Support HDF5 1.8 in the HDF-Java Products

Peter Cao
xcao@hdfgroup.org
The HDF Group

HDF5 1.8 includes a number of new features that offer many users of HDF5 substantial performance improvements and expanded capabilities. Many of these features can only be accessed via revised API calls. To support these new APIs and new features in the HDF-Java products, we need make changes to the current the HDF-Java products. The changes may affect applications that use the HDF-Java products, especially the HDF5 Java wrapper.

This document discusses what work is needed to support HDF5 1.8 in the HDF-Java products and how the work is done. This document includes the following sections:

- Changes and updates to the HDF5 JNI.
- Stages of supporting HF5 1.8 in all the HDF-Java products.

The main work discussed in the RFC is the changes and updates to JHI5 for supporting the new APIs added HDF5 1.8 library. Changes will also be made to the package names and class structure. Although the changes of the package names and structure are not related to supporting HDF5 1.8 features, they will affect applications that use the HDF-Java products. We are looking for your input. Please send your concerns and suggestions to Peter Cao at xcao@hdfgroup.org.

Table of Contents

1 Introduction 3
2 Java HDF5 Interface (JHI5) 3
2.1 Work for fully supporting HDF5 1.8 in JHI5 4
2.2 API compatibility between HDF5 1.6 and HDF5 1.8 5
2.3 Change of package names 7
2.3.1 The changes 7
2.3.2 The impact 7
2.3.3 The solution 8
2.4 Package and class structure for HDF5 JNI 8
2.4.1 Stay with the current class structure 8
2.4.2 Use a new class structure 8
3 Stages of development for supporting HDF5 1.8 in the HDF-Java 9
3.1 Stage one: support of HDF5 1.8 with HDF5 1.6 compatibility flag 10
3.2 Stage two: implement and test HDF5 1.8 JNI APIs 10
3.3 Stage three: using new APIs in the object package 11
3.4 Stage four: support new features in HDFView 11
Revision History 12
Appendix A – Prioritized List of new APIs 13

1 Introduction

HDF5 1.8 represents a major update to the HDF5 Library, utilities, and file format. The changes introduced in HDF5 1.8 provide new capabilities and improve performance.

The current HDF-Java does not support the new APIs (150+) added to the HDF5 1.8 library. Since the first release of HDF5 1.8 (version 1.8.0) in February 2008, HDF5 1.8 has been adopted by many users because of its new features. There is great demand for full support of HDF5 1.8 in the HDF-Java products. This RFC presents the approaches and the work for supporting HDF5 1.8 in HDF-Java.

There are three distinct HDF5 Java Products:

- Java HDF5 Interface (JHI5): the Java Native Interface to the standard HDF5 library.
- Java HDF Object Package: a Java package that implements HDF data objects in an object-oriented form.
- HDFView: a visual tool for browsing and editing HDF4 and HDF5 files.

HDFView is built on the Java HDF Object Package, which in turn is built on the Java HDF5 Interface. Modifications will be needed in all three products to support HDF5 1.8 features. The necessary changes are well understood for JHI5, as it most closely reflects the public HDF5 C library API. This RFC will mainly focus on the update of JHI5.

The Object Package and HDFView, which also work with HDF4, require more careful design and consideration. The design for supporting HDF5 1.8 with these products has not yet been done, and therefore it is impossible to estimate the level of effort to fully support HDF5 1.8 features at this time. The agile development approach will be more efficient for supporting HDF5 1.8 features in the Java HDF Object Package and HDFView.

2 Java HDF5 Interface (JHI5)

The HDF5 JNI (Java Native Interface) includes two parts: the Java code and the native code.

- The Java code or the Java wrapper class (H5.java) is a list of Java native methods that are one-to-one mapping to the HDF5 APIs. For example, H5.H5Dopen() is the Java method to call the HDF5 library API, H5Dopen(hid_t loc_id, const char *name). The “*native*” is a Java keyword to mark that this method is implemented in a language other than Java, in JHI5, it is C. Below is an example of Java wrapper methods.

```
public class H5 {
…
    public static native int H5Dopen(int loc_id, String name)
    throws HDF5LibraryException, NullPointerException;
…
}
```

- The native code is the C implementation for the Java wrapper method. For example, the C native implementation for H5.H5Open() looks like the following:

```
JNIEXPORT jint JNICALL Java_ncsa_hdf_hdf5lib_H5_H5Dopen
  (JNIEnv *env, jclass clss, jint loc_id, jstring name)
{
 ...
    status = H5Dopen((hid_t)loc_id, file );
...
}
```

The C native implementation is invisible to Java applications and should not be directly used. Java applications must use the Java wrapper to call HDF5 library.

Changes in the HDF5 JNI will include:

- Changes for some of the HDF5 Java wrapper methods for the compatibility between HDF5 1.6 and HDF5 1.8.
- New APIs added to JHI5 to support HDF5 1.8 library.
- Changes of the package names.
- Changes of the class structure of the HDF5 JNI.

Changing package names and class structures is not related to supporting new features of HDF5 1.8. It should not hold back the work for JHI5.

2.1 Work for fully supporting HDF5 1.8 in JHI5

The current release of JHI5, based on HDF5 1.6, implements 237 functions. There are 64 functions in the public HDF5 1.6 C library API that are not available in the current JHI5. Among the 64 function, some have function pointers or complex C data structure, which cannot be supported from Java directly. Some of the functions are not important to general users, such as certain functions of property list. In HDF5 1.8, there are 19 API compatibility macros with associated renamed and new Functions. Eleven of those functions are available in the current JHI5, and would need to be updated. One hundred and sixty nine additional functions were introduced in the HDF5 1.8 C library API.

Table 1 -- Status and priorities of JHI5

	HDF5 1.6 C API Public functions	JHI5 API Implemented	HDF5 1.8 C API Renamed functions	HDF5 1.8 C API new functions
No. of APIs	304	237	19	169
Priority	Low priority for the 64 functions that are currently not implemented in JHI5.	Continue to support.	Highest priority to all the 11 implemented functions.	See the table below.
Testing	N/A	Stay as they are. Add unit tests in the future.	Unit tests for all the 11 implemented functions.	Unit tests for the functions added to JHI5.

There are 169 new APIs added to HDF5 1.8. The new APIs are not equally important. We recommend a staged approach to make the HDF5 1.8 features available in JHI5. For details of the changes of APIs, please visit http://www.hdfgroup.org/HDF5/doc/ADGuide/Changes.html.

Table 2 shows the priorities of the new APIs. Appendix A presents a detailed list. The priority in the table is our first assessment and subject to change based on input. The priorities are defined as follow:

- **High priority**: new features or improvements, which are critical to general users. These APIs will be added to JHI5 and fully tested first.
- **Medium priority**: new features or improvements, which are useful but not critical to general users. These APIs will be implemented after we complete all the APIs with higher priority.
- **Low priority**: These APIs have function pointer or complex data structure or they are not important to general users. These APIs will be added to JHI5 one by one as requested.

The complexity of each API is different. The effort for implementing and testing is different. Based on our first estimation, the average time for implementing and testing an API is about 3 hours.

Table 2 -- New APIs with proposed priorities

	Priority		
Package	**High**	**Medium**	**Low**
Attribute	8	3	3
Dataset	5	0	0
Dataspace	0	0	3
Datatype	2	2	3
Error	0	0	18
File	2	0	5
Group	7	0	0
Identifier	2	0	11
Link	13	3	5
Object	8	9	0
Property	4	29	23
Reference	1	0	0
Total no. of APIs	52	46	71
Total no. of hours	153	138	213

2.2 API compatibility between HDF5 1.6 and HDF5 1.8

As part of HDF5 1.8.x release, twenty-three functions that existed in previous versions of the library were updated with new calling parameters and given new names. The updated versions of the functions have a "2" at the end of the original function name. The original versions of these functions were retained and renamed to have a "1" at the end of the original function name. For example,

- **The 1.6 version**: H5Dopen1(hid_t loc_id, const char *name)
- **The 1.8 version**: H5Dopen2(hid_t loc_id, const char *name, hid_t dapl_id)

To make the APIs compatible, the HDF5 1.8 library uses “API Compatibility Macros”, with the same names as the original function names. For information on “API Compatibility Macros”, see “API Compatibility Macros in HDF5“at http://www.hdfgroup.org/HDF5/doc/RM/APICompatMacros.html.

The issue of maintaining the two versions of the same functions exists in the HDF5 Java wrapper. Since Java supports overloading and variable arity, the problem can be easily handled..

The versions of APIs with different arguments be easily handled with a function overloading in Java. For example, H5Dopen1(…) and H5Dopen2(…) can be handled in a single function name.

- **C functions**:
  - hid_t H5Dopen1(hid_t loc_id, const char *name)
  - hid_t H5Dopen2(hid_t loc_id, const char *name, hid_t dapl_id)
- **Java function**:
  - int H5Dopen(int loc_id, String name);
  - int H5Dopen(int loc_id, String name, int dapl_id);

The implementation will include three functions:

- Two public function, used by applications.
- A private native function that calls the HDF5 1.8 API. This private function is used internally by the wrapper. The native function for HDF5 1.6 API is not needed since the 1.8 API always exists in the 1.8 library and includes all the functionality of the 1.6 library API.

The following code shows the implementation of H5Dopen(…) as an example. The private native function H5Dopen2(…) is linked to the 1.8 version API and called by the public function H5Dopen(…).

```
public static int H5Dopen(int loc_id, String name)
  throws HDF5LibraryException, NullPointerException
  {
      return H5Dopen2(loc_id, name, H5Pconstant.H5P_DEFAULT);
  }

public static int H5Dopen(int loc_id, String name, int dapl_id)
  throws HDF5LibraryException, NullPointerException
  {
      return H5Dopen2(loc_id, name, dapl_id);
  }

// private function for internal use only.
private synchronized static native int H5Dopen2(int loc_id, String name, int dapl_id)
  throws HDF5LibraryException, NullPointerException;
```

Java applications do not need to know which version of the library APIs are used. The public functions will work for HDF5 1.6 API and 1.8 API. For example, current applications that use the 1.6 API, e.g. H5Dopen(loc_id, “/dset1”), will work under the new implementation. Applications that use the 1.8 APIs, e.g. H5Dopen(loc_id, “/dset1”, dapl_id) will also work under the new implementation. Both H5Dopen(loc_id, “/dset1”) and H5Dopen(loc_id, “/dset1”, dapl_id) are valid function calls for the Java wrapper.

2.3 Change of package names

Currently all package names start with "ncsa.hdf". "ncsa" will be removed from the new package names. The package name "ncsa.hdf" was used because the HDF-Java products were initially developed at NCSA. The HDF-Java products are now owned and developed by The HDF Group. To avoid confusion, "ncsa" will be removed from the package names.

This change is not directly related to support of HDF5 1.8 features and is not limited to HDF5 Java products. The same name change will be applied to HDF4 Java products. Although this change is one related to any feature, it would be a reasonable time to update the package names with the changes required to support HDF5 1.8 features.

2.3.1 The changes

We propose three options for the package names:

- Option 1: no change, keep the current package names.
- Option 2: remove "ncsa" to use shorter package names.
- Option 3: use Sun's Java naming conventions, which will be "*org.hdfgroup.hdf*".

Table 3 shows the current package names and the proposed package names. The change will be applied to all of the HDF-Java products: the HDF4 Java wrapper, and the HDF5 Java wrapper, the HDF-Java object package, and HDFView.

Table 3 -- the HDF-Java product package names

Option 1	Option 2	Option 3
ncsa.hdf.hdf5lib	hdf.hdf5lib	org.hdfgroup.hdf5lib
ncsa.hdf.hdf5lib.exceptions	hdf.hdf5lib.exceptions	org.hdfgroup.hdf5lib.exceptions
ncsa.hdf.hdflib	hdf.hdflib	org.hdfgroup.hdflib
ncsa.hdf.object	hdf.object	org.hdfgroup.object
ncsa.hdf.object.h4	hdf.object.h4	org.hdfgroup.object.h4
ncsa.hdf.object.h5	hdf.object.h5	org.hdfgroup.object.h5
ncsa.hdf.view	hdf.view	org.hdfgroup.view

2.3.2 The impact

If we change the package names, the change will affect all applications that use the HDF-Java products, including,

- Applications that use HDF5 Java wrappers (H5.java).
- Applications that are built on the HDF-Java object package.
- HDFView plugins that are implementing the plugin interfaces.

The new name will cause compile errors, and possible runtime errors, to existing applications depending on how the HDF-Java classes are used.

- **Compile errors**: errors will be caught at compile time for any direct use of the HDF-Java classes, such as “import ncsa.hdf.object.*”.
- **Runtime errors**: errors will occur at runtime for dynamic loading that uses the HDF-Java classes. For example, Class.forName("ncsa.hdf.view.ViewManager”) will throw a ClassNotFoundException at run time.

2.3.3 The solution

Fixing the problem is straightforward, just changing the package names; however, it can be tedious. The HDF Group will provide scripts that go through application source code to change package names as listed in Table 3.

2.4 Package and class structure for HDF5 JNI

HDF5 1.8 added over 150 new functions (total over 400). To be able to use these new functions in HDF-Java, we need to add them to the HDF5 Java wrapper. We propose two options for packing the APIs: keep the current class structure or break it into smaller packages.

2.4.1 Stay with the current class structure

There are two main classes in the current HDF5 JNI package:

- **hdf.hdf5lib.H5** -- This class contains all the HDF5 APIs.
- **hdf.hdf5lib.HDF5Constants** -- This class contains all HDF5 constants and enumerated types.

We will add the new APIs to the wrapper class, hdf.hdf5lib.H5, and the new constants to the constant class, hdf.hdf5lib.HDF5Constants. No additional classes will be created.

Pros: applications do not need any change except the package names as listed in Table 3.

Cons: maintaining all 400+ APIs in one class (hdf.hdf5lib.H5) can be a challenge.

2.4.2 Use a new class structure

Basically we divide the wrapper and constants into separate classes based on the C interface structure.

The wrapper package, hdf.h5, will include the following classes:

hdf.h5.H5.java	-- Library Functions
hdf.h5.H5A.java	-- Attribute Interface
hdf.h5.H5D.java	-- Dataset Interface
hdf.h5.H5E.java	-- Error Interface
hdf.h5.H5F.java	-- File Interface
hdf.h5.H5G.java	-- Group Interface
hdf.h5.H5I.java	-- Identifier Interface
hdf.h5.H5L.java	-- Link Interface

- hdf.h5.H5O.java -- Object Interface
- hdf.h5.H5P.java -- Property List Interface
- hdf.h5.H5R.java -- Reference Interface
- hdf.h5.H5S.java -- Dataspace Interface
- hdf.h5.H5T.java -- Datatype Interface
- hdf.h5.H5Z.java -- Compression Interface

The constants package, hdf.h5.constants, has a similar list of classes. The enums will be treated as constants.

- hdf.h5.constants.H5constants.java -- Library constants
- hdf.h5.constants.H5Aconstants.java -- Attribute constants
- hdf.h5.constants.H5Dconstants.java -- Dataset constants
- hdf.h5.constants.H5Econstants.java -- Error constants
- hdf.h5.constants.H5Fconstants.java -- File constants
- hdf.h5.constants.H5Gconstants.java -- Group constants
- hdf.h5.constants.H5Iconstants.java -- Identifier constants
- hdf.h5.constants.H5Lconstants.java -- Link constants
- hdf.h5.constants.H5Oconstants.java -- Object constants
- hdf.h5.constants.H5Pconstants.java -- Property List constants
- hdf.h5.constants.H5Rconstants.java -- Reference constants
- hdf.h5.constants.H5Sconstants.java -- Dataspace constants
- hdf.h5.constants.H5Tconstants.java -- Datatype constants
- hdf.h5.constants.H5Zconstants.java -- Compression constants

In this new package structure, the C package name in the function name will be removed to avoid redundancy. For example, H5D.open(…) will be used instead of H5D.H5Dopen(…).

Pros: easier to maintain as APIs are divided into packages according to the library API structure.

Cons: all applications have to change to the new class names. For example, H5.H5Dopen(..) will be changed to H5D. open(…) since H5Dopen(..) will be moved to H5D class as H5D.open(…).

3 Stages of development for supporting HDF5 1.8 in the HDF-Java

There are four stages in order to fully support HF5 1.8 in the HDF-Java products:

- Support HDF5 1.8 with the HDF5 1.6 compatibility flag -- *completed in the HDF-Java 2.5 release (February 2009)*.
- Implement HDF5 1.8 JNI APIs – *the major work to be completed next*.

- Use HDF5 1.8 APIs and features in the HDF-Java object package – *needs more investigation*.
- Support HDF5 1.8 features in HDFView – *this needs more investigation*.

3.1 Stage one: support of HDF5 1.8 with HDF5 1.6 compatibility flag

Stage one is to make the HDF-Java work on HDF5 1.8 library with HDF5 1.6 compatibility flag. This was done in the HDF-Java 2.5 release (February 2009).

Values of some constant variables are fixed at HDF5 1.6 but they are dynamically determined at HDF5 1.8 such as H5E_DATASET. This change does not affect C applications but it causes compile errors for the Java compiler. Some minor changes in the HDF5 JNI fixed the problem.

The HDF-Java version 2.5 takes advantage of many of the improvements introduced in HDF5 Release 1.8.0. However, since it relies on the 1.6 APIs, this version does not include the 150+ new APIs, and cannot use features that are only accessible via those APIs.

HDF5 files created with HDF5 1.8.x APIs can be opened, but some objects that use new file format features, such as external links, are accessible with the HDF-Java products. All files and objects written by version 2.5 of the HDF-Java products will be readable by applications linked with 1.6.x or 1.8.x versions of the HDF5 library. Please refer to the related HDF5 documentation for details:

- API Compatibility Macros in HDF5 (http://www.hdfgroup.org/HDF5/doc/RM/APICompatMacros.html)

3.2 Stage two: implement and test HDF5 1.8 JNI APIs

This task includes reorganizing the package structure, adding the 150+ new APIs to the Java wrapper, and implementing and testing the C JNI functions. The work will depend on the class structures as discussed in section 2.4.

- Option 1: stay with the current class structure.
- Option 2: use new class structure.

Table 4 -- Work for implementing HDF5 1.8 JNI APIs

Task	Option 1	Option 2	Status
Design and documentation	Yes	Yes	Need feedback and revision
Reorganizing the package	No	Yes	
Revise configuration file	No	Yes	
Revise make files	No	Yes	
Java wrapper	Yes	Yes	Most of the work is done. Need revision based on the design.
C JNI implementation	Yes	Yes	Some prototype work and research.
Testing	Yes	Yes	Some prototype work and research.

Test driven development (TDD) will be used for implementing the new HDF5 1.8 JNI APIs. The implementation of each new APIs will follow the development circle below. Details will be discussed in a separate document, “Development Steps for Implementing HDF5 JNI”.

- Write tests for the API to be implemented.

- Check if all tests fail (all tests should fail).
- Write production code.
- Run all tests (all tests should succeed).
- Clean up and check in code.

Current HDF5 JNI APIs are not tested. We will not test all the current APIs at this stage for two reasons. First, testing all the current APIs is a lot of work and will delay the work for HDF5 1.8 support. Second, the current APIs have been used for many years and proved to be stable. However, we will all tests for all the APIs in the future.

3.3 Stage three: using new APIs in the object package

Some of the new functions in HDF5 1.8 can increase the performance of HDF-Java. For example, the new H5Ocopy() function in the HDF5 library has better performance than the current Java copy() function. Another example is to support external links. Further investigation is needed for what is needed to be done and the time estimation.

3.4 Stage four: support new features in HDFView

Below are some examples of the new features that may be added to HDFView. For details of the new features of HDF5 1.8, please refer to the related HDF5 documents at

- What's New in HDF5 1.8.0 (http://www.hdfgroup.org/HDF5/doc/ADGuide/WhatsNew180.html)
- API Compatibility Macros in HDF5 (http://www.hdfgroup.org/HDF5/doc/RM/APICompatMacros.html)
- New Features in HDF5 Release 1.8.0 and Format Compatibility Considerations (http://www.hdfgroup.org/HDF5/doc/ADGuide/CompatFormat180.html)

Table 5 – Work for adding new features in HDFView

Related Objects	New Features
File	Use latest format
	Creation order
Group	External link
	Compact or indexed link
	NULL dataspace
Dataset	Faster access
Attribute	Creation order
	Shared
	Large attribute

Revision History

September 01, 2009:	Version 1 combined the original documents regarding HDF51.8 support in HDF-Java.
September22 , 2009:	Version 2 reorganized the structure and included editorial changes.
September29 , 2009:	Version 3 included changes based on feedback and input from the staff of The HDF Group.

Appendix A – Prioritized List of new APIs

The table below shows a list of new APIs added to HDF5 1.8. The priority given in the table is our first assessment and is subject to change based on input.

The priorities are defined as follow:

- **High priority**: new features or improvements, which are critical to general users. These APIs will be added to JHI5 and fully tested first.
- **Medium priority**: new features or improvements, which are useful but not critical to general users. These APIs will be implemented after we complete all the APIs with higher priority.
- **Low priority**: These APIs have function pointer or complex data structure or they are not important to general users. These APIs will be added to JHI5 one by one as requested.

For details of the APIs, please see the "*HDF5: API Specification Reference Manual*" at http://www.hdfgroup.org/HDF5/doc/RM/RM_H5Front.html.

High	Medium	Low
H5Acreate2	H5Adelete_by_idx	H5Aget_create_plist
H5Aget_info	H5Adelete_by_name	H5Aiterate_by_name
H5Aget_info_by_idx	H5Aget_storage_size	H5Aiterate2
H5Aget_info_by_name	H5Lcreate_ud	H5Eauto_is_v2
H5Aget_name_by_idx	H5Lvisit	H5Eclear2
H5Aopen	H5Lvisit_by_name	H5Eclose_msg
H5Aopen_by_idx	H5Odecr_refcount	H5Eclose_stack
H5Arename_by_name	H5Oget_comment	H5Ecreate_msg
H5Dcreate_anon	H5Oget_comment_by_name	H5Eget_auto2
H5Dcreate2	H5Oincr_refcount	H5Eget_class_name
H5Dget_access_plist	H5Oopen_by_addr	H5Eget_current_stack
H5Dopen2	H5Oset_comment	H5Eget_msg
H5Dset_extent	H5Oset_comment_by_name	H5Eget_num
H5Fget_info	H5Ovisit	H5Epop
H5Fget_intent	H5Ovisit_by_name	H5Eprint2
H5Gcreate_anon	H5Pget_attr_creation_order	H5Epush2
H5Gcreate2	H5Pget_attr_phase_change	H5Eregister_class
H5Gget_create_plist	H5Pget_copy_object	H5Eset_auto2
H5Gget_info	H5Pget_create_intermediate_group	H5Eset_current_stack
H5Gget_info_by_idx	H5Pget_data_transform	H5Eunregister_class
H5Gget_info_by_name	H5Pget_elink_acc_flags	H5Ewalk2
H5Gopen2	H5Pget_link_creation_order	H5Fget_mdc_config
H5Iget_type_ref	H5Pget_link_phase_change	H5Fget_mdc_hit_rate

H5Inmembers	H5Pget_local_heap_size_hint	H5Fget_mdc_size
H5Lcopy	H5Pget_shared_mesg_index	H5Freset_mdc_hit_rate_stats
H5Lcreate_external	H5Pget_shared_mesg_nindexes	H5Fset_mdc_config
H5Lcreate_hard	H5Pget_shared_mesg_phase_change	H5Iclear_type
H5Lcreate_soft	H5Pset_copy_object	H5Idec_type_ref
H5Ldelete	H5Pset_create_intermediate_group	H5Idestroy_type
H5Ldelete_by_idx	H5Pset_data_transform	H5Iinc_type_ref
H5Lexists	H5Pset_elink_acc_flags	H5Iis_valid
H5Lget_info	H5Pset_elink_cb	H5Iobject_verify
H5Lget_info_by_idx	H5Pset_elink_fapl	H5Iregister
H5Lget_name_by_idx	H5Pset_elink_prefix	H5Iregister_type
H5Lget_val	H5Pset_est_link_info	H5Iremove_verify
H5Lget_val_by_idx	H5Pset_fapl_direct	H5Isearch
H5Lmove	H5Pset_link_phase_change	H5Itype_exists
H5Oclose	H5Pset_local_heap_size_hint	H5Lis_registered
H5Ocopy	H5Pset_nbit	H5Literate
H5Oget_info	H5Pset_nlinks	H5Lregister
H5Oget_info_by_idx	H5Pset_scaleoffset	H5Lunpack_elink_val
H5Oget_info_by_name	H5Pset_shared_mesg_index	H5Lunregister
H5Olink	H5Pset_shared_mesg_nindexes	H5Pget_char_encoding
H5Oopen	H5Pset_shared_mesg_phase_change	H5Pget_chunk_cache
H5Oopen_by_idx	H5Tcommit_anon	H5Pget_elink_cb
H5Pget_libver_bounds	H5Tget_create_plist	H5Pget_elink_fapl
H5Pget_nlinks		H5Pget_elink_prefix
H5Pset_libver_bounds		H5Pget_est_link_info
H5Pset_link_creation_order		H5Pget_fapl_direct
H5Rget_name		H5Pget_filter_by_id2
H5Tcommit2		H5Pget_filter2
H5Topen2		H5Pget_mdc_config
		H5Pget_type_conv_cb
		H5Pinsert2
		H5Pregister2
		H5Pset_attr_creation_order
		H5Pset_attr_phase_change
		H5Pset_char_encoding
		H5Pset_chunk_cache
		H5Pset_dxpl_mpio_chunk_opt
		H5Pset_dxpl_mpio_chunk_opt_num
		H5Pset_dxpl_mpio_chunk_opt_ratio
		H5Pset_dxpl_mpio_collective_opt
		H5Pset_mdc_config
		H5Pset_type_conv_cb

		H5Sdecode
		H5Sencode
		H5Sextent_equal
		H5Tcompiler_conv
		H5Tdecode
		H5Tencode