party filters and outlines a testing procedure that can be used by the filters maintainers to test the filters with the HDF5 libraries under development.

# Programming Model for Applications

This section describes the programming model for an application that uses a third-party HDF5 filter plugin to write or read data. For simplicity of presentation, it is assumed that the HDF5 filter plugin is available on the system in a default location. The HDF5 filter plugin is discussed in detail in the “Programming Model for HDF5 Filter Plugins” section on page 10.

## Applying a Third-party Filter When Creating and Writing a Dataset

A third-party filter can be added to the HDF5 filter pipeline by using the H5Pset\_filter function, as a user would do in the past. The identification number and the filter parameters should be available to the application. For example, if the application intends to apply the HDF5 bzip2 compression filter that was registered with The HDF Group and has an identification number 307 (see <http://www.hdfgroup.org/services/contributions.html>) then the application would follow the steps as outlined below:

 dcpl = H5Pcreate (H5P\_DATASET\_CREATE);

 status = H5Pset\_filter (dcpl, (H5Z\_filter\_t)307, H5Z\_FLAG\_MANDATORY,

 (size\_t)6, cd\_values);

 dset = H5Dcreate (file, DATASET, H5T\_STD\_I32LE, space, H5P\_DEFAULT, dcpl,

 status = H5Dwrite (dset, H5T\_NATIVE\_INT, H5S\_ALL, H5S\_ALL, H5P\_DEFAULT,

 wdata[0]);

## Reading Data with an Applied Third-party Filter

An application does not need to do anything special to read the data with a third-party filter applied. For example, if one wants to read data written in 3.1 the following regular steps should be taken:

 file = H5Fopen (FILE, H5F\_ACC\_RDONLY, H5P\_DEFAULT);

 dset = H5Dopen (file, DATASET, H5P\_DEFAULT);

 H5Dread (dset, H5T\_NATIVE\_INT, H5S\_ALL, H5S\_ALL, H5P\_DEFAULT,

 rdata[0]);

The 1.8.11 version of the command-line utility h5dump, for example, will read and display the data as shown:

[epourmal@jam example]$ ./h5dump -p \*.h5

HDF5 "h5ex\_d\_bzip2.h5" {

GROUP "/" {

 DATASET "DS1" {

 DATATYPE H5T\_STD\_I32LE

 DATASPACE SIMPLE { ( 32, 64 ) / ( 32, 64 ) }

 STORAGE\_LAYOUT {

 CHUNKED ( 4, 8 )

 SIZE 6410 (1.278:1 COMPRESSION)

 }

 FILTERS {

 USER\_DEFINED\_FILTER {

 FILTER\_ID 307

 COMMENT bzip2

 PARAMS { 6 }

 }

 }

 …

 }

 DATA {

 (0,0): 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11, -12, -13, -14,

 (0,15): -15, -16, -17, -18, -19, -20, -21, -22, -23, -24, -25, -26,

 (0,27): -27, -28, -29, -30, -31, -32, -33, -34, -35, -36, -37, -38,

 (0,39): -39, -40, -41, -42, -43, -44, -45, -46, -47, -48, -49, -50,

 (0,51): -51, -52, -53, -54, -55, -56, -57, -58, -59, -60, -61, -62,

 (0,63): -63,

……

Compare the output above with the output of the 1.8.10 version of h5dump shown below:

[epourmal@jam example]$ h5dump -p \*.h5

HDF5 "h5ex\_d\_bzip2.h5" {

GROUP "/" {

 DATASET "DS1" {

 DATATYPE H5T\_STD\_I32LE

 DATASPACE SIMPLE { ( 32, 64 ) / ( 32, 64 ) }

 STORAGE\_LAYOUT {

 CHUNKED ( 4, 8 )

 SIZE 6410

 }

 FILTERS {

 UNKNOWN\_FILTER {

 FILTER\_ID 307

 COMMENT HDF5 bzip2 filter; see http://www.hdfgroup.org/services/contributions.html

 PARAMS { 6 }

 }

 }

 …

 }

 **DATA {h5dump error: unable to print data**

 }

 }

}

}

The h5dump utility version 1.8.10 cannot read data compressed with bzip2 because the filter is not available to the HDF5 library.

The complete example of writing and reading data with an applied third-party HDF5 filter is shown in the “Example: Writing and Reading Compressed Data” section on page 18. Please notice that the application DOES NOT need to link with the HDF5 plugin library.

## A Word of Caution When Using Custom Filters

Data goes through the HDF5 filter pipeline only when it is written to the file or read into application memory space from the file. For example, the I/O operation is triggered with a call to H5Fflush, or when a data item (HDF5 metadata or a raw data chunk) is evicted from the cache or brought into the cache. Please notice that H5Dread/write calls on the chunked datasets do not necessarily trigger I/O since the HDF5 Library uses a separate chunk cache.

A data item may remain in the cache until the HDF5 Library is closed. If the HDF5 plugin that has to be applied to the data item becomes unavailable before the file and all objects in the file are closed, an error will occur. The following example demonstrates the issue. Please notice the position of the H5Zunregister call:

 /\*

 \* Create a new group using compression.

 \*/

 gcpl = H5Pcreate (H5P\_GROUP\_CREATE);

 status = H5Pset\_filter(gcpl,H5Z\_FILTER\_BZIP2,H5Z\_FLAG\_MANDATORY,(size\_t)1, cd\_values);

 group = H5Gcreate (file, GNAME, H5P\_DEFAULT, gcpl, H5P\_DEFAULT);

 for (i=0; i < NGROUPS; i++) {

 sprintf(name, "group\_%d", i);

 tmp\_id = H5Gcreate (group, name, H5P\_DEFAULT, H5P\_DEFAULT, H5P\_DEFAULT);

 status = H5Gclose(tmp\_id);

 }

 status = H5Pclose (gcpl);

 status = H5Gclose (group);

 /\*

 \* Unregister the filter. Call to H5Fclose will fail because the library tries

 \* to apply the filter that is not available anymore. This has a cascade effect

 \* on H5Fclose.

 \*/

 H5Zunregister(H5Z\_FILTER\_BZIP2);

 status = H5Fclose (file);

Here is an error stack produced by the program:

HDF5-DIAG: Error detected in HDF5 (1.9.149) thread 0:

 #000: H5F.c line 2060 in H5Fclose(): decrementing file ID failed

 major: Object atom

 minor: Unable to close file

 #001: H5I.c line 1406 in H5I\_dec\_app\_ref(): can't decrement ID ref count

 major: Object atom

 minor: Unable to decrement reference count

 #002: H5F.c line 1837 in H5F\_close(): can't close file

 major: File accessibility

 minor: **Unable to close file**

……….

 #026: H5Z.c line 1045 in H5Z\_find(): **required filter is not registered**

 major: Data filters

 minor: Object not found

To avoid the problem make sure to close all objects to which the filter is applied and flush them using the H5Fflush call before unregistering the filter.

# Programming Model for HDF5 Filter Plugins

This section describes how to create an HDF5 filter, an HDF5 filter plugin, and how to install the HDF5 plugin on the system.

## Writing a Filter Function

The HDF5 filter function for the dynamically loaded filter feature should be written as any custom filter described in <http://www.hdfgroup.org/HDF5/doc/H5.user/Filters.html>. See the “Example” section, section 5, of that document to get an idea of the simple filter function, and see the example of the more sophisticated HDF5 bzip2 filter function in the “Building an HDF5 bzip2 Plugin Example” section on page 17. The HDF5 bzip2 filter function is also available for download from <https://svn.hdfgroup.uiuc.edu/hdf5_plugins/trunk/BZIP2>.

The user has to remember a few things when writing an HDF5 filter function.

1. An HDF5 filter is bidirectional.

The filter handles both input and output to the file; a flag is passed to the filter to indicate the direction.

1. An HDF5 filter operates on a buffer.

The filter reads data from a buffer, performs some sort of transformation on the data, places the result in the same or new buffer, and returns the buffer pointer and size to the caller.

1. An HDF5 filter should return zero in the case of failure.

The signature of the HDF5 filter function and the accompanying filter structure (see the section below) are described in the *HDF5 Reference Manual* <http://www.hdfgroup.org/HDF5/doc/RM/RM_H5Z.html#Compression-Register>.

## Registering a Filter with The HDF Group

If you are writing a filter that will be used by others, it would be a good idea to request a filter identification number and register it with The HDF Group. Please follow the procedure described at <http://www.hdfgroup.org/services/contributions.html>.

The HDF Group anticipates that developers of HDF5 filter plugins will not only register new filters, but will also provide links to the source code and/or binaries for the corresponding HDF5 filter plugins.

It is very important for the users of the filter that developers provide filter information in the “name” field of the filter structure, for example:

const H5Z\_class2\_t H5Z\_BZIP2[1] = {

 H5Z\_CLASS\_T\_VERS, /\* H5Z\_class\_t version \*/

 (H5Z\_filter\_t) 307, /\* Filter id number \*/

 1, /\* encoder\_present flag (set to true) \*/

 1, /\* decoder\_present flag (set to true) \*/

 **"HDF5 bzip2 filter; see**

 **http://www.hdfgroup.org/services/contributions.html",**

/\* Filter info \*/

 NULL, /\* The "can apply" callback \*/

 NULL, /\* The "set local" callback \*/

 (H5Z\_func\_t) H5Z\_filter\_bzip2, /\* The filter function \*/

 };

The HDF5 Library and command-line tools have access to the “name” field. An application can use the H5Pget\_filter<\*> functions to retrieve information about the filters.

Using the example of the structure above, the h5dump tool will print the string “HDF5 bzip2 filter found at …” pointing users to the applied filter (see the example in the “Reading Data with an Applied Third-party Filter” section on page 6) thus solving the problem of the filter’s origin.

## Creating an HDF5 Filter Plugin

The HDF5 filter plugin source should include:

1. The H5PLextern.h header file from the HDF5 distribution.
2. The definition of the filter structure (see the example shown in the section above).
3. The filter function (for example, H5Z\_filter\_bzip2).
4. The two functions necessary for the HDF5 Library to find the correct type of the plugin library while loading it at runtime and to get information about the filter function:

H5PL\_type\_t H5PLget\_plugin\_type(void);

const void\* H5PLget\_plugin\_info(void);

Here is an example of the functions above for the HDF5 bzip2 filter:

H5PL\_type\_t H5PLget\_plugin\_type(void) {return H5PL\_TYPE\_FILTER;}

const void\* H5PLget\_plugin\_info(void) {return H5Z\_BZIP2;}

1. Other functions such as the source of the compression library may also be included.

Build the HDF5 filter plugin as a shared library. The following steps should be taken:

1. When compiling, point to the HDF5 header files.
2. Use the appropriate linking flags.
3. Link with any required external libraries.
4. For example, if libbz2.so is installed on a Linux system, the HDF5 bzip2 plugin library libH5Zbzip2.so may be linked with libbz2.so instead of including bzip2 source into the plugin library.

The complete example of the HDF5 bzip2 plugin library is provided at <http://svn.hdfgroup.uiuc.edu/hdf5_plugins/trunk/BZIP2/> and can be adopted for other plugins. A simple Makefile file for Linux systems is shown in the “Example of Makefile” section on page 26.

## Installing an HDF5 Filter Plugin

The default directory for an HDF5 filter plugin library is defined on UNIX-like systems as

“/usr/local/hdf5/lib/plugin”

and on Windows systems as

"%ALLUSERSPROFILE%/hdf5/lib/plugin".

The default path can be overwritten by a user with the HDF5\_PLUGIN\_PATH environment variable. Several directories can be specified for the search path using “:” as a path separator for UNIX-like systems and “;” for Windows.

Readers are encouraged to try the example in the “Building an HDF5 bzip2 Plugin Example” section on page 17.

# Design

Dynamic loading of the HDF5 filter plugin (or filter library) is triggered only by two events: when an application calls the H5Pset\_filter function to set the filter for the first time, or when the data to which the filter is applied is read for the first time.

## Data Writing

This section outlines the usage of the new feature on the write operation.

When the application enables a filter with H5Pset\_filter, the library will search for it by its filter identifier. The library will first search for the filter among its internal filters and custom filters registered with the library, then it will search in the list of filters that have been dynamically loaded. If it cannot find the filter, it will search for the filter library in the default path or the path indicated by the environment variable HDF5\_PLUGIN\_PATH.

When found, the HDF5 Library will load the filter library, register the filter, and add it to the filter pipeline. There is no need for the application to register the dynamic filter with H5Zregister, as this step is done internally in H5Pset\_filter. The H5PL\_load function in the H5Pocpl.c file in the src directory of the HDF5 Library distribution does the searching and loading of the filter library.

Once the filter library is loaded, it will stay loaded until the HDF5 Library is closed or until the filter is unregistered with the H5Zunregister call. The H5Zregister function is still required when a third-party filter is linked into an application and is not provided as a shared library.

If the dynamically loaded filter has any auxiliary data, such as the compression level for bzip2 filter, it can be passed in through the cd\_values parameter of H5Pset\_filter.

## Data Reading

This section outlines the usage of the new feature on the read operation.

At the application level, the user does not need to do anything special. Inside the HDF5 Library, when the library sees a filter identification value associated with a data item, it first searches the filter among its internal filters, and then it searches in the list of dynamically loaded filters that have been loaded in the library. If it cannot find the filter, it will search for and load the filter plugin library. Once the filter library is loaded, it will stay open until the HDF5 Library is closed. Below is a simple diagram showing the design of the library.

Figure 1. Reading data with a dynamic loaded filter

Yes

Yes

No

No

Filter pipeline

Search & load Filter1 dynamic library (H5PL\_load)

Search & load Filter2 dynamic library (H5PL\_load)

Is Filter2 enabled?

HDF5 file

H5Dread

Is Filter1 enabled?

The required changes were implemented in the H5Z\_pipeline function that resides in the H5Z.c file in the src directory of the HDF5 Library distribution.

## New HDF5 Source Files for Dynamically Loaded Filters

Three new source files were added to the HDF5 source code distribution under the src directory: H5PLextern.h, H5PLprivate.h, and H5PL.c. The H5PL.c file contains functions for managing plugins.

## Testing Dynamically Loaded Filters

The plugin.c file in the test directory contains a test for the feature. The test is controlled by the test\_plugin.sh.in script in the same directory.

The HDF Group will use bzip2 (and other plugins if they become available) to test the feature with the HDF5 libraries under development. For more on the bzip2 plugin, see the “Building an HDF5 bzip2 Plugin Example” section on page 17.

# Proposed Changes to the HDF5 Command-line Tools

## h5dump, h5ls, and h5copy

Since reading the data is transparent to applications including the HDF5 command-line tools, these command line tools do not require any changes. A small improvement was made to h5dump output to replace “UNKNOWN\_FILTER” with “USER\_DEFINED\_FILTER”.

## h5repack

As in the case of the tools mentioned in the section above, h5repack does not require any changes for reading a dataset with an applied third-party filter.

To support an application’s use of a third-party filter while repacking datasets, new syntax is needed to provide filter information to the tool. The required information is the filter identification number, the number of the filter parameters, and the values of the filter parameters. The proposed syntax to specify a user defined filter and its parameters is

h5repack –f UD={ID:k; N:m; CD\_VAL:[n1,…,nm]}…..

where k is a filter identifier, m is a number of values in the CD\_VAL array, and n1,…,nm are compression parameters. For example, to use bzip2 compression with h5repack, one would use

h5repack –f UD={ID:307; N:1; CD\_VAL:[9]} file1.h5 file2.h5

The enhancements to h5repack will be implemented in the HDF5 release 1.8.12.

# Building an HDF5 bzip2 Plugin Example

The HDF Group provides an example of the HDF5 filter plugin that can be checked out from <https://svn.hdfgroup.uiuc.edu/hdf5_plugins/trunk/BZIP2>. It contains the source code for the bzip2 plugin library and an example that uses the plugin. It requires the HDF5 Library with the dynamically loaded feature and the bzip2 library being available on the system.

The plugin and the example can be built using configure or CMake commands. For instructions on how to build with CMake, see the README.txt file in the source code distribution. The bzip2 library that can be built with CMake is available from <https://svn.hdfgroup.uiuc.edu/bzip2/>.

Please follow the instructions below to build with configure.

1. Obtain, build, and install the HDF5 Library with the dynamically loaded filter feature.

You may checkout the source from <http://svn.hdfgroup.uiuc.edu/hdf5/branches/hdf5_1_8/> or download <http://www.hdfgroup.uiuc.edu/ftp/pub/outgoing/hdf5/snapshots/v18/hdf5-1.8.11-snap16.tar.gz>.

1. Make sure that libbz2.so\* is installed on your system. On most UNIX-like systems it will be under /usr/lib.
2. Obtain the source code for the plugin using SVN, or download via FTP the source tar ball:

svn export <https://svn.hdfgroup.uiuc.edu/hdf5_plugins/trunk/BZIP2> <plugin-dir>

ftp://ftp.hdfgroup.uiuc.edu/pub/outgoing/HDF5-plugins/BZIP2-plugin.tar.gz

1. You can build in place, or you can use the “srdir” configure option. If you build in place as shown below, you will find the plugin under the <plugin-dir>/plugins directory.

cd <plugin-dir>

./configure –with-hdf5=<INSTALL-HDF5-DIR> --with-bz2lib=/usr

make

make check

make install

1. Setup the environment variable HDF5\_PLUGIN\_PATH to point to the <plugin-dir>/plugins directory with the command line shown below:

setenv HDF5\_PLUGIN\_PATH <plugin-dir>/plugins

1. Use <INSTALL-HDF5-DIR>/bin/h5dump with the h5ex\_d\_bzip2.h5 file found under the example directory of the bzip2 plugin distribution to get an output as shown in the “Reading Data with an Applied Third-party Filter” section on page 6.

# Example: Writing and Reading Compressed Data

This example shows how to write and read data compressed with the bzip2 compression.

 1 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 2

 3 This example shows how to write data and read it from a dataset

 4 using bzip2 compression.

 5 bzip2 filter is not available in HDF5.

 6 The example uses a new feature available in HDF5 version 1.8.11

 7 to discover, load and register filters at runtime.

 8

 9 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

 10

 11 #include "hdf5.h"

 12 #include <stdio.h>

 13 #include <stdlib.h>

 14

 15 #define FILE "h5ex\_d\_bzip2.h5"

 16 #define DATASET "DS1"

 17 #define DIM0 32

 18 #define DIM1 64

 19 #define CHUNK0 4

 20 #define CHUNK1 8

 21

 22 int

 23 main (void)

 24 {

 25 hid\_t file, space, dset, dcpl; /\* Handles \*/

 26 herr\_t status;

 27 H5Z\_filter\_t filter\_id = 0;

 28 char filter\_name[80];

 29 hsize\_t dims[2] = {DIM0, DIM1},

 30 chunk[2] = {CHUNK0, CHUNK1};

 31 size\_t nelmts = 1; /\* number of elements in cd\_values \*/

 32 const unsigned int cd\_values[1] = {6}; /\* bzip2 default level is 2 \*/

 33 unsigned int values\_out[1] = {99};

 34 unsigned int flags;

 35 htri\_t avail;

 36 unsigned filter\_config;

 37 int wdata[DIM0][DIM1], /\* Write buffer \*/

 38 rdata[DIM0][DIM1], /\* Read buffer \*/

 39 max,

 40 i, j;

 41

 42 /\*

 43 \* Initialize data.

 44 \*/

 45 for (i=0; i<DIM0; i++)

 46 for (j=0; j<DIM1; j++)

 47 wdata[i][j] = i \* j - j;

 48

 49 /\*

 50 \* Create a new file using the default properties.

 51 \*/

 52 file = H5Fcreate (FILE, H5F\_ACC\_TRUNC, H5P\_DEFAULT, H5P\_DEFAULT);

 53

 54 /\*

 55 \* Create dataspace. Setting maximum size to NULL sets the maximum

 56 \* size to be the current size.

 57 \*/

 58 space = H5Screate\_simple (2, dims, NULL);

 59

 60 /\*

 61 \* Create the dataset creation property list, add the bzip2

 62 \* compression filter and set the chunk size.

 63 \*/

 64 dcpl = H5Pcreate (H5P\_DATASET\_CREATE);

 65 status = H5Pset\_filter (dcpl, H5Z\_FILTER\_BZIP2, H5Z\_FLAG\_MANDATORY,

 (size\_t)6, cd\_values);

 66

 67 /\*

 68 \* Check that filter is registered with the library now.

 69 \* If it is registered, retrieve filter's configuration.

 70 \*/

 71 avail = H5Zfilter\_avail(H5Z\_FILTER\_BZIP2);

 72 if (avail) {

 73 status = H5Zget\_filter\_info (H5Z\_FILTER\_BZIP2, &filter\_config);

 74 if ( (filter\_config & H5Z\_FILTER\_CONFIG\_ENCODE\_ENABLED) &&

 75 (filter\_config & H5Z\_FILTER\_CONFIG\_DECODE\_ENABLED) )

 76 printf ("bzip2 filter is available for encoding and decoding.\n");

 77 }

 78 status = H5Pset\_chunk (dcpl, 2, chunk);

 79

 80 /\*

 81 \* Create the dataset.

 82 \*/

 83 printf ("....Writing bzip2 compressed data ................\n");

 84 dset = H5Dcreate (file, DATASET, H5T\_STD\_I32LE, space, H5P\_DEFAULT, dcpl,

 85 H5P\_DEFAULT);

 86

 87 /\*

 88 \* Write the data to the dataset.

 89 \*/

 90 status = H5Dwrite (dset, H5T\_NATIVE\_INT, H5S\_ALL, H5S\_ALL, H5P\_DEFAULT,

 91 wdata[0]);

 92

 93 /\*

 94 \* Close and release resources.

 95 \*/

 96 status = H5Pclose (dcpl);

 97 status = H5Dclose (dset);

 98 status = H5Sclose (space);

 99 status = H5Fclose (file);

 100 status - H5close();

 101

 102

 103 printf ("....Close the file and reopen for reading ........\n");

 104 /\*

 105 \* Now we begin the read section of this example.

 106 \*/

 107

 108 /\*

 109 \* Open file and dataset using the default properties.

 110 \*/

 111 file = H5Fopen (FILE, H5F\_ACC\_RDONLY, H5P\_DEFAULT);

 112 dset = H5Dopen (file, DATASET, H5P\_DEFAULT);

 113

 114 /\*

 115 \* Retrieve dataset creation property list.

 116 \*/

 117 dcpl = H5Dget\_create\_plist (dset);

 118

 119 /\*

 120 \* Retrieve and print the filter id, compression level and filter's name for bzip2.

 121 \*/

 122 filter\_id = H5Pget\_filter2 (dcpl, (unsigned) 0, &flags, &nelmts,

 values\_out, sizeof(filter\_name), filter\_name, NULL);

 123 printf ("Filter info is available from the dataset creation property \n ");

 124 printf (" Filter identifier is ");

 125 switch (filter\_id) {

 126 case H5Z\_FILTER\_BZIP2:

 127 printf ("%d\n", filter\_id);

 128 printf (" Number of parameters is %d with the value %u\n",

 nelmts, values\_out[0]);

 129 printf (" To find more about the filter check %s\n",

 filter\_name);

 130 break;

 131 default:

 132 printf ("Not expected filter\n");

 133 break;

 134 }

 135 /\*

 136 \* Check that filter is not registered with the library yet.

 137 \*/

 138 avail = H5Zfilter\_avail(H5Z\_FILTER\_BZIP2);

 139 if (!avail)

 140 printf ("bzip2 filter is not yet available after the H5Pget\_filter

 call.\n");

 141 else

 142 return 1;

 143

 144

 145 /\*

 146 \* Read the data using the default properties.

 147 \*/

 148 printf ("....Reading bzip2 compressed data ................\n");

 149 status = H5Dread (dset, H5T\_NATIVE\_INT, H5S\_ALL, H5S\_ALL, H5P\_DEFAULT,

 150 rdata[0]);

 151

 152 /\*

 153 \* Find the maximum value in the dataset, to verify that it was

 154 \* read correctly.

 155 \*/

 156 max = rdata[0][0];

 157 for (i=0; i<DIM0; i++)

 158 for (j=0; j<DIM1; j++) {

 159 /\*printf("%d \n", rdata[i][j]); \*/

 160 if (max < rdata[i][j])

 161 max = rdata[i][j];

 162 }

 163 /\*

 164 \* Print the maximum value.

 165 \*/

 166 printf ("Maximum value in %s is %d\n", DATASET, max);

 167 /\*

 168 \* Check that filter is registered with the library now.

 169 \*/

 170 avail = H5Zfilter\_avail(H5Z\_FILTER\_BZIP2);

 171 if (avail)

 172 printf ("bzip2 filter is available now since H5Dread triggered loading

 of the filter.\n");

 173

 174

 175 /\*

 176 \* Close and release resources.

 177 \*/

 178 status = H5Pclose (dcpl);

 179 status = H5Dclose (dset);

 180 status = H5Fclose (file);

 181

 182 return 0;

 183 }

# HDF5 bzip2 Filter Plugin

This section provides partial code for the bzip2 filter plugin. For complete code, check <http://svn.hdfgroup.uiuc.edu/hdf5_plugins/trunk/BZIP2/src/H5Zbzip2.c>

 1 /\*

 2 \* This file is an example of an HDF5 filter plugin.

 3 \* The filter function H5Z\_filter\_bzip2 was adopted from

 4 \* PyTables http://www.pytables.org.

 5 \* The plugin can be used with the HDF5 Library version 1.8.11 to read

 6 \* HDF5 datasets compressed with bzip2 created by PyTables.

 7 \*/

 8

 9 /\*

 10 \*

 11 Copyright Notice and Statement for PyTables Software Library and Utilities:

 12

…

 45 \*/

 46 #include <sys/types.h>

 47 #include <stdlib.h>

 48 #include <string.h>

 49 #include <assert.h>

 50 #include <stdio.h>

 51 #include <H5PLextern.h>

 52

 53 #include <bzlib.h>

 54

 55 static size\_t H5Z\_filter\_bzip2(unsigned int flags, size\_t cd\_nelmts,

 56 const unsigned int cd\_values[], size\_t nbytes,

 57 size\_t \*buf\_size, void \*\*buf);

 58

 59 /\*

…

 62 \* See http://www.hdfgroup.org/services/contributions.html for more information.

 63 \*

 64 \* If you intend your plugin to be used by others, please register your filter

 65 \* with The HDF Group.

 66 \*/

 67 #define H5Z\_FILTER\_BZIP2 307

 68

 69 const H5Z\_class2\_t H5Z\_BZIP2[1] = {{

 70 H5Z\_CLASS\_T\_VERS, /\* H5Z\_class\_t version \*/

 71 (H5Z\_filter\_t)H5Z\_FILTER\_BZIP2, /\* Filter id number \*/

 72 1, /\* encoder\_present flag (set to true) \*/

 73 1, /\* decoder\_present flag (set to true) \*/

 74 "HDF5 bzip2 filter; see http://www.hdfgroup.org/services/contributions.html",

 75 /\* Filter name for debugging \*/

 76 NULL, /\* The "can apply" callback \*/

 77 NULL, /\* The "set local" callback \*/

 78 (H5Z\_func\_t)H5Z\_filter\_bzip2, /\* The actual filter function \*/

 79 }};

 80

 81 H5PL\_type\_t H5PLget\_plugin\_type(void) {return H5PL\_TYPE\_FILTER;}

 82 const void \*H5PLget\_plugin\_info(void) {return H5Z\_BZIP2;}

 83

 84 static size\_t H5Z\_filter\_bzip2(unsigned int flags, size\_t cd\_nelmts,

 85 const unsigned int cd\_values[], size\_t nbytes,

 86 size\_t \*buf\_size, void \*\*buf)

 87 {

 88 char \*outbuf = NULL;

 89 size\_t outbuflen, outdatalen;

 90 int ret;

 91

 92 if (flags & H5Z\_FLAG\_REVERSE) {

 93

 94 /\*\* Decompress data.

 95 \*\*

…

 102 \*\*/

 103

 104 bz\_stream stream;

 105 char \*newbuf = NULL;

 106 size\_t newbuflen;

 107

 108 /\* Prepare the output buffer. \*/

 109 outbuflen = nbytes \* 3 + 1; /\* average bzip2 compression ratio is 3:1 \*/

 110 outbuf = malloc(outbuflen);

 111 if (outbuf == NULL) {

 112 fprintf(stderr, "memory allocation failed for bzip2 decompression\n");

 113 goto cleanupAndFail;

 114 }

 115

 116 /\* Use standard malloc()/free() for internal memory handling. \*/

 117 stream.bzalloc = NULL;

 118 stream.bzfree = NULL;

 119 stream.opaque = NULL;

 120

 121 /\* Start decompression. \*/

 122 ret = BZ2\_bzDecompressInit(&stream, 0, 0);

 123 if (ret != BZ\_OK) {

 124 fprintf(stderr, "bzip2 decompression start failed with error %d\n", ret);

 125 goto cleanupAndFail;

 126 }

 127

 128 /\* Feed data to the decompression process and get decompressed data. \*/

 129 stream.next\_out = outbuf;

 130 stream.avail\_out = outbuflen;

 131 stream.next\_in = \*buf;

 132 stream.avail\_in = nbytes;

 133 do {

 134 ret = BZ2\_bzDecompress(&stream);

 135 if (ret < 0) {

 136 fprintf(stderr, "BUG: bzip2 decompression failed with error %d\n", ret);

 137 goto cleanupAndFail;

 138 }

 139

 140 if (ret != BZ\_STREAM\_END && stream.avail\_out == 0) {

 141 /\* Grow the output buffer. \*/

 142 newbuflen = outbuflen \* 2;

 143 newbuf = realloc(outbuf, newbuflen);

 144 if (newbuf == NULL) {

 145 fprintf(stderr, "memory allocation failed for bzip2 decompression\n");

 146 goto cleanupAndFail;

 147 }

 148 stream.next\_out = newbuf + outbuflen; /\* half the new buffer behind \*/

 149 stream.avail\_out = outbuflen; /\* half the new buffer ahead \*/

 150 outbuf = newbuf;

 151 outbuflen = newbuflen;

 152 }

 153 } while (ret != BZ\_STREAM\_END);

 154

 155 /\* End compression. \*/

 156 outdatalen = stream.total\_out\_lo32;

 157 ret = BZ2\_bzDecompressEnd(&stream);

 158 if (ret != BZ\_OK) {

 159 fprintf(stderr, "bzip2 compression end failed with error %d\n", ret);

 160 goto cleanupAndFail;

 161 }

 162

 163 } else {

 164

 165 /\*\* Compress data.

 166 \*\*

…

 171 \*\*/

 172

 173 unsigned int odatalen; /\* maybe not the same size as outdatalen \*/

 174 int blockSize100k = 9;

 175

 176 /\* Get compression block size if present. \*/

 177 if (cd\_nelmts > 0) {

 178 blockSize100k = cd\_values[0];

 179 if (blockSize100k < 1 || blockSize100k > 9) {

 180 fprintf(stderr, "invalid compression block size: %d\n", blockSize100k);

 181 goto cleanupAndFail;

 182 }

 183 }

 184

 185 /\* Prepare the output buffer. \*/

 186 outbuflen = nbytes + nbytes / 100 + 600; /\* worst case (bzip2 docs) \*/

 187 outbuf = malloc(outbuflen);

 188 if (outbuf == NULL) {

 189 fprintf(stderr, "memory allocation failed for bzip2 compression\n");

 190 goto cleanupAndFail;

 191 }

 192

 193 /\* Compress data. \*/

 194 odatalen = outbuflen;

 195 ret = BZ2\_bzBuffToBuffCompress(outbuf, &odatalen, \*buf, nbytes,

 196 blockSize100k, 0, 0);

 197 outdatalen = odatalen;

 198 if (ret != BZ\_OK) {

 199 fprintf(stderr, "bzip2 compression failed with error %d\n", ret);

 200 goto cleanupAndFail;

 201 }

 202 }

 203

 204 /\* Always replace the input buffer with the output buffer. \*/

 205 free(\*buf);

 206 \*buf = outbuf;

 207 \*buf\_size = outbuflen;

 208 return outdatalen;

 209

 210 cleanupAndFail:

 211 if (outbuf)

 212 free(outbuf);

 213 return 0;

 214 }

# Example of Makefile

This is an example for Makefile. Building shared libraries may be tricky. Please consult system documentation for the flags on your system. You may also want to consider using libtool as we do in the bzip2 plugin example.

 1 CFLAGS = -fPIC -g

 2 CC = gcc

 3 HDF5\_INSTALL = /mnt/scr1/\_tmp/\_build/hdf5/

 4 BZIP2\_INSTALL = /mnt/scr1/bzip2-1.0.5

 5 MAJOR = 0

 6 MINOR = 1

 7 NAME1 = H5Zbzip2

 8 VERSION = $(MAJOR).$(MINOR)

 9

 10 # Include files in hdf5/src build/src directories for hdf5.h and

 H5pubconf.h

 11 INCLUDES = -I./ -I$(HDF5\_INSTALL)/include -I$(BZIP2\_INSTALL)

 12

 13 lib: lib$(NAME1).so.$(VERSION)

 14

 15 $(NAME1).o: $(NAME1).c

 16 $(CC) $(CFLAGS) $(INCLUDES) -c $(NAME1).c

 17

 18 lib$(NAME1).so.$(VERSION): $(NAME1).o

 19 $(CC) -shared -Wl,-soname,lib$(NAME1).so.$(MAJOR) $^ -o $@ -L/$(BZIP2\_INSTALL) -lbz2

 20

 21 clean:

 22 $(RM) \*.o \*.so\*

 23

 24 distclean:

 25 $(RM) \*.o \*.so\*

# Revision History

|  |  |
| --- | --- |
| *March 31, 2013* | Version 1 shared with developers who worked on the project. |
| *April 9, 2013* | Version 2 shared with The HDF Group developers. |
| *April 16, 2013* | Version 3: accepted changes from Mark |

# References

1. The HDF Group. “HDF5 Documentation,” <http://www.hdfgroup.org/HDF5/doc/doc-info.html> (November 15, 2012).