
HDF5 User's Guide

HDF5 Release 1.8.15

May 2015

|

?????? copyright info ??????

??????? update status page ???????

Table of Contents

2. The HDF5 Data Model and File Structure	7
3. The HDF5 Library and Programming Model	51
4. The HDF5 File	109
4.1. Introduction to HDF5 Files	109
4.1.1. File Access Modes	109
4.1.1. File Creation and File Access Properties	110
4.1.1. Low-level File Drivers	111
4.1. Programming Model	112
4.1.1. Creating a New File	112
4.1.1. Opening an Existing File	113
4.1.1. Closing a File	114
4.1. Using h5dump to View a File	114
4.1. File Function Summaries	115
4.1. Creating or Opening an HDF5 File	121
4.1. Closing an HDF5 File	122
4.1. File Property Lists	123
4.1.1. Creating a Property List	123
4.1.1. File Creation Properties	124
4.1.1. File Access Properties	125
4.1. Alternate File Storage Layouts and Low-level File Drivers	126
4.1.1. Identifying the Previously-used File Driver	130
4.1.1. The POSIX (aka SEC2) Driver	131
4.1.1. The Direct Driver	131
4.1.1. The Log Driver	132
4.1.1. The Windows Driver	133
4.1.1. The STDIO Driver	133
4.1.1. The Memory (aka Core) Driver	134
4.1.1. The Family Driver	135
4.1.1. The Multi Driver	136
4.1.1. The Split Driver	137
4.1.1. The Parallel Driver	138
4.1. Code Examples for Opening and Closing Files	139
4.1.1. Example Using the H5F_ACC_TRUNC Flag	139
4.1.1. Example with the File Creation Property List	140
4.1.1. Example with File Access Property List	141
4.1. Working with Multiple HDF5 Files	141
5. HDF5 Groups	152
6. HDF5 Datasets	247
7. HDF5 Datatypes	414

8. HDF5 Dataspaces and Partial I/O	659
9. HDF5 Attributes	757
10. HDF5 Error Handling	799
11. Properties and Property Lists in HDF5	839
12. Additional Resources	929

```

<!doctype HTML public "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
<head>
<title>Chapter 1: The HDF5 Data Model and File Structure</title>

<!--(Meta)=====-->

<!--(Links)=====-->

<!--( Begin styles definition )=====-->
<link href="ed_styles/NewUGelect.css" rel="stylesheet" type="text/css">
<!--( End styles definition )=====-->

</head>

<body>

<!-- #BeginLibraryItem "/ed_libs/styles_UG.lbi" -->
<!--
* * * * *
* Copyright by The HDF Group. *
* Copyright by the Board of Trustees of the University of Illinois. *
* All rights reserved. *
*
* This file is part of HDF5. The full HDF5 copyright notice, including *
* terms governing use, modification, and redistribution, is contained in *
* the files COPYING and Copyright.html. COPYING can be found at the root *
* of the source code distribution tree; Copyright.html can be found at the *
* root level of an installed copy of the electronic HDF5 document set and *
* is linked from the top-level documents page. It can also be found at *
* http://www.hdfgroup.org/HDF5/doc/Copyright.html. If you do not have *

```

* access to either file, you may request a copy from help@hdfgroup.org. *

-->

<!-- #EndLibraryItem --><!--(TOC)=====

->

<SCRIPT language="JavaScript">

<!--

document.writeln ('\

<table x-use-null-cells\

align="right"\

width=240\

cellspacing="0"\

class="tocTable">\

<tr valign="top"> \

<td class="tocTableHeaderCell" colspan="2"> \

Chapter Contents</td>\

</tr>\

-->

<!-- Table Version 3 -->\

<!--

<tr valign="top"> \

<td class="tocTableContentCell2"> \

1.</td>\

<td class="tocTableContentCell3">\

Introduction</td> \

</tr>\

<tr valign="top"> \

<td class="tocTableContentCell2"> \

2.</td>\

<td class="tocTableContentCell3">\

The Abstract Data Model</td>\

```

</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#SModel">3.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#SModel">The HDF5 Storage Model</a></td> \
</tr>\
\
<tr valign="top"> \
  <td class="tocTableContentCell"> \
-->
<!-- editingComment -- "tocTableContentCell" and "tocTableContentCell4" \
-->\
<!-- are the table-closing cell class.\
  <td class="tocTableContentCell2"> \
-->\
<!--
  <a href="#Structure">4.</a></td>\
  <td class="tocTableContentCell4">\
<a href="#Structure">The Structure of an HDF5 File</a>\
-->
<!-- editingComment -- This section not currently complete or validated.\
</tr><tr valign="top"> \
  <td class="tocTableContentCell"> \
  <a href="#Appendix">10</a></td>\
  <td class="tocTableContentCell4"><a href="#Appendix">Appendix</a></td>\
-->\
<!--
  </td></tr>\
</table>\
')
```

-->

</SCRIPT>

<!--(End TOC)=====-->

<div align="center">

2. The HDF5 Data Model and File Structure

</div>

<!-- editingComment

[[[]]]

-->

<!-- HEADER LEFT " " -->

<!-- HEADER RIGHT " " -->

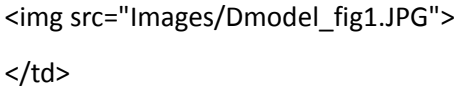
<!-- HEADER LEFT "HDF5 User's Guide" -->

<!-- HEADER RIGHT "HDF5 Data Model" -->

<h3>1.1. Introduction</h3>

<p>The Hierarchical Data Format (HDF) implements a model for managing and storing data. The model includes an abstract data model and an abstract storage model (the data format), and libraries to implement the abstract model and to map the storage model to different storage mechanisms. The HDF5 library provides a programming interface to a concrete implementation of the abstract models. The library also

implements a model of data transfer, i.e., efficient movement of data from one stored representation to another stored representation. The figure below illustrates the relationships between the models and implementations. This chapter explains these models in detail.


<hr/>
Figure 1. HDF5 models and implementations <hr/>

The **Abstract Data Model** is a conceptual model of data, data types, and data organization. The abstract data model is independent of storage medium or programming environment. The **Storage Model** is a standard representation for the objects of the abstract data model. The [HDF5 File Format Specification](http://www.hdfgroup.org/HDF5/format/H5format.html) defines the storage model.

The **Programming Model** is a model of the computing environment and includes platforms from small single systems to large multiprocessors

and clusters. The programming model manipulates (instantiates, populates, and retrieves) objects from the abstract data model.

The **Library** is the concrete implementation of the programming model. The Library exports the HDF5 APIs as its interface.

In addition to implementing the objects of the abstract data model, the Library manages data transfers from one stored form to another.

Data transfer examples include reading from disk to memory and writing from memory to disk.

Stored Data is the concrete implementation of the storage model. The storage model is mapped to several storage mechanisms including single disk files, multiple files (family of files), and memory representations.

The HDF5 Library is a C module that implements the programming model and abstract data model. The HDF5 Library calls the operating system or other storage management software (e.g., the MPI/IO Library) to store and retrieve persistent data. The HDF5 Library may also link to other software such as filters for compression. The HDF5 Library is linked to an application program which may be written in C, C++, Fortran, or Java. The application program implements problem specific algorithms and data structures and calls the HDF5 Library to store and retrieve data. The figure below shows the dependencies of these modules.

```
<table width = 600 cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
```

```

    
  </td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 2. The library, the application
    program, and other modules</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

<br />
<p>It is important to realize that each of the software components manages
data using models and data structures that are appropriate to the
component. When data is passed between layers (during storage or
retrieval), it is transformed from one representation to another. The
figure below <!-- Figure 3 --> suggests some of the kinds of data
structures used in the different layers.</p>

<p>The <em>Application Program</em> uses data structures that represent
the problem and algorithms including variables, tables, arrays, and
meshes among other data structures. Depending on its design and function,
an application may have quite a few different kinds of data structures
and different numbers and sizes of objects.</p>

<p>The HDF5 Library implements the objects of the HDF5 abstract
data model. Some of these objects include groups, datasets, and
attributes. The application program maps the application data

```


structures to a hierarchy of HDF5 objects. Each application will create a mapping best suited to its purposes. </p>

<!-- editingComment

For suggestions and examples, see ???.

[[[Do we have such a document?]]]

-->

<p>The objects of the HDF5 abstract data model are mapped to the objects of the HDF5 storage model, and stored in a storage medium.

<!-- editingComment

(Section ?? below)

-->

The stored objects include header blocks, free lists, data blocks, B-trees, and other objects. Each group or dataset is stored as one or more header and data blocks. See the

<cite>HDF5 File Format Specification</cite> for more information on how these objects are organized.

The HDF5 Library can also use other libraries and modules such as compression.</p>

<!-- NEW PAGE -->

<table width = 600 cellpadding="0" align="center">

<tr valign="top">

<td align="center">

<hr color="green" size="3"/>

</td>

</tr>

<tr><td><hr color="green" size="1" /></td></tr>

```

<tr valign="top">
  <td align="left" >
    <b>Figure 3. Data structures in different layers</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

The important point to note is that there is not necessarily any simple correspondence between the objects of the application program, the abstract data model, and those of the *Format Specification*. The organization of the data of application program, and how it is mapped to the HDF5 abstract data model is up to the application developer. The application program only needs to deal with the library and the abstract data model. Most applications need not consider any details of the [*HDF5 File Format Specification*](../H5.format.html) or the details of how objects of abstract data model are translated to and from storage.

```

<SCRIPT language="JavaScript">
<!--
document.writeln ("
<div align="right">
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>

```


<h3 class="pagebefore">1.2. The Abstract Data Model</h3>

<!-- editingComment

[[Note: In this section some of the figures
are included twice. These are alternative versions of the same diagram,
included for comparison and selection.]]

-->

<p>The abstract data model (ADM) defines concepts for defining and
describing complex data stored in files. The ADM is a very general model
which is designed to conceptually cover many specific models. Many different
kinds of data can be mapped to objects of the ADM, and therefore stored and
retrieved using HDF5. The ADM is not, however, a model of any particular problem or
application domain. Users need to map their data to the concepts of the ADM.</p>

<p>The key concepts include:</p>

File - a contiguous string of bytes in a computer
store (memory, disk, etc.), and

the bytes represent zero or more objects of the model

Group - a collection of objects (including groups)

Dataset - a multidimensional array of data elements
with attributes and other metadata

Dataspace - a description of the dimensions of a
multidimensional array

Datatype - a description of a specific class of data
element including its storage layout as a pattern of bits

- Attribute - a named data value associated with a group,
dataset, or named datatype
- Property List - a collection of parameters (some permanent
and some transient) controlling options in the library
- Link - the way objects are connected

<p>These key concepts are described in more detail below.</p>

<h4>1.2.1. File</h4>

<p>Abstractly, an HDF5 file is a container for an organized
collection of objects.

The objects are groups, datasets, and other objects as defined below.

The objects are organized as a rooted, directed graph. Every HDF5 file has at
least one object, the root group. See the figure below. All objects are
members of the
root group or descendents of the root group.</p>

<!-- NEW PAGE -->

<table width = 600 cellpadding="0" align="center">

<tr valign="top">

<td align="center">

<hr color="green" size="3"/>

</tr>

<tr>

<td>

<table align="center" border="1">

<tr>

<td valign="top" align="center"><code>File</code></td>

```

        </tr>
<tr>
  <td valign="top" align="left">
    <code>
      superblock_vers:int<br />
      global_freelist_vers:int<br />
      symtable_vers:int<br />
      sharedobjectheader_vers:int<br />
      userblock:size_t<br />
      sizeof_addr:size_t<br />
      sizeof_size:size_t<br />
      symtable_tree_rank:int<br />
      symtable_node_size:int<br />
      btree_istore_size:int
    </code>
  </td>
</tr>

<tr>
  <td align="left">&nbsp;  </td>
</tr>
</table>
</td>
</tr>
<tr>
  <td valign="top" align="center"></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 4. The HDF5 file</b>
    <hr color="green" size="3"/></td>

```

```

    </tr>
</table>
<br />

```

HDF5 objects have a unique identity *within a single HDF5 file* and can be accessed only by its names within the hierarchy of the file. HDF5 objects in different files do not necessarily have unique identities, and it is not possible to access a permanent HDF5 object except through a file. See the section ["The Structure of an HDF5 File"](#Structure) below for an explanation of the structure of the HDF5 file.

When the file is created, the *file creation properties* specify settings for the file. The file creation properties include version information and parameters of global data structures. When the file is opened, the *file access properties* specify settings for the current access to the file. File access properties include parameters for storage drivers

```
<!-- editingComment
```

```

    <span class="editingComment">(see section ?? below)</span>

```

```
-->
```

and parameters for caching and garbage collection.

The file creation properties are set permanently for the life of the file, and

the file access properties can be changed by closing and reopening the file.

```
<!-- editingComment
```

```

    <span class="editingComment">

```

See PPP for more information about Property Lists and properties.

```

    </span>

```

```
-->
```

An HDF5 file can be "mounted" as part of another HDF5 file.

This is analogous to Unix file system mounts. The root of the mounted file is attached to a group in the mounting file, and all the contents can be accessed as if the mounted file were part of the mounting file.

```
<!-- editingComment
```

```
<span class="editingComment">
```

```
See XXX for an explanation of mounted files.
```

```
</span>
```

```
-->
```

```
<h4>1.2.2. Group</h4>
```

```
<p>An HDF5 group is analogous to a file system directory. Abstractly,
a group contains zero or more objects, and every object must be a member of at
least one group. The root group is a special case; it may not be a member
of any group.</p>
```

```
<p>Group membership is actually implemented via link objects. See the
figure below. A link object is owned by a group and points to a
<em>named object</em>. Each link has a <em>name</em>, and each link
points to exactly one object. Each named object has at least one and
possibly many links to it.</p>
```

```
<table width = 600 cellpadding="0" align="center">
```

```
<tr valign="top">
```

```
<td align="center">
```

```
<hr color="green" size="3"/>
```

```

```

```
</td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```

<tr valign="top">
  <td align="left" >
    <b>Figure 5. Group membership via link objects</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
<!-- NEW PAGE -->

<p>There are three classes of named objects: group, dataset, and named
datatype. See the figure below. Each of these objects is the member of
at least one group, and this means there is at least one link to it.</p>

```

```

<table width = 600 cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 6. Classes of named objects</b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

```

<h4>1.2.3. Dataset</h4>

```


An HDF5 dataset is a multidimensional (rectangular) array of data elements. See the figure below. The shape of the array (number of dimensions, size of each dimension) is described by the dataspace object (described in the next section below).

A data element is a single unit of data which may be a number, a character, an array of numbers or characters, or a record of heterogeneous data elements. A data element is a set of bits. The layout of the bits is described by the datatype (see below).

The dataspace and datatype are set when the dataset is created, and they cannot be changed for the life of the dataset. The *dataset creation properties* are set when the dataset is created. The dataset creation properties include the fill value and storage properties such as chunking and compression. These properties cannot be changed after the dataset is created.

The dataset object manages the storage and access to the data. While the data is conceptually a contiguous rectangular array, it is physically stored and transferred in different ways depending on the storage properties and the storage mechanism used. The actual storage may be a set of compressed chunks, and the access may be through different storage mechanisms and caches. The dataset maps between the conceptual array of elements and the actual stored data.

<!-- NEW PAGE -->

<table width = 600 cellpadding="0" align="center">

```

<tr valign="top">
  <td align="center">
    <hr color="green" size="3"/>
    
  </td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 7. The dataset</b>
    <hr color="green" size="3"/></td>
</tr>
</table>
<br />

```

<h4>1.2.4. Dataspace</h4>

The HDF5 dataspace describes the layout of the elements of a multidimensional array. Conceptually, the array is a hyper-rectangle with one to 32 dimensions. HDF5 dataspaces can be extendable. Therefore, each dimension has a current size and a maximum size, and the maximum may be unlimited. The dataspace describes this hyper-rectangle: it is a list of dimensions with the current and maximum (or unlimited) sizes. See the figure below.

```

<!-- NEW PAGE -->
<table width = 600 cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">

```

```
<hr color="green" size="3"/>
<table border="1">
  <tr><td align="center">
    <code>Dataspace</code>
  </td></tr>
  <tr><td align="left">
    <code>
      &nbsp;&nbsp;&nbsp;&rank:int<br />
      &nbsp;&nbsp;&current_size:hsize_t[ rank ]&nbsp;&nbsp;&<br />
      &nbsp;&nbsp;&maximum_size:hsize_t[ rank ]
    </code>
  </td></tr>
  <tr><td align="left">&nbsp;&</td></tr>
</table>

&nbsp;&;

</td>

</tr>

<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 8. The dataspace</b><hr color="green" size="3"/></td>
  </tr>
</table>

<br />
```

Dataspace objects are also used to describe hyperslab selections from a dataset. Any subset of the elements of a dataset can be selected for read or write by specifying a set of hyperslabs. A non-rectangular region can be selected by the union of several (rectangular) dataspace.

See SSS for more information about data selection and hyperslabs.

-->

<h4>1.2.5. Datatype</h4>

<p>The HDF5 datatype object describes the layout of a single data element. A data element is a single element of the array; it may be a single number, a character, an array of numbers or carriers, or other data. The datatype object describes the storage layout of this data. </p>

<p>Data types are categorized into 11 classes of datatype. Each class is interpreted according to a set of rules and has a specific set of properties to describe its storage. For instance, floating point numbers have exponent position and sizes which are interpreted according to appropriate standards for number representation. Thus, the datatype class tells what the element means, and the datatype describes how it is stored.</p>

<p>The figure below <!-- formerly Figure 9 --> shows the classification of datatypes. Atomic datatypes are indivisible. each may be a single object; a number, a string, or some other objects. Composite datatypes are composed of multiple elements of atomic datatypes. In addition to the standard types, users can define additional datatypes such as a 24-bit integer or a 16-bit float.</p>

<p>A dataset or attribute has a single datatype object associated with it. See Figure 7 above. The datatype object may be used in the definition of several objects, but by default, a copy of the datatype object will be private to the dataset. </p>

Optionally, a datatype object can be stored in the HDF5 file. The datatype is linked into a group, and therefore given a name. A named datatype can be opened and used in any way that a datatype object can be used.

The details of datatypes, their properties, and how they are used are explained in the [HDF5 Datatypes](11_Datatypes.html) chapter.

<!-- NEW PAGE -->

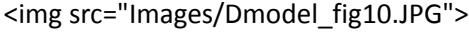
<div data-bbox="233 791 422 827" data-label="Text"> <tr> <td style="text-align: center;"> <div data-bbox="233 846 477 882" data-label="Text"> <td style="text-align: center;"> <div data-bbox="233 900 586 936" data-label="Text"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/> <div data-bbox="233 955 693 993" data-label="Text">  <div data-bbox="233 1012 310 1050" data-label="Text"> </div> </div> </div></td> </div></td></tr> </div>	<div data-bbox="233 846 477 882" data-label="Text"> <td style="text-align: center;"> <div data-bbox="233 900 586 936" data-label="Text"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/> <div data-bbox="233 955 693 993" data-label="Text">  <div data-bbox="233 1012 310 1050" data-label="Text"> </div> </div> </div></td> </div>	<div data-bbox="233 900 586 936" data-label="Text"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/> <div data-bbox="233 955 693 993" data-label="Text">  <div data-bbox="233 1012 310 1050" data-label="Text"> </div> </div> </div>	<div data-bbox="207 1125 794 1161" data-label="Text"> <tr> <td style="text-align: center;"> <hr style="border: 1px solid green; width: 100px; margin: 0 auto;"/> </td> </tr> </div> <div data-bbox="207 1180 422 1215" data-label="Text"> <tr> <td style="vertical-align: top;"> <div data-bbox="233 1236 444 1272" data-label="Text"> <td style="vertical-align: top;"> <div data-bbox="233 1291 730 1329" data-label="Text"> <p>Figure 9. Datatype classifications</p> <div data-bbox="233 1348 651 1383" data-label="Text"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/> <div data-bbox="233 1402 305 1440" data-label="Text"> </div> </div> </div></td> </div></td></tr> </div>	<hr style="border: 1px solid green; width: 100px; margin: 0 auto;"/>	<div data-bbox="233 1236 444 1272" data-label="Text"> <td style="vertical-align: top;"> <div data-bbox="233 1291 730 1329" data-label="Text"> <p>Figure 9. Datatype classifications</p> <div data-bbox="233 1348 651 1383" data-label="Text"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/> <div data-bbox="233 1402 305 1440" data-label="Text"> </div> </div> </div></td> </div>	<div data-bbox="233 1291 730 1329" data-label="Text"> <p>Figure 9. Datatype classifications</p> <div data-bbox="233 1348 651 1383" data-label="Text"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/> <div data-bbox="233 1402 305 1440" data-label="Text"> </div> </div> </div>
<div data-bbox="233 846 477 882" data-label="Text"> <td style="text-align: center;"> <div data-bbox="233 900 586 936" data-label="Text"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/> <div data-bbox="233 955 693 993" data-label="Text">  <div data-bbox="233 1012 310 1050" data-label="Text"> </div> </div> </div></td> </div>	<div data-bbox="233 900 586 936" data-label="Text"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/> <div data-bbox="233 955 693 993" data-label="Text">  <div data-bbox="233 1012 310 1050" data-label="Text"> </div> </div> </div>					
<hr style="border: 1px solid green; width: 100px; margin: 0 auto;"/>						
<div data-bbox="233 1236 444 1272" data-label="Text"> <td style="vertical-align: top;"> <div data-bbox="233 1291 730 1329" data-label="Text"> <p>Figure 9. Datatype classifications</p> <div data-bbox="233 1348 651 1383" data-label="Text"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/> <div data-bbox="233 1402 305 1440" data-label="Text"> </div> </div> </div></td> </div>	<div data-bbox="233 1291 730 1329" data-label="Text"> <p>Figure 9. Datatype classifications</p> <div data-bbox="233 1348 651 1383" data-label="Text"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/> <div data-bbox="233 1402 305 1440" data-label="Text"> </div> </div> </div>					

1.2.6. Attribute

Any HDF5 named data object (group, dataset, or named datatype) may have zero or more user defined attributes. Attributes are used to document

the object. The attributes of an object are stored with the object.

An HDF5 attribute has a name and data. The data portion is similar in structure to a dataset: a dataspace defines the layout of an array of data elements, and a datatype defines the storage layout and interpretation of the elements See the figure below

<hr/> 
<hr/>
Figure 10. Attribute data elements <hr/>

In fact, an attribute is very similar to a dataset with the following limitations:

- An attribute can only be accessed via the object
- Attribute names are significant only within the object

- An attribute should be a small object
 - The data of an attribute must be read or written in a single access
(partial reading or writing is not allowed)
 - Attributes do not have attributes
-

<p>Note that the value of an attribute can be an object reference.</p>

A shared attribute or an attribute that is a large array can be implemented as a reference to a dataset.</p>

<!-- NEW PAGE -->

<p>The name, dataspace, and datatype of an attribute are specified when it is created and cannot be changed over the life of the attribute. An attribute can be opened by name, by index, or by iterating through all the attributes of the object.</p>

<h4>1.2.7. Property List</h4>

<p>HDF5 has a generic property list object. Each list is a collection of name-value pairs. Each class of property list has a specific set of properties.

Each property has an implicit name, a datatype, and a value. See the figure below. <!-- formerly Figure 11 -->

A property list object is created and used in ways similar to the other objects of the HDF5 library.</p>

<p>Property Lists are attached to the object in the library, they can be used by any part of the library. Some properties are permanent (e.g., the chunking strategy for a dataset), others are transient (e.g., buffer sizes for data transfer). A common use of a Property List is to pass parameters from the calling program to a VFL driver or a module of the pipeline.</p>

<p>Property lists are conceptually similar to attributes. Property lists are information relevant to the behavior of the library while attributes are relevant to the user’s data and application.</p>

<code><table width = 600 cellspacing="0" align="center"></code>

```
<tr valign="top">
```

```
<td align="center">
```

<div style="display: flex; justify-content: space-between; align-items: center;"> 1. Introduction 2. Methodology </div>	
<p>1.1 Background</p> <p>1.2 Objectives</p> <p>1.3 Scope</p>	<p>2.1 Data Collection</p> <p>2.2 Analysis Techniques</p> <p>2.3 Results</p>

|

 |


```
<tr><td align="center">
```

```
<code>Property List</code>
```

```
<tr><td align="left" valign="middle">
```

```
<code>
```

 class:H5P_class_t

</code>

</td></tr>

| <tr><td align="left" valign="middle"> |

```
<code>
```

```
   create(class)<br />
```

```
   get_class()
```

</code>

</td>


```

    </tr>
<tr>
  <td valign="top" align="center">
    </td>
  </tr>
<tr>
  <td align="center">
    <table border="1">
      <tr><td align="center">
        <code>Property</code>
      </td></tr>
      <tr><td align="left" valign="middle">
        <code>
          &nbsp;name:string<br />
          &nbsp;value:H5TDatatype&nbsp;
        </code>
      </td></tr>
      <tr><td align="left" valign="middle">
        <code>
          &nbsp;<br />&nbsp;
        </code>
      </td></tr>
    </table>&nbsp;
  </td>
</tr>
</table>
</td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >

```

```

<b>Figure 11. The property list</b>
<hr color="green" size="3"/></span>
</td>
</tr>
</table>
<br />

<!-- NEW PAGE -->

<p>Property lists are used to control optional behavior for file creation,
file access, dataset creation, dataset transfer (read, write), and file
mounting. Some property list classes are shown in the table below.
<!-- Table 1--> Details of the different property lists are explained in
the relevant sections of this document.</p>

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="3" align="left" valign="bottom">
      <b>Table 1. Property list classes and their usage
      </b></td>
    </tr>
    <tr><td colspan="3"><hr color="green" size="3" /></td></tr>
    <tr valign="top">
      <td width="34%">
        <b>Property List Class</b></td>
      <td width="33%">
        <b>Used</b></td>
      <td width="33%">
        <b>Examples</b></td>
    </tr>
    <tr><td colspan="3"><hr color="green" size="1" /></td></tr>
    <tr valign="top">

```

<code>H5P_FILE_CREATE</code>	Properties for file creation.	Set size of user block.
------------------------------	-------------------------------	-------------------------

<code>H5P_FILE_ACCESS</code>	Properties for file access.	Set parameters for VFL driver. An example is MPI I/O.
------------------------------	-----------------------------	-------------------------------------------------------

<code>H5P_DATASET_CREATE</code>	Properties for dataset creation.	Set chunking, compression, or fill value.
---------------------------------	----------------------------------	-------------------------------------------

<code>H5P_DATASET_XFER</code>	Properties for raw data transfer (read and write).	Tune buffer sizes or memory management.
-------------------------------	----------------------------------------------------	-----------------------------------------

<code>H5P_FILE_MOUNT</code>	Properties for file mounting.	
-----------------------------	-------------------------------	--

```

        </tr>
        <tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="SModel">
<div align="right">
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>
<br />
<h4>1.2.8. Link</h4>
<p>This section is under construction.</p>

<br />

```

```

<!-- NEW PAGE -->
<a name="SModel">
<h3 class="pagebefore">1.3. The HDF5 Storage Model</h3>
</a>

```

```

<h4>1.3.1. The Abstract Storage Model: the HDF5 Format Specification</h4>

```

```

<p>The <a name="SupScript1" href=" ../H5.format.html">
<em>HDF5 File Format Specification</em></a>
<!-- editingComment
<span class="editingComment">[ [
[cite it]
] ] </span>
-->
defines how HDF5 objects
and data are mapped to a <em>linear address space</em>. The address space is
assumed to be a contiguous array of bytes stored on some random access
medium.<a href="#FootNote"><sup><font size="-1">1</font></sup></a>
The format defines the standard for how the objects of the
abstract data model are mapped to linear addresses. The stored
representation is self-describing in the sense that the format defines all
the information necessary to read and reconstruct the original
objects of the abstract data model.</p>

```

<p>The HDF5 File Format Specification is organized in three parts:

```

<ol>
<li><strong>Level 0</strong>: File signature and super block</li>
<li><strong>Level 1</strong>: File infrastructure</li>
<ol type="a">
<li><strong>Level 1A</strong>: B-link trees and B-tree nodes</li>
<li><strong>Level 1B</strong>: Group</li>
<li><strong>Level 1C</strong>: Group entry</li>
<li><strong>Level 1D</strong>: Local heaps</li>
<li><strong>Level 1E</strong>: Global heap</li>
<li><strong>Level 1F</strong>: Free-space index</li>
</ol>
<li><strong>Level 2</strong>: Data object</li>

```

```
<ol type="a">
  <li><strong>Level 2A</strong>: Data object headers</li>
  <li><strong>Level 2B</strong>: Shared data object headers</li>
  <li><strong>Level 2C</strong>: Data object data storage</li>
</ol>
</ol>
```

The Level 0 specification defines the header block for the file. Header block elements include a signature, version information, key parameters of the file layout (such as which VFL file drivers are needed), and pointers to the rest of the file. Level 1 defines the data structures used throughout the file: the B-trees, heaps, and groups. Level 2 defines the data structure for storing the data objects and data. In all cases, the data structures are completely specified so that every bit in the file can be faithfully interpreted.</p>

It is important to realize that the structures defined in the HDF5 file format are not the same as the abstract data model: the object headers, heaps, and B-trees of the file specification are not represented in the abstract data model. The format defines a number of objects for managing the storage including header blocks, B-trees, and heaps. The HDF5 File Format Specification defines how the abstract objects (for example, groups and datasets) are represented as headers, B-tree blocks, and other elements.</p>

The HDF5 Library implements operations to write HDF5 objects to the linear format and to read from the linear format to create HDF5 objects. It is important to realize that a single HDF5 abstract object is usually stored as several objects. A dataset, for example, might be stored in a header and in one or more data blocks, and these objects

might not be contiguous on the hard disk.</p>

<!-- NEW PAGE -->

<h4>1.3.2. Concrete Storage Model</h4>

<p>The HDF5 file format defines an abstract linear address space. This can be implemented in different storage media such as a single file or multiple files on disk or in memory.

The HDF5 Library defines an open interface called the Virtual File Layer (VFL). The VFL allows different concrete storage models to be selected. </p>

<!-- editingComment

See Ch. XXX and the VFL document [cite].

-->

<p>The VFL defines an abstract model, an API for random access storage, and an API to plug in alternative VFL driver modules. The model defines the operations that the VFL driver must and may support, and the plug-in API enables the HDF5 Library to recognize the driver and pass it control and data.</p>

<p>A number of VFL drivers have been defined in the HDF5 Library. Some work with a single file, and some work with multiple files split in various ways. Some work in serial computing environments, and some work in parallel computing environments. Most work with disk copies of HDF5 files, but one works with a memory copy. These drivers are listed in the table in the “Alternate File Storage Layouts and Low-level File Drivers” section in “The File” chapter. </p>

```
<!--
<table width = 600 cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 12. Conceptual hierarchy of VFL drivers</b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />
```

9.28.2011. I removed the figure above. Drivers have changed a lot since the figure was created. MEE -->

<p>Each driver isolates the details of reading and writing storage so that the rest of the HDF5 Library and user program can be almost the same for different storage methods. The exception to this rule is that some VFL drivers need information from the calling application. This information is passed using property lists. For example, the Parallel driver requires certain control information that must be provided by the application.</p>

<SCRIPT language="JavaScript">


```
<!--
document.writeln ("
<div align="right">
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>
<br />
<a name="Structure">
<!-- NEW PAGE -->
<h3 class="pagebefore">1.4. The Structure of an HDF5 File</h3>
</a>
```

<h4>1.4.1. Overall File Structure</h4>

<p>An HDF5 file is organized as a rooted, directed graph. Named data objects are the nodes of the graph, and links are the directed arcs. Each arc of the graph has a name, and the root group has the name "root". Objects are created and then inserted into the graph with the link operation which creates a named link from a group to the object. For example, the figure below <!-- formerly Figure 38 -->illustrates the structure of an HDF5 file when one dataset is created. An object can be the target of more than one link. The names on the links must be unique within each group, but there may be many links with the same name in different groups. Link names are unambiguous: some ancestor will have a different name, or they are the same object. The graph is navigated with path names similar to Unix file systems.

<!-- editingComment

[[

[cite something]

]]

-->

An object can be opened with a full path starting at the root group or with a relative path and a starting node (group). Note that all paths are relative to a single HDF5 file. In this sense, an HDF5 file is analogous to a single Unix file system.

^{2}</p>

```
<table width = 300 cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <br />
      a) Newly created file: one group, <code></code><br />
      <br />
      b) Create a dataset called <code>/dset1</code><br />
      (<code>H5create(..., &ldquo;/dset2&rdquo;, ...</code>)<br /><br />
    </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 12. An HDF5 file with one dataset
      <!-- formerly Figure 38 --></b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
```


<p>It is important to note that, just like the Unix file system, HDF5 objects do not have names. The names are associated with paths. An object has a unique (within the file) object ID, but a single object may have many names because there may be many paths to the same object. An object can be renamed (moved to another group) by adding and deleting links. In this case, the object itself never moves. For that matter, membership in a group has no implication for the physical location of the stored object.</p>

<!-- NEW PAGE -->

<p>Deleting a link to an object does not necessarily delete the object. The object remains available as long as there is at least one link to it. After all the links to an object are deleted, it can no longer be opened although the storage may or may not be reclaimed.^{3}</p>

<p>It is important to realize that the linking mechanism can be used to construct very complex graphs of objects. For example, it is possible for an object to be shared between several groups and even to have more than one name in the same group. It is also possible for a group to be a member of itself or to be in a “cycle” in the graph. An example of a cycle is where a child is the parent of one of its own ancestors. </p>

<!-- move the following paragraph to the Links chapter when it is written:

<p>HDF5 also has soft links similar to Unix soft links.

A soft link is an object that contains a name and a path name for the target object. The soft link can be followed to open the target of the link just like a regular (hard) link. Unlike hard links, the target of a soft

link has no count of the soft link to it. The reference count of an object is the number of hard Links (which must be ≥ 1). A second difference is that the hard link cannot be created if the target object does not exist, and always points to the same object. A Soft Link can be created with any path name, whether or not the object exists. Therefore, it may or may not be possible to follow a Soft Link, or the target object may change from one access to another access of the same Soft Link.

1.4.2. HDF5 Path Names and Navigation

The structure of the file constitutes the name space for the objects in the file. A path name is a string of components separated by `"/"`. Each component is the name of a link or the special character `"."` for the current group. Link names (components) can be any string of ASCII characters not containing `"/"` (except the string `"."` which is reserved). However, users are advised to avoid the use of punctuation and non-printing characters because they may create problems for other software. The figure below (formerly Figure 39) gives a BNF grammar for HDF5 path names.

```
<table width = 600 cellpadding="0" align="center">
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<hr color="green" size="3"/>
```

```
<pre>
```

```
PathName ::= AbsolutePathName | RelativePathName
```

```
Separator ::= "/" [ "/" ]*
```

```
AbsolutePathName ::= Separator [ RelativePathName ]
```

```
RelativePathName ::= Component [ Separator RelativePathName ]*
```

```

Component ::= "." | Name
Name ::= Character+ - {"."}
Character ::= {<em>c</em>: <em>c</em> in {{ <em>legal ASCII characters</em> } - {'/'}}

</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 13. A BNF grammar for path names
    <!-- formerly Figure 39--></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

An object can always be addressed by a *full or absolute path* which would start at the root group. As already noted, a given object can have more than one full path name. An object can also be addressed by a relative path which would start at a group and include the path to the object.

The structure of an HDF5 file is “self-describing.” This means that it is possible to navigate the file to discover all the objects in the file. Basically, the structure is traversed as a graph starting at one node and recursively visiting the nodes of the graph.

move the following paragraph to the Links chapter when it is written:

The members of a group can be discovered with the H5Giterate function, and a description of the object can be retrieved with the H5Gget_obj_info function. In this way, all the members of a given group can be determined, and each can be opened to retrieve a description, or the data and

```

attributes of the object.</p> -->
<!-- editingComment
<span class="editingComment">
See ??? for more information about navigating and discovering the contents
of and HDF5 file.
</span>
-->

```

<h4>1.4.3. Examples of HDF5 File Structures</h4>

```

<p>The figure below <!-- formerly Figure 40 -->shows some possible HDF5
file structures with groups and datasets. Part a of the figure shows the
structure of a file with three groups.
Part b of the figure shows a dataset created in &ldquo;/group1&rdquo;. Part
c shows the structure after a dataset called dset2 has been added to the
root group. Part d the structure after another group and dataset have been
added.</p>

```

```

<!-- NEW PAGE -->

```

```

<table width = 600 cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>

      <table width="100%">
        <tr valign="top" align="left">
          <td width="48%">a) Three groups; two are members of the root
group,<br />
          <code>/group1</code> and <code>/group2</code></td>

```

```

        <td width="4%">&nbsp;&nbsp;&nbsp;</td>
        <td width="48%">b) Create a dataset in <code>/group1</code>:
        <br />
        <code>/group1/dset1</code></td>
    </tr>

    <tr valign="middle" align="center">
        <td width="48%"></td>
        <td width="4%">&nbsp;&nbsp;&nbsp;</td>
        <td width="48%"></td>
    </tr>

    <tr valign="top" align="left">
        <td width="48%">c) Another dataset, a member of the root
        group: <br />
        <code>/dset2</code></td>
        <td width="4%">&nbsp;&nbsp;&nbsp;</td>
        <td width="48%">d) And another group and dataset, reusing
        object names: <br />
        <code>/group2/group2/dset2</code></td>
    </tr>

    <tr valign="middle" align="center">
        <td width="48%"></td>
        <td width="4%">&nbsp;&nbsp;&nbsp;</td>
        <td width="48%"></td>
    </tr>
</table>
</td>
</tr>

```

```

<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 14.
    <!-- formerly Figure 40: -->
    Examples of HDF5 file structures with groups and datasets</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

<!-- FOR USE WITH ELECTRONIC VERSION ----->
<br /><br /><br />
<!-- FOR USE WITH ELECTRONIC VERSION ----->

<p>&nbsp;</p>
<a name="FootNote"><hr width="200px" align="left"></a>

<font size="-1"><sup><a href="#SupScript1">1</a></sup>HDF5 requires random
access to the linear address space. For this reason it is not well suited for
some data media such as streams.</font>
<br />
<font size="-1"><sup><a href="#SupScript2">2</a></sup>It could be said that
HDF5 extends the organizing concepts of a file system to the internal
structure of a single file.</font>

<br />
<font size="-1"><sup><a href="#SupScript3">3</a></sup>As of HDF5-1.4, the
storage used for an object is reclaimed, even if all links
are deleted.</font>

```


</body>

</html>

I

```

<!doctype HTML public "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
<head>
<title>Chapter 2: The HDF5 Library and Programming Model</title>

<!--(Meta)=====-->

<!--(Links)=====-->

<!--( Begin styles definition )=====-->
<link href="ed_styles/NewUGelect.css" rel="stylesheet" type="text/css">
<!--( End styles definition )=====-->

</head>

<body>

<!-- #BeginLibraryItem "/ed_libs/styles_UG.lbi" -->
<!--
* * * * *
* Copyright by The HDF Group. *
* Copyright by the Board of Trustees of the University of Illinois. *
* All rights reserved. *
* *
* This file is part of HDF5. The full HDF5 copyright notice, including *
* terms governing use, modification, and redistribution, is contained in *
* the files COPYING and Copyright.html. COPYING can be found at the root *
* of the source code distribution tree; Copyright.html can be found at the *
* root level of an installed copy of the electronic HDF5 document set and *
* is linked from the top-level documents page. It can also be found at *
* http://www.hdfgroup.org/HDF5/doc/Copyright.html. If you do not have *

```

* access to either file, you may request a copy from help@hdfgroup.org. *

-->

<!-- #EndLibraryItem --><!--(TOC)=====-->

<SCRIPT language="JavaScript">

<!--

document.writeln (' \

<table x-use-null-cells\

align="right"\

width="240"\

cellspacing="0"\

class="tocTable">\

<tr valign="top"> \

<td class="tocTableHeaderCell" colspan="2"> \

Chapter Contents</td>\

</tr>\

-->

<!-- Table Version 3 -->\

<!--

<tr valign="top"> \

<td class="tocTableContentCell2"> \

1.</td>\

<td class="tocTableContentCell3">\

Introduction</td> \

</tr>\

<tr valign="top"> \

<td class="tocTableContentCell2"> \

2.</td>\

<td class="tocTableContentCell3">\

Programming Model
\

```

<font size="-1">\
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
<a href="#CreateFile">Create file</a>\
<br /> \
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
<a href="#CreateDataset">Create dataset</a>\
<br /> \
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
<a href="#CloseOperation">Close objects</a>\
<br /> \
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
<a href="#WriteRead">Write and read</a>\
<br /> \
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
<a href="#PartialWR">Partial write/read</a>\
<br /> \
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
<a href="#GetInfo">Get information</a>\
<br /> \
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
<a href="#CreateCDType">Create compound</a>\
<br /> \
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
<a href="#CreateCDType">datatype</a>\
<br /> \
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
<a href="#ExtendChunked">Create extendable or</a>\
<br /> \
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
<a href="#ExtendChunked">chunked dataset</a>\
</td>\

```

```

</tr>\
\
<tr valign="top"> \
  <td class="tocTableContentCell"> \
-->
<!-- editingComment -- "tocTableContentCell" and "tocTableContentCell4" \
-->\
<!-- are the table-closing cell class.\
  <td class="tocTableContentCell2"> \
-->\
<!--
  <a href="#IOPipeline">3.</a></td>\
  <td class="tocTableContentCell4">\
  <a href="#IOPipeline">Data Transfer Pipeline</a>\
-->
<!-- editingComment -- This section not currently complete or validated.\
</tr><tr valign="top"> \
  <td class="tocTableContentCell"> \
  <a href="#Appendix">10</a></td>\
  <td class="tocTableContentCell4"><a href="#Appendix">Appendix</a></td>\
-->\
<!--
  </td></tr>\
</table>\
')
-->
</SCRIPT>
<!--(End TOC 1)=====-->

<!--( TOC 2 )=====-->
<!--

```

```

<table x-use-null-cells
      align="right"
width="240"
cellspacing="0"
class="tocTable">
  <tr valign="top">
    <td class="tocTableHeaderCell">
      <span class="TableHead">Chapter Contents</span></td>
    </tr>
    <tr valign="top">
      <td class="tocTableContentCell">
        <a href="#Intro">1. Introduction</a>
        <br />
        <a href="#AbstractDMod">2. Abstract Data Model</a>
        <br />
        <a href="#SModel">3. HDF5 Storage Model</a>
        <br />
        <a href="#LibPModel">4. Library and</a>
        <br />
        &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="#LibPModel">Programming Model</a>
        <br />
        <a href="#IOPipeline">3. Data Transfer Pipeline</a>
        <br />
        &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="#Structure">HDF5 File</a>
      </td>
    </tr>
  </table>
-->

<!--(End TOC 2)=====-->

<div align="center">

```

``

3. The HDF5 Library and Programming Model

</div>

<!-- editingComment

[[[

]]

-->

<!-- HEADER LEFT " " -->

<!-- HEADER RIGHT " " -->

<!-- HEADER LEFT "HDF5 User's Guide" -->

<!-- HEADER RIGHT "HDF5 Library and Programming Model" -->

<h3>2.1. Introduction</h3>


```
<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="LibPModel">
<div align="right">
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>
```

<p>The HDF5 Library implements the HDF5 abstract data model and storage model. These models were described in the preceding chapter, “The HDF5 Data Model”.</p>

<p>Two major objectives of the HDF5 products are to provide tools that can be used on as many computational platforms as possible (portability), and to provide a reasonably object-oriented data model and programming interface.</p>

```
<!-- editingComment
<span class="editingComment">[ [ [
Explain? E.g., Java is portable,
but there are many platforms on which it does not run.
] ] ]</span>
-->
```

<p>To be as portable as possible, the HDF5 Library is implemented in

portable C. C is not an object-oriented language, but the library uses several mechanisms and conventions to implement an object model.

One mechanism the HDF5 library uses is to implement the objects as data structures. To refer to an object, the HDF5 library implements its own pointers. These pointers are called identifiers.

An identifier is then used to invoke operations on a specific instance of an object. For example, when a group is opened, the API returns a group identifier. This identifier is a reference to that specific group and will be used to invoke future operations on that group.

The identifier is valid only within the context it is created and remains valid until it is closed or the file is closed.

This mechanism is essentially the same as the mechanism that C++ or other object-oriented languages use to refer to objects except that the syntax is C.

Similarly, object-oriented languages collect all the methods for an object in a single name space. An example is the methods of a C++ class.

The C language does not have any such mechanism, but the HDF5 Library simulates this through its API naming convention.

API function names begin with a common prefix that is related to the class of objects that the function operates on.

The table below lists the HDF5 objects and the standard prefixes used by the corresponding HDF5 APIs.

For example, functions that operate on datatype objects all have names beginning with H5T.

<!-- NEW PAGE -->

<table width="300" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="2" align="left" valign="bottom">

Table 1. The HDF5 API naming scheme

Prefix

Operates on

H5A

Attributes

H5D

Datasets

H5E

Error reports

H5F

Files

```
<td>H5G</td>
<td>Groups</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>H5I</td>
  <td>Identifiers</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>H5L</td>
  <td>Links</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>H5O</td>
  <td>Objects</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>H5P</td>
  <td>Property lists</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>H5R</td>
  <td>References</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>H5S</td>
```

```

    <td>Dataspaces</td>
  </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>H5T</td>
  <td>Datatypes</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>H5Z</td>
  <td>Filters</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

```

<a name="ProgModel">

```

```

<h3>2.2. The HDF5 Programming Model</h3>

```

```

</a>

```

```

<p>In this section we introduce the HDF5

```

```

<span class="termDefinition">programming model</span> by means of

```

```

a series of short code samples. These samples illustrate a broad

```

```

selection of common HDF5 tasks. More details are provided in the

```

```

following chapters and in the

```

```

<a href="../RM/RM_H5Front.html" TARGET="H5DocWin">

```

```

<cite>HDF5 Reference Manual</cite></a></p>

```

```
<!-- editingComment
<span class="editingComment"><p>[ [
The following is based on text from the old "Intro to HDF5"
and presumably needs some technical verification.
] ]</span>
-->
```

```
<a name="CreateFile">
<h4>2.2.1. Creating an HDF5 File</h4></a>
```

Before an HDF5 file can be used or referred to in any manner, it must be explicitly created or opened. When the need for access to a file ends, the file must be closed. The example below provides a C code fragment illustrating these steps. In this example, the values for the file creation property list and the file access property list are set to the defaults `H5P_DEFAULT`.

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
hid_t    file;          /* declare file identifier */
/*
 * Create a new file using H5F_ACC_TRUNC
 * to truncate and overwrite any file of the same name,
 * default file creation properties, and
 * default file access properties.
 * Then close the file.
 */
file = H5Fcreate(FILE, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
```

```
status = H5Fclose(file); </pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 1. Creating and closing an HDF5 file</b>
    <hr color="green" size="3"/></td>
    <!-- formerly Figure 1-->
  </tr>
</table>
<br />
```

Note: If there is a possibility that a file of the declared name already exists and you wish to open a new file regardless of that possibility, the flag `H5F_ACC_TRUNC` will cause the operation to overwrite the previous file. If the operation should fail in such a circumstance, use the flag `H5F_ACC_EXCL` instead.

[CreateDataset](#)

2.2.2. Creating and Initializing a Dataset

The essential objects within a dataset are datatype and dataspace. These are independent objects and are created separately from any dataset to which they may be attached. Hence, creating a dataset requires, at a minimum, the following steps:


```
<li>Create and initialize a dataspace for the dataset</li>
<li>Define a datatype for the dataset</li>
<li>Create and initialize the dataset</li>
</ol>
```

<!-- NEW PAGE -->

<p>The code in the example below illustrates the execution of these steps.</p>

```
<table width="600" cellpadding="0" align="center">
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<hr color="green" size="3"/>
```

```
<pre>
```

```
hid_t  dataset, datatype, dataspace; /* declare identifiers */
```

```
/*
```

```
 * Create a dataspace: Describe the size of the array and
```

```
 * create the dataspace for a fixed-size dataset.
```

```
*/
```

```
dimsf[0] = NX;
```

```
dimsf[1] = NY;
```

```
dataspace = H5Screate_simple(RANK, dimsf, NULL);
```

```
/*
```

```
 * Define a datatype for the data in the dataset.
```

```
 * We will store little endian integers.
```

```
*/
```

```
datatype = H5Tcopy(H5T_NATIVE_INT);
```

```
status = H5Tset_order(datatype, H5T_ORDER_LE);
```

```
/*
```

```
 * Create a new dataset within the file using the defined
```

```
 * dataspace and datatype and default dataset creation
```

```
* properties.
* NOTE: H5T_NATIVE_INT can be used as the datatype if
* conversion to little endian is not needed.
*/
```

```
dataset = H5Dcreate(file, DATASETNAME, datatype, dataspace,
    H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT); </pre></td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 2. Create a dataset</b>
```

```
<hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

```
<a name="CloseObject">
```

```
<h4>2.2.3. Closing an Object </h4></a>
```

An application should close an object such as a datatype, dataspace, or dataset once the object is no longer needed.

Since each is an independent object, each must be released

(or closed) separately. This action is frequently referred to as

releasing the object's identifier.

The code in the example below *formerly Figure 3* closes the

datatype, dataspace, and dataset that were created in the preceding

section.

```
<table width="600" cellpadding="0" align="center">
```

```
<tr valign="top">
```

```

<td align="left">
  <hr color="green" size="3"/>
  <pre>
H5Tclose(datatype);
H5Dclose(dataset);
H5Sclose(dataspace);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 3. Close an object</b>
    <hr color="green" size="3"/></td>
    <!-- formerly Figure 3 -->
  </tr>
</table>
<br />

```

There is a long list of HDF5 Library items that return a unique identifier when the item is created or opened. Each time that one of these items is opened, a unique identifier is returned. Closing a file does not mean that the groups, datasets, or other open items are also closed. Each opened item must be closed separately.

<!-- FOR USE WITH ELECTRONIC VERSION ----->

For more information, see

["Using Identifiers"](../Advanced/UsingIdentifiers/index.html) in the "Advanced Topics" page.

<!-- FOR USE WITH ELECTRONIC VERSION ----->

<!-- FOR USE WITH PRINT VERSION -----

<p>For more information, see "Using Identifiers" in the
"Additional Resources" chapter.</p>

<!-- FOR USE WITH PRINT VERSION ----->

<h4>How Closing a File Effects Other Open Structural Elements</h4>

<p>Every structural element in an HDF5 file can be opened, and these elements can be opened more than once. Elements range in size from the entire file down to attributes. When an element is opened, the HDF5 Library returns a unique identifier to the application. Every element that is opened must be closed. If an element was opened more than once, each identifier that was returned to the application must be closed. For example, if a dataset was opened twice, both dataset identifiers must be released (closed) before the dataset can be considered closed. Suppose an application has opened a file, a group in the file, and two datasets in the group. In order for the file to be totally closed, the file, group, and datasets must each be closed. Closing the file before the group or the datasets will not effect the state of the group or datasets: the group and datasets will still be open.</p>

<p>There are several exceptions to the above general rule. One is when the `H5close` function is used. `H5close` causes a general shutdown of the library: all data is written to disk, all identifiers are closed, and all memory used by the library is cleaned up. Another exception occurs on parallel processing systems. Suppose on a parallel system an application has opened a file, a group in the file, and two datasets in the group. If the application uses the `H5Fclose` function to close the file, the call

will fail with an error. The open group and datasets must be closed before the file can be closed. A third exception is when the file access property list includes the property `H5F_CLOSE_STRONG`.

This property closes any open elements when the file is closed with

`H5Fclose`. For more information, see the

[./RM/RM_H5P.html#Property-SetFcloseDegree](http://www.hdfgroup.org/hdf/5.0/RM/RM_H5P.html#Property-SetFcloseDegree)

`H5Pset_fclose_degree` function in the

HDF5 Reference Manual.

[WriteRead](#)

2.2.4. Writing or Reading a Dataset to or from a File

Having created the dataset, the actual data can be written with a call to `H5Dwrite`. See the example below.

<pre> /* * Write the data to the dataset using default transfer * properties. */ status = H5Dwrite(dataset, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT, data); </pre>
<hr style="border: 1px solid green;"/>
<pre> Example 4. Writing a dataset </pre>

```
<hr color="green" size="3"/></td>
<!-- formerly Figure 4 -->
</tr>
</table>
<br />
```

Note that the third and fourth `H5Dwrite` parameters in the above example describe the dataspace in memory and in the file, respectively. For now, these are both set to `H5S_ALL` which indicates that the entire dataset is to be written.

The selection of partial datasets and the use of differing dataspace in memory and in storage will be discussed later in this chapter and in more detail elsewhere in this guide.

Reading the dataset from storage is similar to writing the dataset to storage. To read an entire dataset, substitute `H5Dread` for `H5Dwrite` in the above example.

[PartialWR](#)

2.2.5. Reading and Writing a Portion of a Dataset

The previous section described writing or reading an entire dataset. HDF5 also supports access to portions of a dataset. These parts of datasets are known as selections.

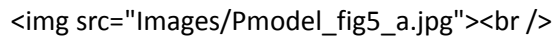
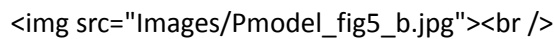
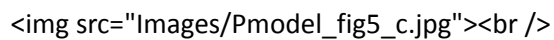
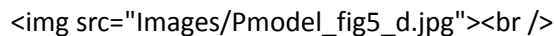
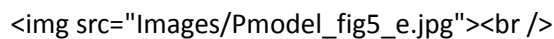
The simplest type of selection is a simple hyperslab. This is an n -dimensional rectangular sub-set of a dataset where n is equal to the

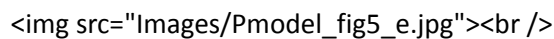
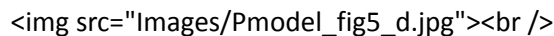
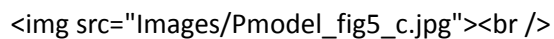
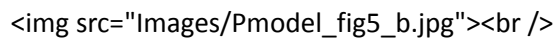
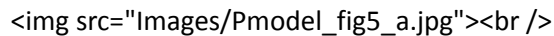
dataset's rank. Other available selections include
 a more complex hyperslab with user-defined stride and block size,
 a list of independent points, or the union of any of these.

The figure below <!-- formerly Figure 5 --> shows several
 sample selections.

<!-- NEW PAGE -->

<div data-bbox="207 739 422 774" data-label="Text"> <table border="0" style="width: 100%;"> <tr> <td style="text-align: center;"> <div data-bbox="233 793 477 829" data-label="Text"> <hr style="border: 1px solid green; width: 100%;"/> <div data-bbox="233 905 784 940" data-label="Text"></div> <div data-bbox="233 959 784 997" data-label="Text"></div> <div data-bbox="233 1014 784 1052" data-label="Text"></div> <div data-bbox="233 1071 784 1106" data-label="Text"></div> <div data-bbox="233 1125 784 1163" data-label="Text"></div> </div> </td> </tr> </table> </div>	<div data-bbox="233 793 477 829" data-label="Text"> <hr style="border: 1px solid green; width: 100%;"/> <div data-bbox="233 905 784 940" data-label="Text"></div> <div data-bbox="233 959 784 997" data-label="Text"></div> <div data-bbox="233 1014 784 1052" data-label="Text"></div> <div data-bbox="233 1071 784 1106" data-label="Text"></div> <div data-bbox="233 1125 784 1163" data-label="Text"></div> </div>
<div data-bbox="233 793 477 829" data-label="Text"> <hr style="border: 1px solid green; width: 100%;"/> <div data-bbox="233 905 784 940" data-label="Text"></div> <div data-bbox="233 959 784 997" data-label="Text"></div> <div data-bbox="233 1014 784 1052" data-label="Text"></div> <div data-bbox="233 1071 784 1106" data-label="Text"></div> <div data-bbox="233 1125 784 1163" data-label="Text"></div> </div>	

<div data-bbox="233 793 477 829" data-label="Text"> <hr style="border: 1px solid green; width: 100%;"/> <div data-bbox="233 905 784 940" data-label="Text"></div> <div data-bbox="233 959 784 997" data-label="Text"></div> <div data-bbox="233 1014 784 1052" data-label="Text"></div> <div data-bbox="233 1071 784 1106" data-label="Text"></div> <div data-bbox="233 1125 784 1163" data-label="Text"></div> </div>



<div data-bbox="233 793 477 829" data-label="Text"> <hr style="border: 1px solid green; width: 100%;"/> <div data-bbox="233 905 784 940" data-label="Text"></div> <div data-bbox="233 959 784 997" data-label="Text"></div> <div data-bbox="233 1014 784 1052" data-label="Text"></div> <div data-bbox="233 1071 784 1106" data-label="Text"></div> <div data-bbox="233 1125 784 1163" data-label="Text"></div> </div>

<div data-bbox="233 1404 446 1440" data-label="Text"> <div data-bbox="233 1459 742 1495" data-label="Text">Figure 1. Dataset selections</div> <div data-bbox="233 1514 544 1551" data-label="Text"><small><!-- formerly Figure 5. --></small></div> <div data-bbox="233 1570 883 1717" data-label="Text"> <p>Selections can take the form of a simple hyperslab, a hyperslab with user-defined stride and block, a selection of points, or a union of any of these forms.</p> </div> <div data-bbox="233 1736 289 1772" data-label="Text"><small><!--</small></div> <div data-bbox="259 1791 768 1829" data-label="Text"> <div data-bbox="259 1791 768 1829" data-label="Text"> <div data-bbox="259 1848 534 1885" data-label="Text">Edit figures to retain...</div> </div> </div> </div>

<div data-bbox="233 1404 446 1440" data-label="Text"> <div data-bbox="233 1459 742 1495" data-label="Text">Figure 1. Dataset selections</div> <div data-bbox="233 1514 544 1551" data-label="Text"><small><!-- formerly Figure 5. --></small></div> <div data-bbox="233 1570 883 1717" data-label="Text"> <p>Selections can take the form of a simple hyperslab, a hyperslab with user-defined stride and block, a selection of points, or a union of any of these forms.</p> </div> <div data-bbox="233 1736 289 1772" data-label="Text"><small><!--</small></div> <div data-bbox="259 1791 768 1829" data-label="Text"> <div data-bbox="259 1791 768 1829" data-label="Text"> <div data-bbox="259 1848 534 1885" data-label="Text">Edit figures to retain...</div> </div> </div> </div>

Figure 1. Dataset selections<!-- formerly Figure 5. -->

Selections can take the form of a simple hyperslab,
 a hyperslab with user-defined stride and block,
 a selection of points, or a union of any of these forms.

<!--

Edit figures to retain...

Edit figures to retain...

upper left,
 maybe a second simple hyperslab,
 2nd left,
 box on right (3-D point),
 and an interesting-looking union.

None of the text in the JPEGs need be retained.

Use new figure filenames as we are holding the current figure for possible reuse elsewhere.

If it's not already been done, remember that the original figure is to be used, in toto,
 in the "Memory<-->Disk Data Transfer" discussion, with "Key" to add characters '(hyperslab)'
 immediately to the right of the hyperslab icon.

]]

-->

<hr color="green" size="3"/></td>

</tr>

</table>

<!-- NEW PAGE -->

<p>Selections and hyperslabs are portions of a dataset.

As described above, a simple hyperslab

is a rectangular array of data elements with the same rank as the
 dataset's dataspace. Thus, a simple hyperslab is a
 logically contiguous collection of points within the dataset. </p>

<p>The more general case of a hyperslab
 can also be a regular pattern of points or blocks within the dataspace.

Four parameters are required to describe a general hyperslab: the
 starting coordinates, the block size, the stride or space between
 blocks, and the number of blocks. These parameters are each expressed

as a one-dimensional array with length equal to the rank of the dataspace

and are described in the table below

<!-- formerly table 2-->.</p>

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<td colspan="2" align="left" valign="bottom">
```

```
<b>Table 2. Hyperslab parameters</b>
```

```
</td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<b>Parameter&nbsp;&nbsp;&nbsp;</b></td>
```

```
<td>
```

```
<b>Definition</b></td>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td><code><i>start</i></code></td>
```

```
<td>
```

```
The coordinates of the starting location of the hyperslab
```

```
in the dataset's dataspace.</td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td><code><i>block</i></code></td>
```

```
<td>
```

```
The size of each block to be selected from the dataspace.
```

```
If the <code>block</code> parameter is set to NULL,
```

the block size defaults to a single element in each dimension,
as if the block array was set to all `1`s (all ones).

This will result in the selection of

a uniformly spaced set of `count` points

starting at `start` and

on the interval defined by `stride`.

`stride`

The number of elements separating the starting point of each element
or block to be selected.

If the `stride` parameter is set to NULL,

the stride size defaults to 1 (one) in each dimension

and no elements are skipped.

`count`

The number of elements or blocks to select along each dimension.

Reading Data into a Differently Shaped Memory Block

For maximum flexibility in user applications, a selection in storage
can be mapped into a differently-shaped selection in memory. All that

is required is that the two selections contain the same number of data elements. In this example, we will first define the selection to be read from the dataset in storage, and then we will define the selection as it will appear in application memory.

Suppose we want to read a 3 x 4 hyperslab from a two-dimensional dataset in a file beginning at the dataset element <1,2>.

The first task is to create the dataspace that describes the overall rank and dimensions of the dataset in the file and to specify the position and size of the in-file hyperslab that we are extracting from that dataset. See the code below.

formerly Figure 6-->

<!-- NEW PAGE -->

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
/*
 * Define dataset dataspace in file.
 */
dataspace = H5Dget_space(dataset); /* dataspace identifier */
rank      = H5Sget_simple_extent_ndims(dataspace);
status_n  = H5Sget_simple_extent_dims(dataspace, dims_out, NULL);

/*
 * Define hyperslab in the dataset.
 */
offset[0] = 1;
offset[1] = 2;
```

```

count[0] = 3;
count[1] = 4;
status = H5Sselect_hyperslab(dataspace, H5S_SELECT_SET, offset, NULL,
    count, NULL);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 5. Define the selection to be read from storage </b>
        <!-- formerly Figure 6.-->
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

<p>The next task is to define a dataspace in memory.

Suppose that we have in memory a three-dimensional 7 x 7 x 3 array into which we wish to read the two-dimensional 3 x 4 hyperslab described above and that we want the memory selection to begin at the element <3,0,0> and reside in the plane of the first two dimensions of the array. Since the in-memory dataspace is three-dimensional, we have to describe the in-memory selection as three-dimensional. Since we are keeping the selection in the plane of the first two dimensions of the in-memory dataset, the in-memory selection will be a 3 x 4 x 1 array defined as <3,4,1>.. </p>

<!-- NOT EDITED TO..... ----->

<p>Notice that we must describe two things: the dimensions of the in-memory array, and the size and position of the hyperslab that we wish to read in. The code below <!--formerly Figure 7 -->illustrates how this would be done.</p>

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
/*
 * Define memory dataspace.
 */
dimsm[0] = 7;
dimsm[1] = 7;
dimsm[2] = 3;
memspace = H5Screate_simple(RANK_OUT,dimsm,NULL);

/*
 * Define memory hyperslab.
 */
offset_out[0] = 3;
offset_out[1] = 0;
offset_out[2] = 0;
count_out[0] = 3;
count_out[1] = 4;
count_out[2] = 1;
status = H5Sselect_hyperslab(memspace, H5S_SELECT_SET, offset_out, NULL,
    count_out, NULL);</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      <b>Example 6. Define the memory dataspace and selection </b>

```

```

<hr color="green" size="3"/></td>
<!-- formerly Figure 7.-->
</tr>
</table>
<br />

```

The hyperslab defined in the code above has the following parameters:

`start=(3,0,0)`, `count=(3,4,1)`, stride and block size are `NULL`.

<!--TO HERE ----->

Writing Data into a Differently Shaped Disk Storage Block

Now let's consider the opposite process of writing a selection from memory to a selection in a dataset in a file. Suppose that the source dataspace in memory is a 50-element, one-dimensional array called `vector`

<!-- formerly Figure 8-->

and that the source selection is a 48-element simple hyperslab that starts at the second element of `vector`.

See the figure below.

```

<table width="400" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>

```

```

<tr>
  <td align="center"><br />
  <table align="center" border="1">
    <tr valign="top" align="center">
      <td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<code>-1</code>&nbsp;&nbsp;&nbsp;&nbsp;</td>
      <td>&nbsp;&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;&nbsp;</td>
      <td>&nbsp;&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;&nbsp;</td>
      <td>&nbsp;&nbsp;&nbsp;&nbsp;<code>3</code>&nbsp;&nbsp;&nbsp;&nbsp;</td>
      <td>&nbsp;&nbsp;&nbsp;&nbsp;<code>...</code>&nbsp;&nbsp;&nbsp;&nbsp;</td>
      <td>&nbsp;&nbsp;&nbsp;&nbsp;<code>49</code>&nbsp;&nbsp;&nbsp;&nbsp;</td>
      <td>&nbsp;&nbsp;&nbsp;&nbsp;<code>50</code>&nbsp;&nbsp;&nbsp;&nbsp;</td>
      <td>&nbsp;&nbsp;&nbsp;&nbsp;<code>-1</code>&nbsp;&nbsp;&nbsp;&nbsp;</td>
    </tr>
  </table>&nbsp;&nbsp;&
</td></tr>
</td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 2. A one-dimensional array</b>
    <!-- formerly Figure 8-->
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

Further suppose that we wish to write this data to the file as a series of 3 x 2-element blocks in a two-dimensional dataset, skipping one row and one column between blocks.

Since the source selection contains 48 data elements and

each block in the destination selection contains 6 data elements,

we must define the destination selection with 8 blocks.

We will write 2 blocks in the first dimension and 4 in the second.

The code below <!-- formerly Figure 9 --> shows how to achieve this objective.</p>

```
<table width="600" cellspacing="0" align="center">
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<hr color="green" size="3"/>
```

```
<pre>
```

```
/* Select the hyperslab for the dataset in the file, using 3 x 2 blocks,
```

```
 * a (4,3) stride, a (2,4) count, and starting at the position (0,1).
```

```
 */
```

```
start[0] = 0; start[1] = 1;
```

```
stride[0] = 4; stride[1] = 3;
```

```
count[0] = 2; count[1] = 4;
```

```
block[0] = 3; block[1] = 2;
```

```
ret = H5Sselect_hyperslab(fid, H5S_SELECT_SET, start, stride, count, block);
```

```
/*
```

```
 * Create dataspace for the first dataset.
```

```
 */
```

```
mid1 = H5Screate_simple(MSPACE1_RANK, dim1, NULL);
```

```
/*
```

```
/*
```

```
 * Select hyperslab.
```

```
 * We will use 48 elements of the vector buffer starting at the second element.
```

```
 * Selected elements are 1 2 3 . . . 48
```

```
 */
```



```

start[0] = 1;
stride[0] = 1;
count[0] = 48;
block[0] = 1;
ret = H5Sselect_hyperslab(mid1, H5S_SELECT_SET, start, stride, count, block);

/*
 * Write selection from the vector buffer to the dataset in the file.
 *
ret = H5Dwrite(dataset, H5T_NATIVE_INT, mid1, fid, H5P_DEFAULT, vector)</pre></td>

```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 7. The destination selection
```

```
<!-- formerly Figure 9--></b>
```

```
<hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

```
<!-- NEW PAGE -->
```

```
<a name="GetInfo">
```

```
<h4>2.2.6. Getting Information about a Dataset</h4>
```

```
</a>
```

Although reading is analogous to writing, it is often first necessary to query a file to obtain information about the dataset to be read.

For instance, we often need to determine the datatype associated with a dataset, or its dataspace (i.e., rank and dimensions).

As illustrated in the code example below <!--formerly Figure 10-->,

there are several `get` routines for obtaining this information.

<pre> /* * Get datatype and dataspace identifiers, * then query datatype class, order, and size, and * then query dataspace rank and dimensions. */ datatype = H5Dget_type (dataset); /* datatype identifier */ class = H5Tget_class (datatype); if (class == H5T_INTEGER) printf("Dataset has INTEGER type \n"); order = H5Tget_order (datatype); if (order == H5T_ORDER_LE) printf("Little endian order \n"); size = H5Tget_size (datatype); printf ("Size is %d \n", size); dataspace = H5Dget_space (dataset); /* dataspace identifier */ /* Find rank and retrieve current and maximum dimension sizes */ rank = H5Sget_simple_extent_dims (dataspace, dims, max_dims); </pre>

```
<td align="left">
```

```
<b>Example 8. Routines to get dataset parameters
```

```
<!--formerly Figure 10--></b>
```

```
<hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

```
<a name="CreateCDType">
```

```
<h4>2.2.7. Creating and Defining Compound Datatypes</h4></a>
```

A compound datatype is a collection of one or more data elements. Each element might be an atomic type, a small array, or another compound datatype.

The provision for nested compound datatypes allows these structures to become quite complex. An HDF5 compound datatype has some similarities to a C struct or a Fortran common block. Though not originally designed with databases in mind, HDF5 compound datatypes are sometimes used in a way that is similar to a database record. Compound datatypes can become either a powerful tool or a complex and difficult-to-debug construct. Reasonable caution is advised.

To create and use a compound datatype,

you need to create a datatype with class

compound (`H5T_COMPOUND`)

and specify the total size of the data element in bytes.

A compound datatype consists of zero or more uniquely named members.

Members can be defined in any order but must occupy non-overlapping regions

within the datum. The table below formerly

Table 3 --> lists the properties of compound datatype members.

```
<!-- NEW PAGE -->
```

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<td colspan="2" align="left" valign="bottom">
```

```
<b>Table 3. Compound datatype member properties</b></td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="2" /></td></tr>
```

```
<tr valign="top">
```

```
<td width="20%"><b>Parameter</b></td>
```

```
<td width="80%"><b>Definition</b></td>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>Index</td>
```

```
<td>An index number between zero and N-1, where N is the number of  
members in the compound. The elements are indexed in the order of  
their location in the array of bytes.</td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>Name</td>
```

```
<td>A string that must be unique within the members  
of the same datatype.</td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>Datatype</td>
```

```
<td>An HDF5 datatype.</td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

<td>Offset</td>

<td>A fixed byte offset which defines the location of the first
byte of that member in the compound datatype.</td>

</tr>

<tr><td colspan="2"><hr color="green" size="2" /></td></tr>

</table>

<p>Properties of the members of a compound datatype are
defined when the member is added to the compound type. These
properties cannot be modified later. </p>

<h4>Defining Compound Datatypes</h4>

<p>Compound datatypes must be built out of other datatypes.
To do this, you first create an empty compound datatype and specify
its total size. Members are then added to the compound datatype
in any order. </p>

<p>Each member must have a descriptive name. This is the
key used to uniquely identify the member within the
compound datatype. A member name in an HDF5 datatype does
not necessarily have to be the same as the name of the
corresponding member in the C struct in memory although
this is often the case. You also do not need to define all the
members of the C struct in the HDF5 compound datatype
(or vice versa). </p>

<p>Usually a C struct will be defined to hold a data
point in memory, and the offsets of the members in memory

will be the offsets of the struct members from the beginning of an instance of the struct. The library defines the macro that computes the offset of member `m` within a struct variable `s`.:

```
<code>HOFFSET(s,m)</code>
```

The code below <!-- formerly Figure 11 --> shows an example in which a compound datatype is created to describe complex numbers whose type is defined by the `complex_t` struct.

<!-- NEW PAGE -->

```
<table width="600" cellpadding="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="left">
```

```
      <hr color="green" size="3"/>
```

```
    <pre>
```

```
    Typedef struct {
```

```
      double re; /*real part */
```

```
      double im; /*imaginary part */
```

```
    } complex_t;
```

```
    complex_t tmp; /*used only to compute offsets */
```

```
    hid_t complex_id = H5Tcreate (H5T_COMPOUND, sizeof tmp);
```

```
    H5Tinsert (complex_id, "real", HOFFSET(tmp,re),
```

```
              H5T_NATIVE_DOUBLE);
```

```
    H5Tinsert (complex_id, "imaginary", HOFFSET(tmp,im),
```

```
              H5T_NATIVE_DOUBLE);</pre></td>
```

```
  </tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

<td align="left">

Example 9. A compound datatype for complex numbers

<!-- formerly Figure 11-->

<hr color="green" size="3"/></td>

</tr>

</table>

<!-- editingComment

<p>For more information about Datatypes, see Chapter ???.

-->

<h4>2.2.8. Creating and Writing Extendable Datasets</h4>

<p>An extendable dataset is one whose dimensions can grow.

One can define an HDF5 dataset to have certain initial

dimensions with the capacity to later increase the size

of any of the initial dimensions. For example, the figure below

<!-- formerly Figure 12 -->shows a 3 x 3 dataset (a)

which is later extended to be a 10 x 3 dataset by adding 7

rows (b), and further extended to be a 10 x 5 dataset by

adding two columns (c).</p>

<table width="600" cellspacing="0" align="center">

<tr valign="top">

```
<td align="center">
```

```
<hr color="green" size="3"/>
```

```
<table width="80%" cellpadding="0" class="fullImgTable" align="center">
```

```
<tr>
```

```
<td align="center"><br />
```

```
<table align="center">
```

```
<tr>
```

```
<td valign="middle" align="center" width="45%">
```

```
<table align="center" border="1">
```

```
<tr>
```

```
<td>&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;</td>
```

```
<td>&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;</td>
```

```
<td>&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;</td>
```

```
</tr>
```

```
<tr>
```

```
<td>&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;</td>
```

```
<td>&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;</td>
```

```
<td>&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;</td>
```

```
</tr>
```

```
<tr>
```

```
<td>&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;</td>
```

```
<td>&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;</td>
```

```
<td>&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;</td>
```

```
</tr>
```

```
</table>
```

```
a)&nbsp;&nbsp;&nbsp;Initially,&nbsp;&nbsp;&nbsp;3&nbsp;&nbsp;&nbsp;x&nbsp;&nbsp;&nbsp;3
```

```
<br />&nbsp;&nbsp;&nbsp;<br />
```

```
<table align="center" border="1">
```



```

<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
</tr>
<tr>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
</tr>
<tr>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
</tr>
<tr>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
<td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
</tr>
</table>
b)&nbsp;&nbsp;&nbsp;Extend&nbsp;&nbsp;&nbsp;to&nbsp;&nbsp;&nbsp;10&nbsp;&nbsp;&nbsp;x&nbsp;&nbsp;&nbsp;3

```

```

</td>
<td valign="middle" align="center" width="55%">
<code>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</code>
</td>
<td valign="middle" align="center" width="55%">

```

```

<table align="center" border="1">
<tr>
<td>&nbsp;&nbsp;&nbsp;<code>1</code>&nbsp;&nbsp;&nbsp;</td>

```


[illegible]

```

        <td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
        <td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
        <td>&nbsp;&nbsp;&nbsp;<code>2</code>&nbsp;&nbsp;&nbsp;</td>
        <td>&nbsp;&nbsp;&nbsp;<code>3</code>&nbsp;&nbsp;&nbsp;</td>
        <td>&nbsp;&nbsp;&nbsp;<code>3</code>&nbsp;&nbsp;&nbsp;</td>
    </tr>
</table>
    c)&nbsp;&nbsp;&nbsp;Extend&nbsp;&nbsp;&nbsp;to&nbsp;&nbsp;&nbsp;10&nbsp;&nbsp;&nbsp;x&nbsp;&nbsp;&nbsp;5
</tr>
</table>&nbsp;&nbsp;&
</td></tr>
</table>
</td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left" >
        <b>Figure 3. Extending a dataset</b>
        <!-- formerly Figure 12-->
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

HDF5 requires the use of chunking when defining extendable datasets. Chunking makes it possible to extend datasets efficiently without having to reorganize contiguous storage excessively.

To summarize, an extendable dataset requires two conditions:

- Define the dataspace of the dataset as unlimited in all dimensions that might eventually be extended
- Enable chunking in the dataset creation properties

<p>For example, suppose we wish to create a dataset similar to the one shown in the figure above<!-- formerly Figure 12-->. We want to start with a 3 x 3 dataset, and then later we will extend it. To do this, go through the steps below.</p>

<p>First, declare the dataspace to have unlimited dimensions. See the code shown below. Note the use of the predefined constant <code>H5S_UNLIMITED</code> to specify that a dimension is unlimited.</p>

<pre> Hsize_t dims[2] = {3, 3}; /* dataset dimensions at the creation time */ hsize_t maxdims[2] = {H5S_UNLIMITED, H5S_UNLIMITED}; /* * Create the data space with unlimited dimensions. */ dataspace = H5Screate_simple(RANK, dims, maxdims); </pre>

Example 10. Declaring a dataspace with unlimited dimensions

← formerly Figure 13 →

Next, set the dataset creation property list to enable chunking. See the code below.

--

--

--

```


```

```

hid_t cparms;

```

```

hsize_t chunk_dims[2] = {2, 5};

```

```

/*

```

```

 * Modify dataset creation properties to enable chunking.

```

```

 */

```

```

cparms = H5Pcreate (H5P_DATASET_CREATE);

```

```

status = H5Pset_chunk(cparms, RANK, chunk_dims);

```

--

--

--

Example 11. Enable chunking

← formerly Figure 14 →

</tr>

</table>

<!-- NEW PAGE -->

<p>The next step is to create the dataset. See the code below.</p>

<table width="600" cellpadding="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

/*

* Create a new dataset within the file using cparms

* creation properties.

*/

dataset = H5Dcreate(file, DATASETNAME, H5T_NATIVE_INT, dataspace,

H5P_DEFAULT, cparms, H5P_DEFAULT);</pre></td>

</tr>

<tr><td><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td align="left">

Example 12. Create a dataset

<!-- formerly Figure 15-->

<hr color="green" size="3"/></td>

</tr>

</table>

Finally, when the time comes to extend the size of the dataset, invoke `H5Dextend`. Extending the dataset along the first dimension by seven rows leaves the dataset with new dimensions of `<10,3>`. See the code below.

<pre> /* * Extend the dataset. Dataset becomes 10 x 3. */ dims[0] = dims[0] + 7; size[0] = dims[0]; size[1] = dims[1]; status = H5Dextend (dataset, size); </pre>	<hr style="border: 1px solid green;"/>
<hr style="border: 1px solid green;"/>	
<pre> </pre>	<p>Example 13. Extend the dataset by seven rows</p> <p><i>formerly Figure 16--></i></p> <hr style="border: 1px solid green;"/>

[Groups](#)

2.2.9. Creating and Working with Groups

Groups provide a mechanism for organizing meaningful and extendable sets of datasets within an HDF5 file. The H5G API provides several routines for working with groups.

Creating a Group

With no datatype, dataspace, or storage layout to define, creating a group is considerably simpler than creating a dataset. For example, the following code creates a group called `Data` in the root group of `file`.

<pre> /* * Create a group in the file. */ grp = H5Gcreate(file, "/Data", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT); </pre>
<hr style="border: 1px solid green; width: 30px; margin: 0 auto;"/>
<pre> Example 14. Create a group <!-- formerly Figure 17--> </pre> </td> </tr> </table> </div> <div data-bbox="111 937 137 953" data-label="Page-Footer">92</div> <div data-bbox="750 937 883 955" data-label="Page-Footer">The HDF Group</div>--></pre>

```

        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

```
<!-- NEW PAGE -->
```

A group may be created within another group by providing the absolute name of the group to the `H5Gcreate` function or by specifying its location. For example, to create the group `Data_new` in the group `Data`, you might use the sequence of calls shown below.

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
/*
 * Create group "Data_new" in the group "Data" by specifying
 * absolute name of the group.
 */
grp_new = H5Gcreate(file, "/Data/Data_new", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

or

/*
 * Create group "Data_new" in the "Data" group.
 */
grp_new = H5Gcreate(grp, "Data_new", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);</pre></td>
    </tr>

```

```
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 15. Create a group within a group</b>
    <!-- formerly Figure 18-->
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
```

This first parameter of `H5Gcreate` is a location identifier. `file` in the first example specifies only the file. `grp` in the second example specifies a particular group in a particular file. Note that in this instance, the group identifier `grp` is used as the first parameter in the `H5Gcreate` call so that the relative name of `Data_new` can be used.

The third parameter of `H5Gcreate` optionally specifies how much file space to reserve to store the names of objects that will be created in this group. If a non-positive value is supplied, the library provides a default size.

Use `H5Gclose` to close the group and release the group identifier.

<!-- NEW PAGE -->

Creating a Dataset within a Group

As with groups, a dataset can be created in a particular group

by specifying either its absolute name in the file or its relative name with respect to that group. The next code excerpt uses the absolute name.

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
/*
 * Create the dataset "Compressed_Data" in the group Data using the
 * absolute name. The dataset creation property list is modified
 * to use GZIP compression with the compression effort set to 6.
 * Note that compression can be used only when the dataset is
 * chunked.
 */
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_deflate(plist, 6);
dataset = H5Dcreate(file, "/Data/Compressed_Data",
    H5T_NATIVE_INT, dataspace, H5P_DEFAULT, plist, H5P_DEFAULT);
      </pre>
    </td><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      <b>Example 16. Create a dataset within

```

```

        a group using an absolute name </b>
        <!-- formerly Figure 19-->
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

Alternatively, you can first obtain an identifier for the group in which the dataset is to be created, and then create the dataset with a relative name.

```

<table width="600" cellspacing="0" align="center">
    <tr valign="top">
        <td align="left">
            <hr color="green" size="3"/>
            <pre>
/*
 * Open the group.
 */
grp = H5Gopen(file, "Data", H5P_DEFAULT);

/*
 * Create the dataset "Compressed_Data" in the "Data" group
 * by providing a group identifier and a relative dataset
 * name as parameters to the H5Dcreate function.
 */
dataset = H5Dcreate(grp, "Compressed_Data", H5T_NATIVE_INT,
                    dataspace, H5P_DEFAULT, plist, H5P_DEFAULT);
            </pre>
        </td>
    </tr>
    <tr><td><hr color="green" size="1" /></td></tr>

```

```
<tr valign="top">
  <td align="left">
    <b>Example 17. Create a dataset within a group using a relative name</b>
    <!-- formerly Figure 20-->
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
```

<!-- NEW PAGE -->

<h4>Accessing an Object in a Group</h4>

<p>Any object in a group
can be accessed by its absolute or relative name. The
first code snippet below illustrates the use of the absolute
name to access the dataset <code>Compressed_Data</code> in
the group <code>Data</code> created in the examples above.
The second code snippet illustrates the use of the
relative name.</p>

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
/*
 * Open the dataset "Compressed_Data" in the "Data" group.
 */
dataset = H5Dopen(file, "/Data/Compressed_Data", H5P_DEFAULT);
```

```

</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 18. Accessing a group using its absolute name</b>
    <!-- formerly Figure 21-->
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
<br />

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
/*
 * Open the group "data" in the file.
 */
grp = H5Gopen(file, "Data", H5P_DEFAULT);

/*
 * Access the "Compressed_Data" dataset in the group.
 */
dataset = H5Dopen(grp, "Compressed_Data", H5P_DEFAULT);</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">

```



```
<td align="left">
<b>Example 19. Accessing a group using its relative name</b>
<!-- formerly Figure 22-->
<hr color="green" size="3"/></td>
</tr>
</table>
<br />
```

```
<!-- NEW PAGE -->
<a name="Attrs">
<h4>2.2.10. Working with Attributes</h4>
</a>
```

An attribute is a small dataset that is attached to a normal dataset or group. Attributes share many of the characteristics of datasets, so the programming model for working with attributes is similar in many ways to the model for working with datasets. The primary differences are that an attribute must be attached to a dataset or a group and sub-setting operations cannot be performed on attributes. </p>

To create an attribute belonging to a particular dataset or group, first create a dataspace for the attribute with the call to `H5Screate`, and then create the attribute using `H5Acreate`. For example, the code shown below creates an attribute called `Integer_attribute` that is a member of a dataset whose identifier is `dataset`. The attribute identifier is `attr2`. `H5Awrite` then sets the value of the attribute of that of the integer variable point.

`H5Aclose` then releases the attribute identifier.

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
Int point = 1;          /* Value of the scalar attribute */

/*
 * Create scalar attribute.
 */
aid2 = H5Screate(H5S_SCALAR);
attr2 = H5Acreate(dataset, "Integer attribute", H5T_NATIVE_INT, aid2,
                  H5P_DEFAULT, H5P_DEFAULT);

/*
 * Write scalar attribute.
 */
ret = H5Awrite(attr2, H5T_NATIVE_INT, &point);

/*
 * Close attribute dataspace.
 */
ret = H5Sclose(aid2);

/*
 * Close attribute.
```

```

*/
ret = H5Aclose(attr2); </pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 20. Create an attribute</b>
    <!-- formerly Figure 23-->
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

To read a scalar attribute whose name and datatype are known, first open the attribute using `H5Aopen_by_name`, and then use `H5Aread` to get its value. For example, the code shown below reads a scalar attribute called `Integer_attribute` whose datatype is a native integer and whose parent dataset has the identifier `dataset`.

```

<!-- NEW PAGE -->
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
/*
 * Attach to the scalar attribute using attribute name, then read and
 * display its value.
 */

```

```

attr = H5Aopen_by_name(file_id, dataset_name, "Integer attribute",
                        H5P_DEFAULT, H5P_DEFAULT);
ret = H5Aread(attr, H5T_NATIVE_INT, &point_out);
printf("The value of the attribute \"Integer attribute\" is %d \n", point_out);
ret = H5Aclose(attr);

```

```

</tr>

```

```

<tr><td><hr color="green" size="1" /></td></tr>

```

```

<tr valign="top">

```

```

<td align="left">

```

```

<b>Example 21. Read a known attribute</b>

```

```

<!-- formerly Figure 24-->

```

```

<hr color="green" size="3"/></td>

```

```

</tr>

```

```

</table>

```

```

<br />

```

To read an attribute whose characteristics are not known, go through these steps. First, query the file to obtain information about the attribute such as its name, datatype, rank, and dimensions, and then read the attribute. The following code opens an attribute by its index value using `H5Aopen_by_idx`, and then it reads in information about the datatype with `H5Aread`.

```

<table width="600" cellspacing="0" align="center">

```

```

<tr valign="top">

```

```

<td align="left">

```

```

<hr color="green" size="3"/>

```

```

<pre>

```

```

/*
 * Attach to the string attribute using its index, then read and display the value.
 */
attr = H5Aopen_by_idx(file_id, dataset_name, index_type, iter_order, 2,
                      H5P_DEFAULT, H5P_DEFAULT);
atype = H5Tcopy(H5T_C_S1);
H5Tset_size(atype, 4);
ret = H5Aread(attr, atype, string_out);
printf("The value of the attribute with the index 2 is %s\n", string_out);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 22. Read an unknown attribute
    <!-- formerly Figure 25--></b>
    <hr color="green" size="3" /></td>
  </tr>
</table>
<br />

```

In practice, if the characteristics of attributes are not known, the code involved in accessing and processing the attribute can be quite complex. For this reason, HDF5 includes a function called `H5Aiterate`. This function applies a user-supplied function to each of a set of attributes. The user-supplied function can contain the code that interprets, accesses, and processes each attribute.

<!-- NEW PAGE -->

<h3>2.3. The Data Transfer Pipeline</h3>

<!-- editingComment

[[[

This section needs to be reviewed in detail by QAK and others.

]]<

br>

[[[

There probably should be a whole chapter on data transfer,

selection, transformation, etc.}

]]

-->

<p>The HDF5 Library implements data transfers between

different storage locations. At the lowest levels, the

HDF5 Library reads and writes blocks of bytes to and from

storage using calls to the virtual file layer (VFL) drivers.

In addition to this, the HDF5 Library manages caches of metadata

and a data I/O pipeline. The data I/O pipeline applies compression

to data blocks, transforms data elements, and implements selections. </p>

<p>A substantial portion of the HDF5 Library's work is in

transferring data from one environment or media to another. This

most often involves a transfer between system memory and a storage

medium. Data transfers are affected by compression, encryption,

machine-dependent differences in numerical representation, and other

features. So, the bit-by-bit arrangement of a given dataset is

often substantially different in the two environments.</p>

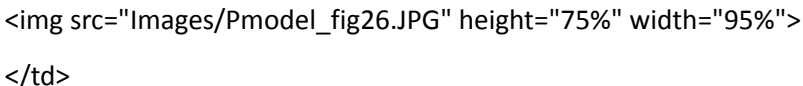
<p>Consider the representation on disk of a compressed and

encrypted little-endian array as compared to the same array

after it has been read from disk, decrypted, decompressed, and loaded into memory on a big-endian system. HDF5 performs all of the operations necessary to make that transition during the I/O process with many of the operations being handled by the VFL and the data transfer pipeline.

The figure below formerly Figure 26 provides a simplified view of a sample data transfer with four stages. Note that the modules are used only when needed. For example, if the data is not compressed, the compression stage is omitted.

NEW PAGE

<div style="text-align: center;"> <hr style="border: 1px solid green; width: 300px; margin: 0 auto;"/>  </div>
<hr style="border: 1px solid green; width: 100%; margin: 10px 0;"/>
<div style="display: flex;"> <div style="flex: 1; padding-right: 10px;"> <p>Figure 4. A data transfer from storage to memory</p> <p><small>formerly Figure 26</small></p> <hr style="border: 1px solid green; width: 100%; margin: 0;"/> </div> </div>

For a given I/O request, different combinations of actions may be performed by the pipeline. The library

automatically sets up the pipeline and passes data through the processing steps. For example, for a `read` request (from disk to memory), the library must determine which logical blocks contain the requested data elements and fetch each block into the library's cache. If the data needs to be decompressed, then the compression algorithm is applied to the block after it is read from disk. If the data is a selection, the selected elements are extracted from the data block after it is decompressed. If the data needs to be transformed (for example, byte swapped), then the data elements are transformed after decompression and selection.

While an application must sometimes set up some elements of the pipeline, use of the pipeline is normally transparent to the user program. The library determines what must be done based on the metadata for the file, the object, and the specific request. An example of when an application might be required to set up some elements in the pipeline is if the application used a custom error-checking algorithm.

`<!-- editingComment`

``

For more details of the pipeline, see `[citeit]`.

``

`-->`

In some cases, it is necessary to pass parameters to and from modules in the pipeline or among other parts of the library that are not directly called through the programming API. This is accomplished through the use of dataset transfer and data access property lists.

The VFL provides an interface whereby user applications can add custom modules to the data transfer pipeline. For example, a custom compression algorithm can be used with the HDF5 Library by linking an appropriate module into the pipeline through the VFL. This requires creating an appropriate wrapper for the compression module

```
<!-- editingComment
```

```
<span class="editingComment">[ [ [
[cite filter doc and ref manual]
] ] ]</span>
```

```
-->
```

and registering it with the library with `H5Zregister`.

The algorithm can then be applied to a dataset with an

`H5Pset_filter` call which will add the algorithm to the selected dataset's transfer property list.

```
<SCRIPT language="JavaScript">
```

```
<!--
```

```
document.writeln ("
```

```
<div align="right">
```

```
<a href="#TOP"><font size="-1">(Top)</font></a>
```

```
</div>
```

```
</a>
```

```
";
```

```
-->
```

```
</SCRIPT>
```

```
<br /><br /><br />
```

```
<br /><br /><br />
```

</body>

</html>

I

4. The HDF5 File

4.1. Introduction to HDF5 Files

The purpose of this chapter is to describe how to work with HDF5 data files.

If HDF5 data is to be written to or read from a file, the file must first be explicitly created or opened with the appropriate file driver and access privileges. Once all work with the file is complete, the file must be explicitly closed.

This chapter discusses the following:

- File access modes
- Creating, opening, and closing files
- The use of file creation property lists
- The use of file access property lists
- The use of low-level file drivers

This chapter assumes an understanding of the material presented in the data model chapter. For more information, see "[The HDF5 Data Model and File Structure](#)" on page 7.

4.1.1. File Access Modes

There are two issues regarding file access:

- What should happen when a new file is created but a file of the same name already exists? Should the create action fail, or should the existing file be overwritten?
- Is a file to be opened with read-only or read-write access?

Four access modes address these concerns. Two of these modes can be used with `H5Fcreate`, and two modes can be used with `H5Fopen`.

- `H5Fcreate` accepts `H5F_ACC_EXCL` or `H5F_ACC_TRUNC`
- `H5Fopen` accepts `H5F_ACC_RDONLY` or `H5F_ACC_RDWR`

The access modes are described in the table below.

Table 4-1. Access flags and modes

Access Flag	Resulting Access Mode
H5F_ACC_EXCL	If the file already exists, H5Fcreate fails. If the file does not exist, it is created and opened with read-write access. (<i>Default</i>)
H5F_ACC_TRUNC	If the file already exists, the file is opened with read-write access, and new data will overwrite any existing data. If the file does not exist, it is created and opened with read-write access.
H5F_ACC_RDONLY	An existing file is opened with read-only access. If the file does not exist, H5Fopen fails. (<i>Default</i>)
H5F_ACC_RDWR	An existing file is opened with read-write access. If the file does not exist, H5Fopen fails.

By default, H5Fopen opens a file for read-only access; passing H5F_ACC_RDWR allows read-write access to the file.

By default, H5Fcreate fails if the file already exists; only passing H5F_ACC_TRUNC allows the truncating of an existing file.

4.1.2. File Creation and File Access Properties

File creation and file access property lists control the more complex aspects of creating and accessing files.

File creation property lists control the characteristics of a file such as the size of the user-block, a user-definable data block; the size of data address parameters; properties of the B-trees that are used to manage the data in the file; and certain HDF5 library versioning information.

For more information, see "[File Creation Properties](#)" on page 124. This section has a more detailed discussion of file creation properties. If you have no special requirements for these file characteristics, you can simply specify H5P_DEFAULT for the default file creation property list when a file creation property list is called for.

File access property lists control properties and means of accessing a file such as data alignment characteristics, metadata block and cache sizes, data sieve buffer size, garbage collection settings, and parallel I/O. Data alignment, metadata block and cache sizes, and data sieve buffer size are factors in improving I/O performance.

For more information, see "[File Access Properties](#)" on page 125. This section has a more detailed discussion of file access properties. If you have no special requirements for these file access characteristics, you can simply specify `H5P_DEFAULT` for the default file access property list when a file access property list is called for.

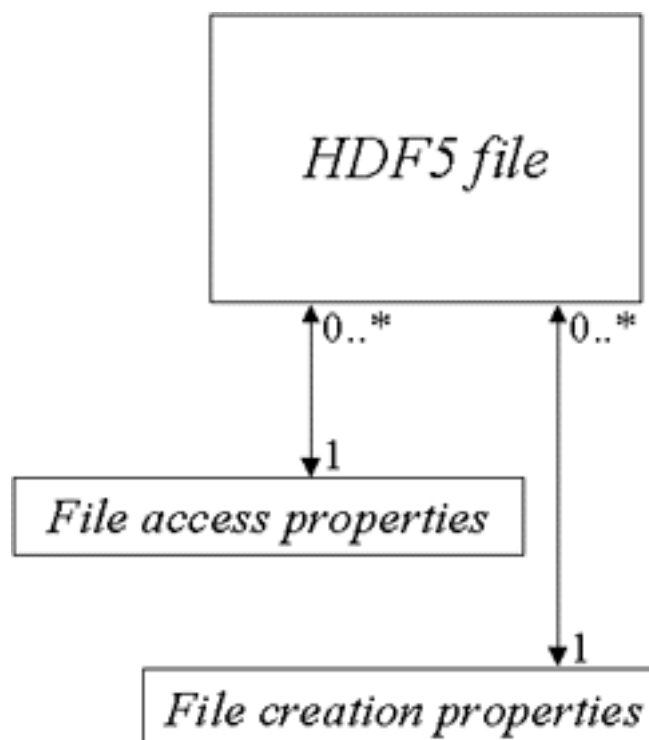


Figure 4-1. UML model for an HDF5 file and its property lists

4.1.3. Low-level File Drivers

The concept of an HDF5 file is actually rather abstract: the address space for what is normally thought of as an HDF5 file might correspond to any of the following at the storage level:

- Single file on a standard file system
- Multiple files on a standard file system
- Multiple files on a parallel file system
- Block of memory within an application's memory space
- More abstract situations such as virtual files

This HDF5 address space is generally referred to as an *HDF5 file* regardless of its organization at the storage level.

HDF5 accesses a file (the address space) through various types of *low-level file drivers*. The default HDF5 file storage layout is as an unbuffered permanent file which is a single, contiguous file on local disk. Alternative layouts are designed to suit the needs of a variety of systems, environments, and applications.

4.2. Programming Model

Programming models for creating, opening, and closing HDF5 files are described in the sub-sections below.

4.2.1. Creating a New File

The programming model for creating a new HDF5 file can be summarized as follows:

- Define the file creation property list
- Define the file access property list
- Create the file

First, consider the simple case where we use the default values for the property lists. See the example below.

```
file_id = H5Fcreate ("SampleFile.h5", H5F_ACC_EXCL,  
H5P_DEFAULT, H5P_DEFAULT)
```

Code Example 4-1. Creating an HDF5 file using property list defaults

Note: This example specifies that `H5Fcreate` should fail if `SampleFile.h5` already exists.

A more complex case is shown in the example below. In this example, we define file creation and access property lists (though we do not assign any properties), specify that `H5Fcreate` should fail if `SampleFile.h5` already exists, and create a new file named `SampleFile.h5`. The example does not specify a driver, so the default driver, `H5FD_SEC2`, will be used.

```
fcplist_id = H5Pcreate (H5P_FILE_CREATE)
<...set desired file creation properties...>
faplist_id = H5Pcreate (H5P_FILE_ACCESS)
<...set desired file access properties...>
file_id = H5Fcreate ("SampleFile.h5", H5F_ACC_EXCL, fcplist_id,
                    faplist_id)
```

Code Example 4-2. Creating an HDF5 file using property lists

Notes:

A root group is automatically created in a file when the file is first created.

File property lists, once defined, can be reused when another file is created within the same application.

4.2.2. Opening an Existing File

The programming model for opening an existing HDF5 file can be summarized as follows:

- Define or modify the file access property list including a low-level file driver (optional)
- Open the file

The code in the example below shows how to open an existing file with read-only access.

```
faplist_id = H5Pcreate (H5P_FILE_ACCESS)
status = H5Pset_fapl_stdio (faplist_id)
file_id = H5Fopen ("SampleFile.h5", H5F_ACC_RDONLY, faplist_id)
```

Code Example 4-3. Opening an HDF5 file

4.2.3. Closing a File

The programming model for closing an HDF5 file is very simple:

- Close file

We close `SampleFile.h5` with the code in the example below.

```
status = H5Fclose (file_id)
```

Code Example 4-4. Closing an HDF5 file

Note that `H5Fclose` flushes all unwritten data to storage and that `file_id` is the identifier returned for `SampleFile.h5` by `H5Fopen`.

More comprehensive discussions regarding all of these steps are provided below.

4.3. Using `h5dump` to View a File

`h5dump` is a command-line utility that is included in the HDF5 distribution. This program provides a straight-forward means of inspecting the contents of an HDF5 file. You can use `h5dump` to verify that a program is generating the intended HDF5 file. `h5dump` displays ASCII output formatted according to the HDF5 DDL grammar.

The following `h5dump` command will display the contents of `SampleFile.h5`:

```
h5dump SampleFile.h5
```

If no datasets or groups have been created in and no data has been written to the file, the output will look something like the following:

```
HDF5 "SampleFile.h5" {  
  GROUP "/" {  
  }  
}
```

Note that the root group, indicated above by `/`, was automatically created when the file was created.

`h5dump` is described on the [Tools](#) page of the [HDF5 Reference Manual](#). The HDF5 DDL grammar is described in the document [DDL in BNF for HDF5](#).

4.4. File Function Summaries

General library functions and macros (H5), file functions (H5F), file related property list functions (H5P), and file driver functions (H5D) are listed below.

Function Listing 4-1. General library functions and macros (H5)

C Function Fortran Function	Purpose
H5check_version h5check_version_f	Verifies that HDF5 library versions are consistent.
H5close h5close_f	Flushes all data to disk, closes all open identifiers, and cleans up memory.
H5dont_atexit h5dont_atexit_f	Instructs the library not to install the <code>atexit</code> cleanup routine.
H5garbage_collect h5garbage_collect_f	Garbage collects on all free-lists of all types.
H5get_libversion h5get_libversion_f	Returns the HDF library release number.
H5open h5open_f	Initializes the HDF5 library.
H5set_free_list_limits h5set_free_list_limits_f	Sets free-list size limits.
H5_VERSION_GE (none)	Determines whether the version of the library being used is greater than or equal to the specified version.
H5_VERSION_LE (none)	Determines whether the version of the library being used is less than or equal to the specified version.

Function Listing 4-2. File functions (H5F)

C Function Fortran Function	Purpose
H5Fclear_elink_file_cache (none)	Clears the external link open file cache for a file.
H5Fclose h5fclose_f	Closes HDF5 file.
H5Fcreate h5fcreate_f	Creates new HDF5 file.
H5Fflush h5fflush_f	Flushes data to HDF5 file on storage medium.
H5Fget_access_plist h5fget_access_plist_f	Returns a file access property list identifier.
H5Fget_create_plist h5fget_create_plist_f	Returns a file creation property list identifier.
H5Fget_file_image h5fget_file_image_f	Retrieves a copy of the image of an existing, open file.
H5Fget_filesize h5fget_filesize_f	Returns the size of an HDF5 file.
H5Fget_freespace h5fget_freespace_f	Returns the amount of free space in a file.
H5Fget_info (none)	Returns global information for a file.
H5Fget_intent (none)	Determines the read/write or read-only status of a file.
H5Fget_mdc_config (none)	Obtain current metadata cache configuration for target file.
H5Fget_mdc_hit_rate (none)	Obtain target file's metadata cache hit rate.
H5Fget_mdc_size (none)	Obtain current metadata cache size data for specified file.

Function Listing 4-2. File functions (H5F)

C Function Fortran Function	Purpose
H5Fget_mpi_atomicity h5fget_mpi_atomicity_f	Retrieves the atomicity mode in use.
H5Fget_name h5fget_name_f	Retrieves the name of the file to which the object belongs.
H5Fget_obj_count h5fget_obj_count_f	Returns the number of open object identifiers for an open file.
H5Fget_obj_ids h5fget_obj_ids_f	Returns a list of open object identifiers.
H5Fget_vfd_handle (none)	Returns pointer to the file handle from the virtual file driver.
H5Fis_hdf5 h5fis_hdf5_f	Determines whether a file is in the HDF5 format.
H5Fmount h5fmount_f	Mounts a file.
H5Fopen h5fopen_f	Opens existing HDF5 file.
H5Freopen h5freopen_f	Returns a new identifier for a previously-opened HDF5 file.
H5Freset_mdc_hit_rate_stats (none)	Reset hit rate statistics counters for the target file.
H5Fset_mdc_config (none)	Use to configure metadata cache of target file.
H5Fset_mpi_atomicity h5fset_mpi_atomicity_f	Use to set the MPI atomicity mode.
H5Funmount h5funmount_f	Unmounts a file.

Function Listing 4-3. File creation property list functions (H5P)

C Function Fortran Function	Purpose
H5Pset/get_userblock h5pset/get_userblock_f	Sets/retrieves size of user-block.
H5Pset/get_sizes h5pset/get_sizes_f	Sets/retrieves byte size of offsets and lengths used to address objects in HDF5 file.
H5Pset/get_sym_k h5pset/get_sym_k_f	Sets/retrieves size of parameters used to control symbol table nodes.
H5Pset/get_istore_k h5pset/get_istore_k_f	Sets/retrieves size of parameter used to control B-trees for indexing chunked datasets.
H5Pget_file_image h5pget_file_image_f	Retrieves a copy of the file image designated as the initial content and structure of a file.
H5Pset_file_image h5pset_file_image_f	Sets an initial file image in a memory buffer.
H5Pset_shared_mesg_nindexes h5pset_shared_mesg_nindexes_f	Sets number of shared object header message indexes.
H5Pget_shared_mesg_nindexes (none)	Retrieves number of shared object header message indexes in file creation property list.
H5Pset_shared_mesg_index h5pset_shared_mesg_index_f	Configures the specified shared object header message index.
H5Pget_shared_mesg_index (none)	Retrieves the configuration settings for a shared message index.
H5Pset_shared_mesg_phase_change (none)	Sets shared object header message storage phase change thresholds.
H5Pget_shared_mesg_phase_change (none)	Retrieves shared object header message phase change information.
H5Pget_version h5pget_version_f	Retrieves version information for various objects for file creation property list.

Function Listing 4-4. File access property list functions (H5P)

C Function Fortran Function	Purpose
H5Pset/get_alignment h5pset/get_alignment_f	Sets/retrieves alignment properties.
H5Pset/get_cache h5pset/get_cache_f	Sets/retrieves metadata cache and raw data chunk cache parameters.
H5Pset/get_elink_file_cache_size (none)	Sets/retrieves the size of the external link open file cache from the specified file access property list.
H5Pset/get_fclose_degree h5pset/get_fclose_degree_f	Sets/retrieves file close degree property.
H5Pset/get_gc_references h5pset/get_gc_references_f	Sets/retrieves garbage collecting references flag.
H5Pset_family_offset h5pset_family_offset_f	Sets offset property for low-level access to a file in a family of files.
H5Pget_family_offset (none)	Retrieves a data offset from the file access property list.
H5Pset/get_meta_block_size h5pset/get_meta_block_size_f	Sets the minimum metadata block size or retrieves the current metadata block size setting.
H5Pset_mdc_config (none)	Set the initial metadata cache configuration in the indicated File Access Property List to the supplied value.
H5Pget_mdc_config (none)	Get the current initial metadata cache configuration from the indicated File Access Property List.
H5Pset/get_sieve_buf_size h5pset/get_sieve_buf_size_f	Sets/retrieves maximum size of data sieve buffer.
H5Pset_libver_bounds h5pset_libver_bounds_f	Sets bounds on library versions, and indirectly format versions, to be used when creating objects.
H5Pget_libver_bounds (none)	Retrieves library version bounds settings that indirectly control the format versions used when creating objects.

Function Listing 4-4. File access property list functions (H5P)

C Function Fortran Function	Purpose
H5Pset_small_data_block_size h5pset_small_data_block_size_f	Sets the size of a contiguous block reserved for small data.
H5Pget_small_data_block_size h5pget_small_data_block_size_f	Retrieves the current small data block size setting.

Function Listing 4-5. File driver functions (H5P)

C Function Fortran Function	Purpose
H5Pset_driver (none)	Sets a file driver.
H5Pget_driver h5pget_driver_f	Returns the identifier for the driver used to create a file.
H5Pget_driver_info (none)	Returns a pointer to file driver information.
H5Pset/get_fapl_core h5pset/get_fapl_core_f	Sets driver for buffered memory files (i.e., in RAM) or retrieves information regarding driver.
H5Pset_fapl_direct h5pset_fapl_direct_f	Sets up use of the direct I/O driver.
H5Pget_fapl_direct h5pget_fapl_direct_f	Retrieves direct I/O driver settings.
H5Pset/get_fapl_family h5pset/get_fapl_family_f	Sets driver for file families, designed for systems that do not support files larger than 2 gigabytes, or retrieves information regarding driver.
H5Pset_fapl_log (none)	Sets logging driver.
H5Pset/get_fapl_mpio h5pset/get_fapl_mpio_f	Sets driver for files on parallel file systems (MPI I/O) or retrieves information regarding the driver.

Function Listing 4-5. File driver functions (H5P)

C Function Fortran Function	Purpose
H5Pset_fapl_mpio h5pset_fapl_mpio_f	No longer available.
H5Pget_fapl_mpio h5pget_fapl_mpio_f	No longer available.
H5Pset/get_fapl_multi h5pset/get_fapl_multi_f	Sets driver for multiple files, separating categories of metadata and raw data, or retrieves information regarding driver.
H5Pset_fapl_sec2 h5pset_fapl_sec2_f	Sets driver for unbuffered permanent files or retrieves information regarding driver.
H5Pset_fapl_split h5pset_fapl_split_f	Sets driver for split files, a limited case of multiple files with one metadata file and one raw data file.
H5Pset_fapl_stdio H5Pset_fapl_stdio_f	Sets driver for buffered permanent files.
H5Pset_fapl_windows (none)	Sets the Windows I/O driver.
H5Pset_multi_type (none)	Specifies type of data to be accessed via the MULTI driver enabling more direct access.
H5Pget_multi_type (none)	Retrieves type of data property for MULTI driver.

4.5. Creating or Opening an HDF5 File

This section describes in more detail how to create and how to open files.

New HDF5 files are created and opened with `H5Fcreate`; existing files are opened with `H5Fopen`. Both functions return an object identifier which must eventually be released by calling `H5Fclose`.

To create a new file, call `H5Fcreate`:

```
hid_t H5Fcreate (const char *name, unsigned flags, hid_t fcpl_id,
               hid_t fapl_id)
```

`H5Fcreate` creates a new file named *name* in the current directory. The file is opened with read and write access; if the `H5F_ACC_TRUNC` flag is set, any pre-existing file of the same name in the same directory is truncated. If `H5F_ACC_TRUNC` is not set or `H5F_ACC_EXCL` is set and if a file of the same name exists, `H5Fcreate` will fail.

The new file is created with the properties specified in the property lists *fcpl_id* and *facpl_id*. *fcpl* is short for file creation property list. *facpl* is short for file access property list. Specifying `H5P_DEFAULT` for either the creation or access property list calls for the library's default creation or access properties.

For more information, see "[File Property Lists](#)" on page 123. For more information, see "[File Access Modes](#)" on page 109.

If `H5Fcreate` successfully creates the file, it returns a file identifier for the new file. This identifier will be used by the application any time an object identifier, an OID, for the file is required. Once the application has finished working with a file, the identifier should be released and the file closed with `H5Fclose`.

To open an existing file, call `H5Fopen`:

```
hid_t H5Fopen (const char *name, unsigned flags, hid_t facpl_id)
```

`H5Fopen` opens an existing file with read-write access if `H5F_ACC_RDWR` is set and read-only access if `H5F_ACC_RDONLY` is set.

facpl_id is the file access property list identifier. Alternatively, `H5P_DEFAULT` indicates that the application relies on the default I/O access parameters. Creating and changing access property lists is documented further below.

A file can be opened more than once via multiple `H5Fopen` calls. Each such call returns a unique file identifier and the file can be accessed through any of these file identifiers as long as they remain valid. Each of these file identifiers must be released by calling `H5Fclose` when it is no longer needed.

4.6. Closing an HDF5 File

`H5Fclose` both closes a file and releases the file identifier returned by `H5Fopen` or `H5Fcreate`. `H5Fclose` must be called when an application is done working with a file; while the HDF5 Library makes every effort to maintain file integrity, failure to call `H5Fclose` may result in the file being abandoned in an incomplete or corrupted state.

To close a file, call `H5Fclose`:

```
herr_t H5Fclose (hid_t file_id)
```

This function releases resources associated with an open file. After closing a file, the file identifier, *file_id*, cannot be used again as it will be undefined.

`H5Fclose` fulfills three purposes: to ensure that the file is left in an uncorrupted state, to ensure that all data has been written to the file, and to release resources. Use [H5Fflush](#) if you wish to ensure that all data has been written to the file but it is premature to close it.

Note regarding serial mode behavior: When `H5Fclose` is called in serial mode, it closes the file and terminates new access to it, but it does not terminate access to objects that remain individually open within the file. That is, if `H5Fclose` is called for a file but one or more objects within the file remain open, those objects will remain accessible until they are individually closed. To illustrate, assume that a file, `fileA`, contains a dataset, `data_setA`, and that both are open when `H5Fclose` is called for `fileA`. `data_setA` will remain open and accessible, including writable, until it is explicitly closed. The file will be automatically and finally closed once all objects within it have been closed.

Note regarding parallel mode behavior: Once `H5Fclose` has been called in parallel mode, access is no longer available to any object within the file.

4.7. File Property Lists

Additional information regarding file structure and access are passed to `H5Fcreate` and `H5Fopen` through property list objects. Property lists provide a portable and extensible method of modifying file properties via simple API functions. There are two kinds of file-related property lists:

- File creation property lists
- File access property lists

In the following sub-sections, we discuss only one file creation property, user-block size, in detail as a model for the user. Other file creation and file access properties are mentioned and defined briefly, but the model is not expanded for each; complete syntax, parameter, and usage information for every property list function is provided in the "[H5P: Property List Interface](#)" section of the [HDF5 Reference Manual](#). For more information, see "[Properties and Property Lists in HDF5](#)" on page 839.

4.7.1. Creating a Property List

If you do not wish to rely on the default file creation and access properties, you must first create a property list with `H5Pcreate`.

```
hid_t H5Pcreate (hid_t cls_id)
```

`type` is the type of property list being created. In this case, the appropriate values are `H5P_FILE_CREATE` for a file creation property list and `H5P_FILE_ACCESS` for a file access property list.

Thus, the following calls create a file creation property list and a file access property list with identifiers `fcpl_id` and `fapl_id`, respectively:

```
fcpl_id = H5Pcreate (H5P_FILE_CREATE)
fapl_id = H5Pcreate (H5P_FILE_ACCESS)
```

Once the property lists have been created, the properties themselves can be modified via the functions described in the following sub-sections.

4.7.2. File Creation Properties

File creation property lists control the file metadata, which is maintained in the superblock of the file. These properties are used only when a file is first created.

User-block Size

```
herr_t H5Pset_userblock (hid_t plist, hsize_t size)
herr_t H5Pget_userblock (hid_t plist, hsize_t *size)
```

The *user-block* is a fixed-length block of data located at the beginning of the file and is ignored by the HDF5 Library. This block is specifically set aside for any data or information that developers determine to be useful to their applications but that will not be used by the HDF5 Library. The *size* of the user-block is defined in bytes and may be set to any power of two with a minimum size of 512 bytes. In other words, user-blocks might be 512, 1024, or 2048 bytes in size.

This property is set with `H5Pset_userblock` and queried via `H5Pget_userblock`. For example, if an application needed a 4K user-block, then the following function call could be used:

```
status = H5Pset_userblock(fcpl_id, 4096)
```

The property list could later be queried with

```
status = H5Pget_userblock(fcpl_id, size)
```

and the value 4096 would be returned in the parameter *size*.

Other properties, described below, are set and queried in exactly the same manner. Syntax and usage are detailed in the "[H5P: Property List Interface](#)" section of the [HDF5 Reference Manual](#).

Offset and Length Sizes

This property specifies the number of bytes used to store the offset and length of objects in the HDF5 file. Values of 2, 4, and 8 bytes are currently supported to accommodate 16-bit, 32-bit, and 64-bit file address spaces.

These properties are set and queried via `H5Pset_sizes` and `H5Pget_sizes`.

Symbol Table Parameters

The size of symbol table B-trees can be controlled by setting the 1/2-rank and 1/2-node size parameters of the B-tree.

These properties are set and queried via `H5Pset_sym_k` and `H5Pget_sym_k`.

Indexed Storage Parameters

The size of indexed storage B-trees can be controlled by setting the 1/2-rank and 1/2-node size parameters of the B-tree.

These properties are set and queried via `H5Pset_istore_k` and `H5Pget_istore_k`.

Version Information

Various objects in an HDF5 file may over time appear in different versions. The HDF5 Library keeps track of the version of each object in the file.

Version information is retrieved via `H5Pget_version`.

4.7.3. File Access Properties

This section discusses file access properties that are not related to the low-level file drivers. File drivers are discussed separately later in this chapter. For more information, see "[Alternate File Storage Layouts and Low-level File Drivers](#)" on page 126.

File access property lists control various aspects of file I/O and structure.

Data Alignment

Sometimes file access is faster if certain data elements are aligned in a specific manner. This can be controlled by setting alignment properties via the `H5Pset_alignment` function. There are two values involved:

- A threshold value
- An alignment interval

Any allocation request at least as large as the threshold will be aligned on an address that is a multiple of the alignment interval.

Metadata Block Allocation Size

Metadata typically exists as very small chunks of data; storing metadata elements in a file without blocking them can result in hundreds or thousands of very small data elements in the file. This can result in a highly fragmented file and seriously impede I/O. By blocking metadata elements, these small elements can be grouped in larger sets, thus alleviating both problems.

`H5Pset_meta_block_size` sets the minimum size in bytes of metadata block allocations.

`H5Pget_meta_block_size` retrieves the current minimum metadata block allocation size.

Metadata Cache

Metadata and raw data I/O speed are often governed by the size and frequency of disk reads and writes. In many cases, the speed can be substantially improved by the use of an appropriate cache.

`H5Pset_cache` sets the minimum cache size for both metadata and raw data and a preemption value for raw data chunks. `H5Pget_cache` retrieves the current values.

Data Sieve Buffer Size

Data sieve buffering is used by certain file drivers to speed data I/O and is most commonly when working with dataset hyperslabs. For example, using a buffer large enough to hold several pieces of a dataset as it is read in for hyperslab selections will boost performance noticeably.

`H5Pset_sieve_buf_size` sets the maximum size in bytes of the data sieve buffer.

`H5Pget_sieve_buf_size` retrieves the current maximum size of the data sieve buffer.

Garbage Collection References

Dataset region references and other reference types use space in an HDF5 file's global heap. If garbage collection is on (1) and the user passes in an uninitialized value in a reference structure, the heap might become corrupted. When garbage collection is off (0), however, and the user re-uses a reference, the previous heap block will be orphaned and not returned to the free heap space. When garbage collection is on, the user must initialize the reference structures to 0 or risk heap corruption.

`H5Pset_gc_references` sets the garbage collecting references flag.

4.8. Alternate File Storage Layouts and Low-level File Drivers

The concept of an HDF5 file is actually rather abstract: the address space for what is normally thought of as an HDF5 file might correspond to any of the following:

- Single file on standard file system
- Multiple files on standard file system
- Multiple files on parallel file system
- Block of memory within application's memory space
- More abstract situations such as virtual files

This HDF5 address space is generally referred to as an *HDF5 file* regardless of its organization at the storage level.

HDF5 employs an extremely flexible mechanism called the *virtual file layer*, or VFL, for file I/O. A full understanding of the VFL is only necessary if you plan to write your own drivers (see "[Virtual File Layer](#)" and "[List of VFL Functions](#)" in the *HDF5 Technical Notes*). For our purposes here, it is sufficient to know that the low-level drivers used for file I/O reside in the VFL, as illustrated in the following figure. Note that `H5FD_STREAM` is not available with 1.8.x and later versions of the library.

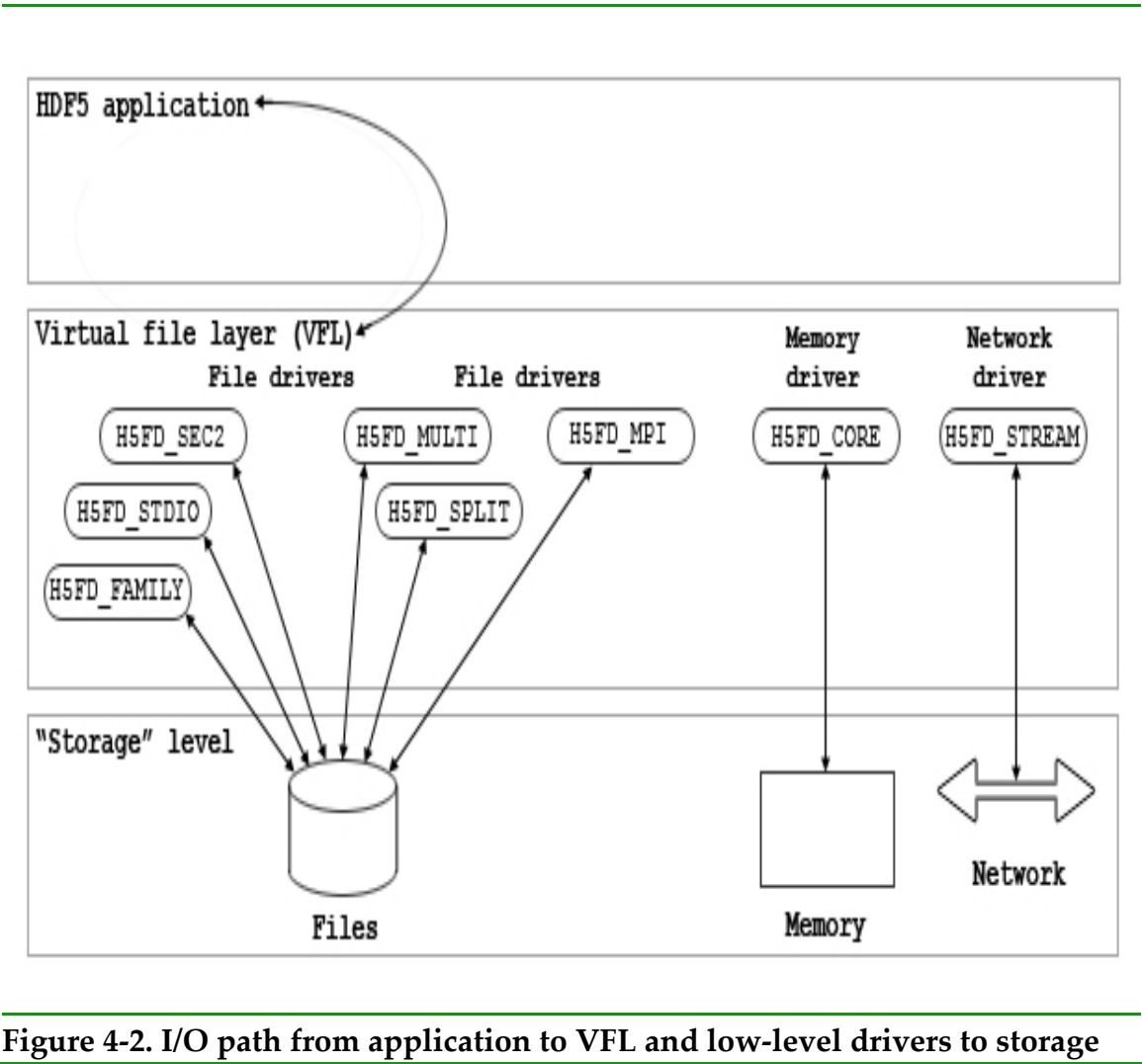


Figure 4-2. I/O path from application to VFL and low-level drivers to storage

As mentioned above, HDF5 applications access HDF5 files through various *low-level file drivers*. The default driver for that layout is the POSIX driver (also known as the SEC2 driver), `H5FD_SEC2`. Alternative layouts and drivers are designed to suit the needs of a variety of systems, environments, and applications. The drivers are listed in the table below.

Table 4-2. Supported file drivers

Driver Name	Driver Identifier	Description	Related API
POSIX	H5FD_SEC2	This driver uses POSIX file-system functions like read and write to perform I/O to a single, permanent file on local disk with no system buffering. This driver is POSIX-compliant and is the default file driver for all systems.	H5Pset_fapl_sec2
Direct	H5FD_DIRECT	This is the H5FD_SEC2 driver except data is written to or read from the file synchronously without being cached by the system.	H5Pset_fapl_direct
Log	H5FD_LOG	This is the H5FD_SEC2 driver with logging capabilities.	H5Pset_fapl_log
Windows	H5FD_WINDOWS	This driver was modified in HDF5-1.8.8 to be a wrapper of the POSIX driver, H5FD_SEC2. This change should not affect user applications.	H5Pset_fapl_windows
STDIO	H5FD_STDIO	This driver uses functions from the standard C <code>stdio.h</code> to perform I/O to a single, permanent file on local disk with additional system buffering.	H5Pset_fapl_stdio

Table 4-2. Supported file drivers

Driver Name	Driver Identifier	Description	Related API
Memory	H5FD_CORE	With this driver, an application can work with a file in memory for faster reads and writes. File contents are kept in memory until the file is closed. At closing, the memory version of the file can be written back to disk or abandoned.	H5Pset_fapl_core
Family	H5FD_FAMILY	With this driver, the HDF5 file's address space is partitioned into pieces and sent to separate storage files using an underlying driver of the user's choice. This driver is for systems that do not support files larger than 2 gigabytes.	H5Pset_fapl_family
Multi	H5FD_MULTI	With this driver, data can be stored in multiple files according to the type of the data. I/O might work better if data is stored in separate files based on the type of data. The Split driver is a special case of this driver.	H5Pset_fapl_multi
Split	H5FD_SPLIT	This file driver splits a file into two parts. One part stores metadata, and the other part stores raw data. This splitting a file into two parts is a limited case of the Multi driver.	H5Pset_fapl_split
Parallel	H5FD_MPIO	This is the standard HDF5 file driver for parallel file systems. This driver uses the MPI standard for both communication and file I/O.	H5Pset_fapl_mpio

Table 4-2. Supported file drivers

Driver Name	Driver Identifier	Description	Related API
Parallel POSIX	H5FD_MPIOPOSIX	This driver is no longer available.	
Stream	H5FD_STREAM	This driver is no longer available.	

For more information, see the [HDF5 Reference Manual](#) entries for the function calls shown in the column on the right in the table above.

Note that the low-level file drivers manage alternative *file* storage layouts. *Dataset* storage layouts (chunking, compression, and external dataset storage) are managed independently of file storage layouts.

If an application requires a special-purpose low-level driver, the VFL provides a public API for creating one. For more information on how to create a driver, see "[Virtual File Layer](#)" and "[List of VFL Functions](#)" in the *HDF5 Technical Notes*.

4.8.1. Identifying the Previously-used File Driver

When creating a new HDF5 file, no history exists, so the file driver must be specified if it is to be other than the default.

When opening existing files, however, the application may need to determine which low-level driver was used to create the file. The function `H5Pget_driver` is used for this purpose. See the example below.

```
hid_t H5Pget_driver (hid_t fapl_id)
```

Code Example 4-5. Identifying a driver

`H5Pget_driver` returns a constant identifying the low-level driver for the access property list `fapl_id`. For example, if the file was created with the POSIX (aka SEC2) driver, `H5Pget_driver` returns `H5FD_SEC2`.

If the application opens an HDF5 file without both determining the driver used to create the file and setting up the use of that driver, the HDF5 Library will examine the superblock and the driver definition block to identify the driver. See the [HDF5 File Format Specification](#) for detailed descriptions of the superblock and the driver definition block.

4.8.2. The POSIX (aka SEC2) Driver

The POSIX driver, `H5FD_SEC2`, uses functions from section 2 of the POSIX manual to access unbuffered files stored on a local file system. This driver is also known as the SEC2 driver. The HDF5 Library buffers metadata regardless of the low-level driver, but using this driver prevents data from being buffered again by the lowest layers of the library.

The function `H5Pset_fapl_sec2` sets the file access properties to use the POSIX driver. See the example below.

```
herr_t H5Pset_fapl_sec2 (hid_t fapl_id)
```

Code Example 4-6. Using the POSIX, aka SEC2, driver

Any previously-defined driver properties are erased from the property list.

Additional parameters may be added to this function in the future. Since there are no additional variable settings associated with the POSIX driver, there is no `H5Pget_fapl_sec2` function.

4.8.3. The Direct Driver

The Direct driver, `H5FD_DIRECT`, functions like the POSIX driver except that data is written to or read from the file synchronously without being cached by the system.

The functions `H5Pset_fapl_direct` and `H5Pget_fapl_direct` are used to manage file access properties. See the example below.

```
herr_t H5Pset_fapl_direct( hid_t fapl_id, size_t alignment,
                          size_t block_size, size_t cbuf_size )

herr_t H5Pget_fapl_direct( hid_t fapl_id, size_t *alignment,
                          size_t *block_size, size_t *cbuf_size )
```

Code Example 4-7. Using the Direct driver

H5Pset_fapl_direct sets the file access properties to use the Direct driver; any previously defined driver properties are erased from the property list. H5Pget_fapl_direct retrieves the file access properties used with the Direct driver. fapl_id is the file access property list identifier. alignment is the memory alignment boundary. block_size is the file system block size. cbuf_size is the copy buffer size.

Additional parameters may be added to this function in the future.

4.8.4. The Log Driver

The Log driver, H5FD_LOG, is designed for situations where it is necessary to log file access activity.

The function H5Pset_fapl_log is used to manage logging properties. See the example below.

```
herr_t H5Pset_fapl_log (hid_t fapl_id, const char *logfile,
                      unsigned int flags, size_t buf_size)
```

Code Example 4-8. Logging file access

H5Pset_fapl_log sets the file access property list to use the Log driver. File access characteristics are identical to access via the POSIX driver. Any previously defined driver properties are erased from the property list.

Log records are written to the file logfile.

The logging levels set with the verbosity parameter are shown in the table below.

Table 4-3. Logging levels

Level	Comments
0	Performs no logging.
1	Records where writes and reads occur in the file.
2	Records where writes and reads occur in the file and what kind of data is written at each location. This includes raw data or any of several types of metadata (object headers, superblock, B-tree data, local headers, or global headers).

There is no H5Pget_fapl_log function.

Additional parameters may be added to this function in the future.

4.8.5. The Windows Driver

The Windows driver, `H5FD_WINDOWS`, was modified in HDF5-1.8.8 to be a wrapper of the POSIX driver, `H5FD_SEC2`. In other words, if the Windows drivers is used, any file I/O will instead use the functionality of the POSIX driver. This change should be transparent to all user applications. The Windows driver used to be the default driver for Windows systems. The POSIX driver is now the default.

The function `H5Pset_fapl_windows` sets the file access properties to use the Windows driver. See the example below.

```
herr_t H5Pset_fapl_windows (hid_t fapl_id)
```

Code Example 4-9. Using the Windows driver

Any previously-defined driver properties are erased from the property list.

Additional parameters may be added to this function in the future. Since there are no additional variable settings associated with the POSIX driver, there is no `H5Pget_fapl_windows` function.

4.8.6. The STDIO Driver

The STDIO driver, `H5FD_STDIO`, accesses permanent files in a local file system like the POSIX driver does. The STDIO driver also has an additional layer of buffering beneath the HDF5 Library.

The function `H5Pset_fapl_stdio` sets the file access properties to use the STDIO driver. See the example below.

```
herr_t H5Pset_fapl_stdio (hid_t fapl_id)
```

Code Example 4-10. Using the STDIO driver

Any previously defined driver properties are erased from the property list.

Additional parameters may be added to this function in the future. Since there are no additional variable settings associated with the STDIO driver, there is no `H5Pget_fapl_stdio` function.

4.8.7. The Memory (aka Core) Driver

There are several situations in which it is reasonable, sometimes even required, to maintain a file entirely in system memory. You might want to do so if, for example, either of the following conditions apply:

- Performance requirements are so stringent that disk latency is a limiting factor
- You are working with small, temporary files that will not be retained and, thus, need not be written to storage media

The Memory driver, `H5FD_CORE`, provides a mechanism for creating and managing such in-memory files. The functions `H5Pset_fapl_core` and `H5Pget_fapl_core` manage file access properties. See the example below.

```
herr_t H5Pset_fapl_core (hid_t access_properties,
                        size_t block_size, hbool_t backing_store)
herr_t H5Pget_fapl_core (hid_t access_properties,
                        size_t *block_size), hbool_t *backing_store)
```

Code Example 4-11. Managing file access for in-memory files

`H5Pset_fapl_core` sets the file access property list to use the Memory driver; any previously defined driver properties are erased from the property list.

Memory for the file will always be allocated in units of the specified `block_size`.

The `backing_store` Boolean flag is set when the in-memory file is created. `backing_store` indicates whether to write the file contents to disk when the file is closed. If `backing_store` is set to 1 (TRUE), the file contents are flushed to a file with the same name as the in-memory file when the file is closed or access to the file is terminated in memory. If `backing_store` is set to 0 (FALSE), the file is not saved.

The application is allowed to open an existing file with the `H5FD_CORE` driver. While using `H5Fopen` to open an existing file, if `backing_store` is set to 1 and the `flag` for `H5Fopen` is set to `H5F_ACC_RDWR`, changes to the file contents will be saved to the file when the file is closed. If `backing_store` is set to 0 and the `flag` for `H5Fopen` is set to `H5F_ACC_RDWR`, changes to the file contents will be lost when the file is closed. If the `flag` for `H5Fopen` is set to `H5F_ACC_RDONLY`, no change to the file will be allowed either in memory or on file.

If the file access property list is set to use the Memory driver, `H5Pget_fapl_core` will return `block_size` and `backing_store` with the relevant file access property settings.

Note the following important points regarding in-memory files:

- Local temporary files are created and accessed directly from memory without ever being written to disk

- Total file size must not exceed the available virtual memory
- Only one HDF5 file identifier can be opened for the file, the identifier returned by `H5Fcreate` or `H5Fopen`
- The changes to the file will be discarded when access is terminated unless `backing_store` is set to 1

Additional parameters may be added to these functions in the future.

See the "[HDF5 File Image Operations](#)" section for information on more advanced usage of the Memory file driver, and see the "[Modified Region Writes](#)" section for information on how to set write operations so that only modified regions are written to storage.

4.8.8. The Family Driver

HDF5 files can become quite large, and this can create problems on systems that do not support files larger than 2 gigabytes. The HDF5 **file family** mechanism is designed to solve the problems this creates by splitting the HDF5 file address space across several smaller files. This structure does not affect how meta-data and raw data are stored: they are mixed in the address space just as they would be in a single, contiguous file.

HDF5 applications access a family of files via the Family driver, `H5FD_FAMILY`. The functions `H5Pset_fapl_family` and `H5Pget_fapl_family` are used to manage file family properties. See the example below.

```
herr_t H5Pset_fapl_family (hid_t fapl_id, hsize_t memb_size,
                          hid_t member_properties)

herr_t H5Pget_fapl_family (hid_t fapl_id, hsize_t *memb_size,
                          hid_t *member_properties)
```

Code Example 4-12. Managing file family properties

Each member of the family is the same *logical* size though the size and disk storage reported by file system listing tools may be substantially smaller. Examples of file system listing tools are 'ls -l' on a Unix system or the detailed folder listing on an Apple Macintosh or Microsoft Windows system. The name passed to `H5Fcreate` or `H5Fopen` should include a `printf(3c)`-style integer format specifier which will be replaced with the family member number. The first family member is numbered zero (0).

`H5Pset_fapl_family` sets the access properties to use the Family driver; any previously defined driver properties are erased from the property list. `member_properties` will serve as the file access property

list for each member of the file family. *memb_size* specifies the logical size, in bytes, of each family member. *memb_size* is used only when creating a new file or truncating an existing file; otherwise the member size is determined by the size of the first member of the family being opened. Note: If the size of the *off_t* type is four bytes, the maximum family member size is usually $2^{31}-1$ because the byte at offset 2,147,483,647 is generally inaccessible.

`H5Pget_fapl_family` is used to retrieve file family properties. If the file access property list is set to use the Family driver, *member_properties* will be returned with a pointer to a copy of the appropriate member access property list. If *memb_size* is non-null, it will contain the logical size, in bytes, of family members.

Additional parameters may be added to these functions in the future.

4.8.8.1. Unix Tools and an HDF5 Utility

It occasionally becomes necessary to **repartition** a file family. A command-line utility for this purpose, `h5repart`, is distributed with the HDF5 Library.

```
h5repart [-v] [-b block_size[suffix]] [-m member_size[suffix]] source
destination
```

`h5repart` repartitions an HDF5 file by copying the source file or file family to the destination file or file family, preserving holes in the underlying UNIX files. Families are used for the source and/or destination if the name includes a `printf`-style integer format such as `%d`. The `-v` switch prints input and output file names on the standard error stream for progress monitoring, `-b` sets the I/O block size (the default is 1KB), and `-m` sets the output member size if the destination is a family name (the default is 1GB). *block_size* and *member_size* may be suffixed with the letters *g*, *m*, or *k* for GB, MB, or KB respectively.

The `h5repart` utility is described on the [Tools](#) page of the [HDF5 Reference Manual](#).

An existing HDF5 file can be split into a family of files by running the file through `split(1)` on a UNIX system and numbering the output files. However, the HDF5 Library is lazy about extending the size of family members, so a valid file cannot generally be created by concatenation of the family members.

Splitting the file and rejoining the segments by concatenation (`split(1)` and `cat(1)` on UNIX systems) does not generate files with holes; holes are preserved only through the use of `h5repart`.

4.8.9. The Multi Driver

In some circumstances, it is useful to separate metadata from raw data and some types of metadata from other types of metadata. Situations that would benefit from use of the Multi driver include the following:

- In networked situations where the small metadata files can be kept on local disks but larger raw data files must be stored on remote media
- In cases where the raw data is extremely large

- In situations requiring frequent access to metadata held in RAM while the raw data can be efficiently held on disk

In either case, access to the metadata is substantially easier with the smaller, and possibly more localized, metadata files. This often results in improved application performance.

The Multi driver, `H5FD_MULTI`, provides a mechanism for segregating raw data and different types of metadata into multiple files. The functions `H5Pset_fapl_multi` and `H5Pget_fapl_multi` are used to manage access properties for these multiple files. See the example below.

```
herr_t H5Pset_fapl_multi (hid_t fapl_id, const H5FD_mem_t *memb_map,
                        const hid_t *memb_fapl, const char * const *memb_name,
                        const haddr_t *memb_addr, hbool_t relax)
herr_t H5Pget_fapl_multi (hid_t fapl_id, const H5FD_mem_t *memb_map,
                        const hid_t *memb_fapl, const char **memb_name,
                        const haddr_t *memb_addr, hbool_t *relax)
```

Code Example 4-13. Managing access properties for multiple files

`H5Pset_fapl_multi` sets the file access properties to use the Multi driver; any previously defined driver properties are erased from the property list. With the Multi driver invoked, the application will provide a base name to `H5Fopen` or `H5Fcreate`. The files will be named by that base name as modified by the rule indicated in `memb_name`. File access will be governed by the file access property list `memb_properties`.

See [H5Pset_fapl_multi](#) and [H5Pget_fapl_multi](#) in the *HDF5 Reference Manual* for descriptions of these functions and their usage.

Additional parameters may be added to these functions in the future.

4.8.10. The Split Driver

The Split driver, `H5FD_SPLIT`, is a limited case of the Multi driver where only two files are created. One file holds metadata, and the other file holds raw data.

The function `H5Pset_fapl_split` is used to manage Split file access properties. See the example below.

```
herr_t H5Pset_fapl_split (hid_t access_properties,
                        const char *meta_extension, hid_t meta_properties,
                        const char *raw_extension, hid_t raw_properties)
```

Code Example 4-14. Managing access properties for split files

`H5Pset_fapl_split` sets the file access properties to use the Split driver; any previously defined driver properties are erased from the property list.

With the Split driver invoked, the application will provide a base file name such as *file_name* to `H5Fcreate` or `H5Fopen`. The metadata and raw data files in storage will then be named *file_name.meta_extension* and *file_name.raw_extension*, respectively. For example, if *meta_extension* is defined as `.meta` and *raw_extension* is defined as `.raw`, the final filenames will be *file_name.meta* and *file_name.raw*.

Each file can have its own file access property list. This allows the creative use of other low-level file drivers. For instance, the metadata file can be held in RAM and accessed via the Memory driver while the raw data file is stored on disk and accessed via the POSIX driver. Metadata file access will be governed by the file access property list in *meta_properties*. Raw data file access will be governed by the file access property list in *raw_properties*.

Additional parameters may be added to these functions in the future. Since there are no additional variable settings associated with the Split driver, there is no `H5Pget_fapl_split` function.

4.8.11. The Parallel Driver

Parallel environments require a parallel low-level driver. HDF5's default driver for parallel systems is called the Parallel driver, `H5FD_MPIO`. This driver uses the MPI standard for both communication and file I/O.

The functions `H5Pset_fapl_mpio` and `H5Pget_fapl_mpio` are used to manage file access properties for the `H5FD_MPIO` driver. See the example below.

```
herr_t H5Pset_fapl_mpio (hid_t fapl_id, MPI_Comm comm,
                        MPI_info info)
herr_t H5Pget_fapl_mpio (hid_t fapl_id, MPI_Comm *comm,
                        MPI_info *info)
```

Code Example 4-15. Managing parallel file access properties

The file access properties managed by `H5Pset_fapl_mpio` and retrieved by `H5Pget_fapl_mpio` are the MPI communicator, `comm`, and the MPI info object, `info`. `comm` and `info` are used for file open. `info` is an information object much like an HDF5 property list. Both are defined in `MPI_FILE_OPEN` of MPI-2.

The communicator and the info object are saved in the file access property list `fapl_id`. `fapl_id` can then be passed to `MPI_FILE_OPEN` to create and/or open the file.

`H5Pset_fapl_mpio` and `H5Pget_fapl_mpio` are available only in the parallel HDF5 Library and are not collective functions. The Parallel driver is available only in the parallel HDF5 Library.

Additional parameters may be added to these functions in the future.

4.9. Code Examples for Opening and Closing Files

4.9.1. Example Using the `H5F_ACC_TRUNC` Flag

The following example uses the `H5F_ACC_TRUNC` flag when it creates a new file. The default file creation and file access properties are also used. Using `H5F_ACC_TRUNC` means the function will look for an existing file with the name specified by the function. In this case, that name is `FILE`. If the function does not find an existing file, it will create one. If it does find an existing file, it will empty the file in preparation for a new set of data. The identifier for the "new" file will be passed back to the application program. For more information, see "[File Access Modes](#)" on page 109.

```
hid_t file; /* identifier */

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

/* Close the file. */
status = H5Fclose(file);
```

Code Example 4-16. Creating a file with default creation and access properties

4.9.2. Example with the File Creation Property List

The example below shows how to create a file with 64-bit object offsets and lengths.

```
hid_t create_plist;
hid_t file_id;
create_plist = H5Pcreate(H5P_FILE_CREATE);
H5Pset_sizes(create_plist, 8, 8);
file_id = H5Fcreate("test.h5", H5F_ACC_TRUNC,
    create_plist, H5P_DEFAULT);
    .
    .
    .
H5Fclose(file_id);
```

Code Example 4-17. Creating a file with 64-bit offsets

4.9.3. Example with File Access Property List

This example shows how to open an existing file for independent datasets access by MPI parallel I/O:

```
hid_t access_plist;
hid_t file_id;
access_plist = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpi(access_plist, MPI_COMM_WORLD, MPI_INFO_NULL);

/* H5Fopen must be called collectively */
file_id = H5Fopen("test.h5", H5F_ACC_RDWR, access_plist);
.
.
.

/* H5Fclose must be called collectively */
H5Fclose(file_id);
```

Code Example 4-18. Opening an existing file for parallel I/O

4.10. Working with Multiple HDF5 Files

Multiple HDF5 files can be associated so that the files can be worked with as though all the information is in a single HDF5 file. A temporary association can be set up by means of the `H5Fmount` function. A permanent association can be set up by means of the external link function `H5Lcreate_external`.

The purpose of this section is to describe what happens when the `H5Fmount` function is used to mount one file on another.

When a file is mounted on another, the mounted file is mounted at a group, and the root group of the mounted file takes the place of that group until the mounted file is unmounted or until the files are closed.

The figure below shows two files before one is mounted on the other. File1 has two groups and three datasets. The group that is the target of the A link has links, Z and Y, to two of the datasets. The group that is the target of the B link has a link, W, to the other dataset. File2 has three groups and three datasets. The

groups in File2 are the targets of the AA, BB, and CC links. The datasets in File2 are the targets of the ZZ, YY, and WW links.

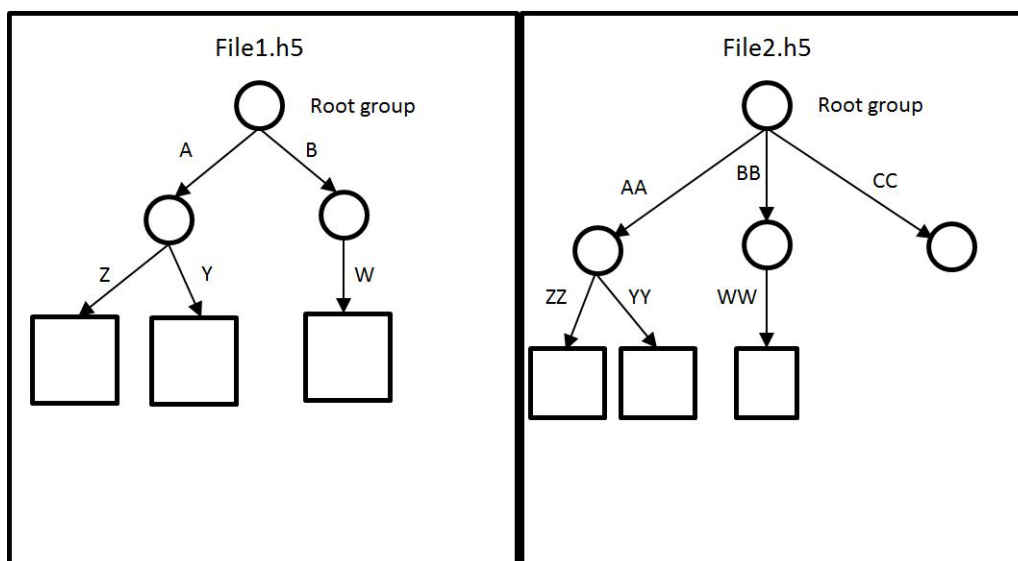


Figure 4-3. Two separate files

The figure below shows the two files after File2 has been mounted File1 at the group that is the target of the B link.

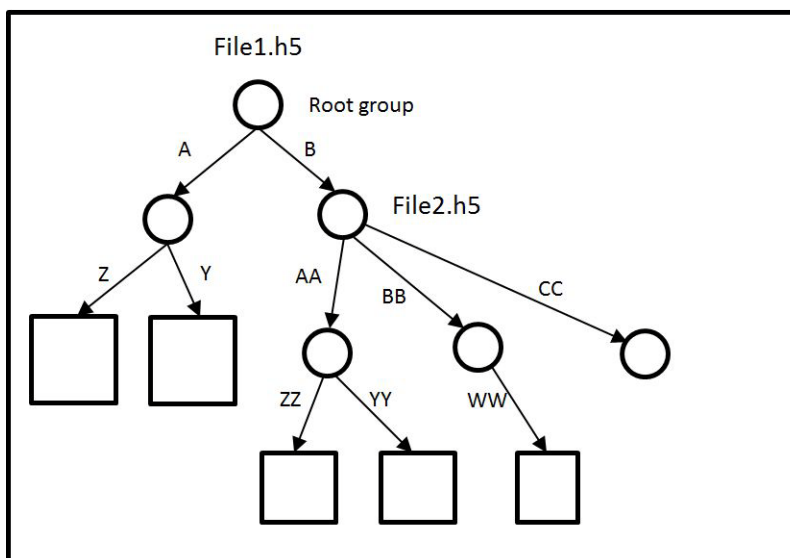


Figure 4-4. File2 mounted on File1

Note: Note that the dataset that is the target of the W link is not shown in the figure above. That dataset is masked by the mounted file.

If a file is mounted on a group that has members, those members are hidden until the mounted file is unmounted. There are two ways around this if you need to work with a group member. One is to mount the file on an empty group. Another is to open the group member before you mount the file. Opening the group member will return an identifier that you can use to locate the group member.

The example below shows how `H5Fmount` might be used to mount File2 onto File1.

```
status = H5Fmount(loc_id, "/B", child_id, plist_id)
```

Code Example 4-19. Using H5Fmount

Note: `loc_id` is the file identifier for File1, `/B` is the link path to the group where File2 is mounted, `child_id` is the file identifier for File2, and `plist_id` is a property list identifier.

For more information, see "[HDF5 Groups](#)" on page 152. , and the [H5Fmount](#), [H5Funmount](#), and [H5Lcreate_external](#) functions in the [HDF5 Reference Manual](#).

```

<!doctype HTML public "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
<head>
<title>Chapter 4: HDF5 Groups</title>

<!--(Meta)=====-->

<!--(Links)=====-->

<!--( Begin styles definition )=====-->
<link href="ed_styles/NewUGelect.css" rel="stylesheet" type="text/css">
<!--( End styles definition )=====-->

</head>

<body>

<!-- #BeginLibraryItem "/ed_libs/styles_UG.lbi" -->
<!--
* * * * *
* Copyright by The HDF Group. *
* Copyright by the Board of Trustees of the University of Illinois. *
* All rights reserved. *
*
* This file is part of HDF5. The full HDF5 copyright notice, including *
* terms governing use, modification, and redistribution, is contained in *
* the files COPYING and Copyright.html. COPYING can be found at the root *
* of the source code distribution tree; Copyright.html can be found at the *
* root level of an installed copy of the electronic HDF5 document set and *
* is linked from the top-level documents page. It can also be found at *
* http://hdfgroup.org/HDF5/doc/Copyright.html. If you do not have *

```

```

* access to either file, you may request a copy from help@hdfgroup.org.  *
* * * * *
-->
<!-- #EndLibraryItem --><!-- HEADER LEFT "HDF5 User's Guide" -->
<!-- HEADER RIGHT "HDF5 Groups" -->

<!-- ( TOC )=====-->
<SCRIPT language="JavaScript">
<!--
document.writeln ('\
<table x-use-null-cells\
        align=right\
width=240\
cellspacing="0"\
class="tocTable">\
  <tr valign="top"> \
    <td class="tocTableHeaderCell" colspan="2"> \
      <span class="TableHead">Chapter Contents</span></td>\
  </tr>\
-->
<!-- Table Version 3 -->\
<!--
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#Intro">1.</a></td>\
  <td class="tocTableContentCell3">\
    <a href="#Intro">Introduction</a></td> \
  </tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#DGroupObj">2.</a></td>\

```

```

    <td class="tocTableContentCell3">\
<a href="#DGroupObj">Description of the Group Object</a></td>\
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#H5Dump">3.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#H5Dump">Using <code>h5dump</code></a></td> \
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#GroupFuncSums">4.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#GroupFuncSums">Group (H5G) Function Summaries</a></td>\
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#ProgModel">5.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#ProgModel">Programming Model</a></td> \
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#DiscoverInfo">6.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#DiscoverInfo">Discovering Information About Objects</a></td>\
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#DiscoverGrObjs">7.</a></td>\
    <td class="tocTableContentCell3">\

```

```

<a href="#DiscoverGrObjs">Discovering Objects in a Group</a></td>\
</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#DiscoverAll">8.</a></td>\
  <td class="tocTableContentCell3">\
    <a href="#DiscoverAll">Discovering All the Objects in the File</a></td>\
  </tr>\
\
<tr valign="top"> \
  <td class="tocTableContentCell"> \
-->
<!-- editingComment -- "tocTableContentCell" and "tocTableContentCell4" \
-->\
<!-- are the table-closing cell class.\
  <td class="tocTableContentCell2"> \
-->\
<!--
  <a href="#Examples">9.</a></td>\
  <td class="tocTableContentCell4">\
    <a href="#Examples">Examples of File Structures</a>\
-->
<!-- editingComment -- This section not currently complete or validated.\
</tr><tr valign="top"> \
  <td class="tocTableContentCell"> \
    <a href="#Appendix">10</a></td>\
  <td class="tocTableContentCell4"><a href="#Appendix">Appendix</a></td>\
-->\
<!--
</td></tr>\
</table>\

```

```

')
-->
</SCRIPT>
<!--(End TOC)=====-->

<!--(TOC VERS 1 and 2)=====-->
<!-- Table Version 1
<tr valign="top">
  <td class="tocTableContentCell" colspan="2">
    <a href="#Intro">1: Introduction</a>
  <br />
    <a href="#DGroupObj">2: Description of the Group</a>
  <br />
    <a href="#h5dump">3: Using <code>h5dump</code></a>
  <br />
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="#DGroupObj">Object</a>
  <br />
    <a href="#GroupFuncSums">4: Group Function</a>
  <br />
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="#GroupFuncSums">Summaries</a>
  <br />
    <a href="#ProgModel">5: The Programming Model</a>
  <br />
    <a href="#DiscoverInfo">6: Discovering Information</a>
  <br />
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="#DiscoverInfo">About Objects</a>
  <br />
    <a href="#DiscoverAll">7: Discovering All the</a>
  <br />
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="#DiscoverAll">Objects in the File</a>
  <br />

```

```

<a href="#Examples">8: Examples of File</a>
<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="#Examples">Structures</a>
<br />
<a href="#Appendix">9: Appendix</a>
</td>
</tr>
-->

<!-- Table Version 2
<tr valign="top">
  <td class="tocTableContentCell2">
    <a href="#Intro">1</a></td>
    <td class="tocTableContentCell3"><a href="#Intro">Introduction</a>
  </tr><tr valign="top">
    <td class="tocTableContentCell2">
      <a href="#DGroupObj">2</a></td>
      <td class="tocTableContentCell3"><a href="#DGroupObj">Description of the Group Object</a>
    </tr><tr valign="top">
      <td class="tocTableContentCell2">
        <a href="#h5dump">3</a></td>
        <td class="tocTableContentCell3"><a href="#h5dump">Using <code>h5dump</code></a>
      </tr><tr valign="top">
        <td class="tocTableContentCell2">
          <a href="#GroupFuncSums">4</a></td>
          <td class="tocTableContentCell3"><a href="#GroupFuncSums">Group Function Summaries</a>
        </tr><tr valign="top">
          <td class="tocTableContentCell2">
            <a href="#ProgModel">5</a></td>
            <td class="tocTableContentCell3"><a href="#ProgModel">Programming Model</a>
          </tr><tr valign="top">

```

```

<td class="tocTableContentCell2">
<a href="#DiscoverInfo">6</a></td>
<td class="tocTableContentCell3"><a href="#DiscoverInfo">Discovering Information About Objects</a>
</tr><tr valign="top">
<td class="tocTableContentCell2">
<a href="#DiscoverAll">7</a></td>
<td class="tocTableContentCell3"><a href="#DiscoverGrObjs">Discovering Objects in a Group</a>
</tr><tr valign="top">
<td class="tocTableContentCell2">
<a href="#DiscoverAll">8</a></td>
<td class="tocTableContentCell3"><a href="#DiscoverAll">Discovering All the Objects in the File</a>
</tr><tr valign="top">
<td class="tocTableContentCell2">
<a href="#Examples">9</a></td>
<td class="tocTableContentCell3"><a href="#Examples">Examples of File Structures</a>
</tr><tr valign="top">
<td class="tocTableContentCell">
<a href="#Appendix">10</a></td>
<td class="tocTableContentCell4"><a href="#Appendix">Appendix</a></td>
</tr>
-->

<!--(END TOC VERS 1 and 2)=----->

<div align="center">
<a name="TOP">

```

5. HDF5 Groups

</div>

<h3>4.1. Introduction</h3>

<!-- editingComment

[[[

The use of the term "path name" becomes rather stilted in this chapter.

Add a note early on that the shorthand "path" is generally used as an equivalent. Then edit "path name" ==> "path" where appropriate.

]]]

-->

<p>

As suggested by the name Hierarchical Data Format,
an HDF5 file is hierarchically structured.

The HDF5 group and link objects implement this hierarchy. </p>

<p>

In the simple and most common case,
the file structure is a tree structure;

in the general case, the file structure may be a
directed graph with a designated entry point.

The tree structure is very similar to the file system

structures employed on UNIX systems, directories and files, and on Apple Macintosh and Microsoft Windows systems, folders and files.

HDF5 groups are analogous to the directories and folders;

HDF5 datasets are analogous to the files.

The one very important difference

between the HDF5 file structure and

the above-mentioned file system analogs

is that HDF5 groups are linked as a directed graph,

allowing circular references;

the file systems are strictly hierarchical,

allowing no circular references.

The figures below illustrate the range of possibilities.

In Figure 1, the group structure is strictly hierarchical,

identical to the file system analogs.

In Figures 2 and 3, the structure takes advantage of the directed graph's allowance of circular references.

In Figure 2, `GroupA` is not only

a member of the root group, ,

but a member of `GroupC`.

Since Group C is a member of Group B

and Group B is a member of Group A,

Dataset1 can be accessed by means of the circular reference

`/Group A/Group B/Group C/Group A/Dataset1`.

Figure 3 illustrates an extreme case in which
`GroupB` is a member of itself, enabling a
reference to a member dataset such as
`/Group A/Group B/Group B/Group B/Dataset2`.

```
<table align="center" width="400" border="0">
  <tr valign="center" align="center">
    <td>
      <hr color="green" size="3"/>
      </td>
    </tr>
    <tr valign="top" align="left">
      <td>
        <hr color="green" size="1" />
        <b>Figure 1. An HDF5 file with a strictly hierarchical group structure</b>
        <hr color="green" size="3"/></td>
      </tr>
</table>
<br />
<br />
```

```
<table align="center" width="400" border="0">
  <tr valign="center" align="center">
    <td><hr color="green" size="3"/>
    </td>
  </tr>
  <tr valign="top" align="left">
    <td >
      <hr color="green" size="1" />
```


Figure 2. An HDF5 file with a directed graph group structure including a circular reference

|
 --- | ||
 --- |

Figure 3. An HDF5 file with a directed graph group structure and one group as a member of itself

As becomes apparent upon reflection,
 directed graph structures can become quite complex;
 caution is advised!

<p>

The balance of this chapter discusses the following topics:</p>

- The HDF5 group object (or a group)
and its structure in more detail
- HDF5 link objects (or links)
- The programming model for working with groups and links
- HDF5 functions provided for working with groups, group members, and
links
- Retrieving information about objects in a group
- Discovery of the structure of an HDF5 file and the contained objects
- Examples of file structures

<SCRIPT language="JavaScript">

<!--

document.writeln ("

<div align=right>

(Top)

</div>

");

-->

</SCRIPT>

<h3 class=pagebefore>4.2. Description of the Group Object</h3>

<h4>4.2.1 The Group Object</h4>

<p>

Abstractly, an HDF5 group contains zero or more objects
and every object must be a member of at least one group.

The root group, the sole exception, may not belong to any group.</p>

```
<table width = 400 cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td></tr>
    <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 4. Abstract model of the HDF5 group object</b>
      <hr color="green" size="3"/>
    </td></tr>
</table>
<br />
```

<p>

Group membership is actually implemented via
link objects. See the figure above.

A link object is owned by a group and points to a

named object.

Each link has a name, and each link points to exactly one object.

Each named object has at least one and possibly many links to it.</p>

<p>There are three classes of named objects: group, dataset, and named datatype. See the figure below.

Each of these objects is the member of at least one group, which means there is at least one link to it.</p>

```
<table width="500" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center"><hr color="green" size="3"/>
    
  </td></tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr>
    <td align="left" ><b>Figure 5. Classes of named objects</b>
      <hr color="green" size="3"/>
    </td></tr>
</table>
<br />
```

<p>

The primary operations on a group are to add and remove members and to discover member objects.

These abstract operations, as listed in the figure below, are implemented in the H5G APIs, as listed in section 4,

“Group Function Summaries.”</p>

<p>

To add and delete members of a group,
links from the group to existing objects in the file
are created and deleted with the
<code>link</code> and <code>unlink</code> operations.

When a new named object is created,
the HDF5 Library executes the link operation in the background
immediately after creating the object
(i.e., a new object is added as a member of the group in which it
is created without further user intervention).</p>

<p>

Given the name of an object, the get_object_info
method retrieves a description of the object,
including the number of references to it.

The iterate method iterates through the members of
the group, returning the name and type of each object.</p>

<!--

<table x-use-null-cells

width="600"

cellspacing="0"

class="fullImgTable"

align="center">

<tr valign="top">

<td class="fullImgTableImgCell" align="center">

-->

```

<!-- NEW PAGE -->
<table border="0" width="300" align="center">
  <tr valign="top"><td>&nbsp;&nbsp;&nbsp;</td>
    <td align="center"><hr color="green" size="3"/>
  <br />
  <table align="center" border="1">
    <tr><td align="center"><code><b>Group</b></code></td></tr>
    <tr><td align="left"><code>size:size_t</code></td></tr>
    <tr><td align="left"><code>create()<br />
      open()<br />
      close()<br />
      <br />
      link()<br />
      unlink()<br />
      move()<br />
      <br />
      iterate()<br />
      get_object_info()<br />
      get_link_info()</code></td></tr>
  </table>
  <br />
</td><td>&nbsp;&nbsp;&nbsp;</td></tr>
<tr><td>&nbsp;&nbsp;&nbsp;</td>
  <td align="left"><hr color="green" size="1"/>
    <b>Figure 6. The group object</b><br />
    <hr color="green" size="3"/>
  </td><td>&nbsp;&nbsp;&nbsp;</td></tr>
</table>
<br />

<p>

```

Every HDF5 file has a single root group, with the name `/`. The root group is identical to any other HDF5 group, except:

- The root group is automatically created when the HDF5 file is created (`H5Fcreate`).
- The root group has no parent, but, by convention has a reference count of 1.
- The root group cannot be deleted (i.e., unlinked)!

4.2.2 The Hierarchy of Data Objects

An HDF5 file is organized as a rooted, directed graph using HDF5 group objects. The named data objects are the nodes of the graph, and the links are the directed arcs.

Each arc of the graph has a name, with the special name `/` reserved for the root group.

New objects are created and then inserted into the graph with a link operation that is automatically executed by the library; existing objects are inserted into the graph with a link operation explicitly called by the user, which creates a named link from a group to the object.

An object can be the target of more than one link.

The names on the links must be unique within each group, but there may be

many links with the same name in different groups.

These are unambiguous, because some ancestor must have a different name, or else they are the same object.

The graph is navigated with path names, analogous to Unix file systems (see section 2.3,

[HDF5 Path Names](#PathNames)).

An object can be opened with a full path starting at the root group, or with a relative path and a starting point.

That starting point is always a group, though it may be the current working group, another specified group, or the root group of the file.

Note that all paths are relative to a single HDF5 file.

In this sense, an HDF5 file is analogous to a single UNIX file system.

[¹](#FootNote)

It is important to note that, just like the UNIX file system,

HDF5 objects do not have *names*, the names are associated with *paths*.

An object has an *object identifier* that is unique within the file, but a single object may have many *names* because there may be many paths to the same object.

An object can be renamed, or moved to another group, by adding and deleting links. In this case, the object itself never moves.

For that matter, membership in a group has no implication for the physical location of the stored object.

Deleting a link to an object does not necessarily delete the object.

The object remains available as long as there is at least one link to it.

After all links to an object are deleted, it can no longer be opened, and the storage may be reclaimed.

It is also important to realize that the linking mechanism can be used to construct very complex graphs of objects.

For example, it is possible for object to be shared between several groups and even to have more than one name in the same group.

It is also possible for a group to be a member of itself, or to create other *cycles* in the graph, such as in the case where a child group is linked to one of its ancestors.

HDF5 also has *soft links* similar to UNIX soft links.

A *soft link* is an object that has a name and a path name for the target object. The soft link can be followed to open the target of the link just like a regular or *hard* link.

The differences are that the hard link cannot be created if the target object does not exist and it always points to the same object.

A soft link can be created with any path name, whether or not the object exists; it may or may not, therefore, be possible to follow a soft link.

Furthermore, a soft link's target object may be changed.

[PathNames](#)

4.2.3 HDF5 Path Names

editingComment

editingComment

```
[[[
```

The opening paragraph here is still foggy, and now a bit redundant.

A discussion of objects before launching into the discussion of links might be helpful?

```
]]]
```

```
</span>
```

```
-->
```

```
<p>
```

The structure of the HDF5 file constitutes the name space for the objects in the file.

A path name is a string of components separated by slashes (`<code>/</code>`).

Each component is the name of a hard or soft link which points to an object in the file.

The slash not only separates the components, but indicates their hierarchical relationship; the component indicated by the link name following a slash is always a member of the component indicated by the link name preceding that slash.
</p>

```
<p>
```

The first component in the path name may be any of the following:
</p>

```
<ul><li>the special character dot (<code>.</code>, a period),
```

```
    indicating the current group </li>
```

```
    <li>the special character slash (<code>/</code>),
```

```
    indicating the root group </li>
```

```
    <li>any member of the current group </li>
```

```
</ul>
```

```
<p>
```

Component link names may be any string of ASCII characters

not containing a slash or a dot

(`</code>` and `</code>.`, which are reserved as noted above).

However, users are advised to avoid the use of punctuation and

non-printing characters, as they may create problems for other software.

The figure below provides a BNF grammar for HDF5 path names.

<pre> PathName ::= AbsolutePathName RelativePathName Separator ::= "/" ["/"] * AbsolutePathName ::= Separator [RelativePathName] RelativePathName ::= Component [Separator RelativePathName] * Component ::= "." Characters Characters ::= Character+ - { "." } Character ::= { c: c ^ { { legal ASCII characters } - { '/' } } } </pre>	<hr/> <p>Figure 7. A BNF grammar for HDF5 path names</p> <hr/>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------

<!-- NEW PAGE -->

An object can always be addressed by either a

full or absolute path name, starting at the root group,

or by a `relative` path name, starting in a known location such as the current working group.

As noted elsewhere, a given object may have multiple full and relative path names.

`<p>`

Consider, for example, the file illustrated in the figure below.

`<code>Dataset1</code>` can be identified by either of these absolute path names:`</p>`

`<pre>`

`/GroupA/Dataset1`

`/GroupA/GroupB/GroupC/Dataset1</pre>`

`<p>`

Since an HDF5 file is a directed graph structure, and is therefore not limited to a strict tree structure, and since this illustrated file includes the sort of circular reference that a directed graph enables,

`<code>Dataset1</code>` can also be identified by this absolute path name:`</p>`

`<pre>`

`/GroupA/GroupB/GroupC/GroupA/Dataset1</pre>`

`<p>`

Alternatively, if the current working location is `<code>GroupB</code>`,

`<code>Dataset1</code>` can be identified by either of these relative path names:`</p>`

`<pre>`

`GroupC/Dataset1`

`GroupC/GroupA/Dataset1</pre>`

`<p>`

Note that relative path names in HDF5 do not employ the `<code>../</code>` notation, the UNIX notation indicating a parent directory, to indicate a parent group.</p>

```
<table width="400" align="center" border="0">
```

```
<tr valign="bottom" align="center">
```

```
<td><hr color="green" size="3"/>
```

```

```

```
</td></tr>
```

```
<tr valign="top" align="left">
```

```
<td><hr color="green" size="1"/>
```

```
<b>Figure 8.
```

```
An HDF5 file with a directed graph group structure
```

```
including a circular reference</b>
```

```
<hr color="green" size="3"/>
```

```
</td></tr>
```

```
</table>
```

```
<br />
```

```
<!-- *** BEGIN _topic/group_implementations.htm NEAR DUPLICATE *** -->
```

```
<!-- *** SEE editingComment FOLLOWING SECTION HEADING *** -->
```

```
<a name="GroupStyles">
```

```
<h4>4.2.4 Group Implementations in HDF5</h4></a>
```

```
<!-- editingComment
```

```
<span class="editingComment">
```

```
[[[
```

```
This section is a close copy of but not identical to
```

_topic/group_implementations.htm, which is sourced into the RM.

If/when the UG is broken into smaller files, re-unifying these sections should be considered.

]]]

-->

<p>

The original HDF5 group implementation provided a single indexed structure for link storage.

A new group implementation, in HDF5 Release 1.8.0, enables more efficient compact storage for very small groups, improved link indexing for large groups, and other advanced features.</p>

The <i>original indexed</i> format remains the default.

Links are stored in a B-tree in the group's local heap.

Groups created in the new <i>compact-or-indexed</i> format, the implementation introduced with Release 1.8.0, can be tuned for performance, switching between the compact and indexed formats at thresholds set in the user application.

The <i>compact</i> format will conserve file space and processing overhead when working with small groups and is particularly valuable when a group contains no links.

Links are stored as a list of messages in the group's header.

The <i>indexed</i> format will yield improved performance when working with large groups, e.g., groups containing thousands to millions of members.

- Links are stored in a fractal heap and indexed with an improved B-tree.

- The new implementation also enables the use of link names consisting of non-ASCII character sets (see ../RM/RM_H5P.html#Property-SetCharEncoding)
`H5Pset_char_encoding` and is required for all link types other than hard or soft links, e.g., external and user-defined links (see the ../RM/RM_H5L.html) H5L APIs).

The original group structure and the newer structures are not directly interoperable.

By default, a group will be created in the original indexed format. An existing group can be changed to a compact-or-indexed format if the need arises; there is no capability to change back.

As stated above, once in the compact-or-indexed format, a group can switch between compact and indexed as needed.

Groups will be initially created in the compact-or-indexed format only when one or more of the following conditions is met:

- The *low version bound* value of the *library version bounds* property has been set to Release 1.8.0 or later in the file access property list (see ../RM/RM_H5P.html#Property-SetLibverBounds)
`H5Pset_libver_bounds`).

Currently, that would require an `H5Pset_libver_bounds` call with the `low` parameter set to `H5F_LIBVER_LATEST`.

When this property is set for an HDF5 file, all objects in the file will be created using the latest available format; no effort will be made to create a file that can be read by older libraries.

The creation order tracking property,

`H5P_CRT_ORDER_TRACKED`, has been set in the group creation property list

(see http://www.hdfgroup.org/hdf/5.0/RM/RM_H5P.html#Property-SetLinkCreationOrder)

`H5Pset_link_creation_order`.

The compact-or-indexed format also enables completely new capabilities: user-defined and external links and non-ASCII link names.

An existing group, currently in the original indexed format, will be converted to the compact-or-indexed format upon the occurrence of any of the following events:

An external or user-defined link is inserted into the group.

A link named with a string composed of non-ASCII characters is inserted into the group.

QUESTION -- REVEAL THIS CIRCUMSTANCE ONLY AFTER (AND IF!)

-- IT IS IMPLEMENTED, THEN DELETE THE COMMENT

The `max_compact` and `min_dense`

properties have been set in the group creation property list
 (see [\(see \[\]\(../RM/RM_H5P.html#Property-SetLinkPhaseChange\)](../RM/RM_H5P.html#Property-SetLinkPhaseChange)
).
`H5Pset_link_phase_change`

[[[

Masked bullet, immediately above:

Not currently implemented.

-- June 2007, FMB

Confirmed with NF that H5Pset_link_phase_change
 still does not trigger a group style change.

-- August 2010, FMB

]]]

-->

<p>

The compact-or-indexed format offers performance improvements
 that will be most notable at the extremes,
 i.e., in groups with zero members
 and in groups with tens of thousands of members.
 But measurable differences may sometimes appear
 at a threshold as low as eight group members.
 Since these performance thresholds and criteria differ from
 application to application, tunable settings are provided to
 govern the switch between the compact and indexed formats
 (see [\(see \[\]\(../RM/RM_H5P.html#Property-SetLinkPhaseChange\)](../RM/RM_H5P.html#Property-SetLinkPhaseChange)
).
`H5Pset_link_phase_change`
 Optimal thresholds will depend on the application and the
 operating environment.</p>

<p>

Future versions of HDF5 will retain the ability to
create, read, write, and manipulate
all groups stored in either the original indexed format or
the compact-or-indexed format.</p>

<!-- *** END _topic/group_implementations.htm NEAR DUPLICATE *** -->

<SCRIPT language="JavaScript">

<!--

document.writeln ("

<div align=right>

(Top)

</div>

");

-->

</SCRIPT>

<!-- NEW PAGE -->

<h3 class=pagebefore>4.3. Using <code>h5dump</code></h3>

<p>You can use <code>h5dump</code>, the command-line utility
distributed with HDF5, to examine a file for purposes either of
determining where to create an object within an HDF5 file or
to verify that you have created an object in the intended place.
inspecting the contents of an HDF5 file.</p>

<p>

In the case of the new group created in section 5.1,
“Creating a group,”
the following <code>h5dump</code> command will display the
contents of <code>FileA.h5</code>:</p>

<dir><pre>
h5dump FileA.h5
</pre></dir>

<p>

Assuming that the discussed objects, <code>GroupA</code> and
<code>GroupB</code> are the only objects that exist in
<code>FileA.h5</code>, the output will look something like the
following:</p>

<dir><pre>
HDF5 "FileA.h5" {
GROUP "/" {
GROUP GroupA {
GROUP GroupB {
}
}
}
}
</pre></dir>

<p>

<code>h5dump</code> is fully described on the
Tools page of the

[<cite>HDF5 Reference Manual</cite>.</p>](../RM/RM_H5Front.html)

The HDF5 DDL grammar is fully described in the document
[DDL in BNF for HDF5,
an element of this <cite>HDF5 User's Guide</cite>.</p>](../ddl.html)

```
<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="GroupFuncSums">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>
```

```
<!-- NEW PAGE -->
<a name="GroupFuncSums">
<h3 class=pagebefore>4.4. Group Function Summaries</h3>
</a>
<p>Functions that can be used with groups (H5G functions) and property
list functions that can used with groups (H5P functions) are listed below.
A number of group functions have been deprecated. Most of these have become
link (H5L) or object (H5O) functions. These replacement functions are also
listed below.</p>
```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="3" align="left" valign="bottom">
      <b>Function Listing 1. Group functions (H5G)</b></td>
    </tr>
  <tr height="5"><td colspan="3"><hr color="green" size="3" /></td></tr>
  <tr valign="top">
    <td>
      <b>C Function<br />Fortran Function</b>
    </td><td>&nbsp;</td>
    <td>
      <b>Purpose</b>
    </td>
  </tr>
  <tr><td colspan="3"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td colspan=1 rowspan=1>
      <code>H5Gcreate<br />h5gcreate_f</code>
    </td><td>&nbsp;</td>
    <td colspan=1 rowspan=1>
      Creates a new empty group and gives it a name. The C function is a
      macro: see <a href=" ../RM/APICompatMacros.html">&ldquo;API
      Compatibility Macros in HDF5.&rdquo;</a>
    </td>
  </tr>
  <tr><td colspan="3"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td colspan=1 rowspan=1>
      <code>H5Gcreate_anon<br />h5gcreate_anon_f</code>
    </td><td>&nbsp;</td>
    <td colspan=1 rowspan=1>

```

Creates a new empty group without linking it into the file structure.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Gopen
h5gopen_f</code>

</td><td> </td>

<td>

Opens an existing group for modification and returns a group

identifier for that group. The C function is a

macro: see “API

Compatibility Macros in HDF5.”

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Gclose
h5gclose_f</code>

</td><td> </td>

<td>

Closes the specified group.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Gget_create_plist
h5gget_create_plist_f</code>

</td><td> </td>

<td>

Gets a group creation property list identifier.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Gget_info<br />h5gget_info_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Retrieves information about a group.

Use instead of `H5Gget_num_objs`.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Gget_info_by_idx<br />h5gget_info_by_idx_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Retrieves information about a group according to the group's position within an index.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Gget_info_by_name<br />h5gget_info_by_name_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Retrieves information about a group.

```
</td>
```

```
</tr>
```

```

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>(none)<br />h5gget_obj_info_idx_f</code>
  </td><td>&nbsp;</td>
  <td>
    Returns name and type of the group member identified by its index.
    Use with the <code>h5gn_members_f</code> function.
    <code>h5gget_obj_info_idx_f</code> and <code>h5gn_members_f</code>
    are the Fortran equivalent of
    the C function <code>H5Literate</code>.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>(none)<br />h5gn_members_f</code>
  </td><td>&nbsp;</td>
  <td>
    Returns the number of group members.
    Use with the <code>h5gget_obj_info_idx_f</code> function.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<!-- NEW PAGE -->

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">

```



```

    <td colspan="3" align="left" valign="bottom">
    <b>Function Listing 2. Link (H5L) and object (H5O) functions
    </b></td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
<tr valign="top">
    <td>
        <b>C Function<br />Fortran Function</b>
    </td><td>&nbsp;</td>
    <td>
        <b>Purpose</b>
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
<code>H5Lcreate_hard<br />h5lcreate_hard_f</code>
    </td><td>&nbsp;</td>
    <td>
Creates a hard link to an object.
        Replaces <code>H5Glink</code> and <code>H5Glink2</code>.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
<code>H5Lcreate_soft<br />h5lcreate_soft_f</code>
    </td><td>&nbsp;</td>
    <td>
Creates a soft link to an object.
        Replaces <code>H5Glink</code> and <code>H5Glink2</code>.
    </td>

```

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Lcreate_external<br />h5lcreate_external_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Creates a soft link to an object in a different file.

Replaces `H5Glink` and `H5Glink2`.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Lcreate_ud<br />(none)</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Creates a link of a user-defined type.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Lget_val<br />(none)</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Returns the value of a symbolic link.

Replaces `H5Gget_linkval`.

```
</td>
```

```
</tr>
```

```

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
<code>H5Literate<br />h5literate_f</code>
  </td><td>&nbsp;</td>
  <td>
Iterates through links in a group.
  Replaces <code>H5Giterate</code>.
  See also <code>H5Ovisit</code> and <code>H5Lvisit</code>.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
<code>H5Literate_by_name<br />h5literate_by_name_f</code>
  </td><td>&nbsp;</td>
  <td>
Iterates through links in a group.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
<code>H5Lvisit<br />(none)</code>
  </td><td>&nbsp;</td>
  <td>
Recursively visits all links starting from a specified group.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">

```

<td>

<code>H5Ovisit
h5ovisit_f</code>

</td><td> </td>

<td>

Recursively visits all objects accessible from a specified object.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Lget_info
h5lget_info_f</code>

</td><td> </td>

<td>

Returns information about a link.

Replaces <code>H5Gget_objinfo</code>.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Oget_info
(none)</code>

</td><td> </td>

<td>

Retrieves the metadata for an object specified by an identifier.

Replaces <code>H5Gget_objinfo</code>.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Lget_name_by_idx
h5lget_name_by_idx_f</code>

```
</td><td>&nbsp;</td>
```

```
<td>
```

Retrieves name of the nth link in a group, according to the order within a specified field or index.

Replaces `H5Gget_objname_by_idx`.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Oget_info_by_idx<br />(none)</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Retrieves the metadata for an object, identifying the object by an index position.

Replaces `H5Gget_objtype_by_idx`.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Oget_info_by_name<br />h5oget_info_by_name_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Retrieves the metadata for an object, identifying the object by location and relative name.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Oset_comment<br />(none)</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Sets the comment for specified object.

Replaces `H5Gset_comment`.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Oget_comment<br />(none)</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Gets the comment for specified object.

Replaces `H5Gget_comment`.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Ldelete<br />h5ldelete_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Removes a link from a group.

Replaces `H5Gunlink`.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Lmove<br />h5lmove_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Renames a link within an HDF5 file.

Replaces `H5Gmove` and `H5Gmove2`.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
```

```
</table>
```

```
<br />
```

```
<br />
```

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<td colspan="3" align="left" valign="bottom">
```

```
<b>Function Listing 3. Group creation property list functions (H5P)
```

```
</b></td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<span class="TableHead">C Function<br />Fortran Function</span>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

```
<span class="TableHead">Purpose</span>
```

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pall_filters_avail<br />(none)</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Verifies that all required filters are available.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pget_filter<br />h5pget_filter_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Returns information about a filter in a pipeline. The C function

is a macro: see ["API](../RM/APICompatMacros.html)

Compatibility Macros in HDF5."["API](#)

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pget_filter_by_id<br />h5pget_filter_by_id_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Returns information about the specified filter. The C function

is a macro: see ["API](../RM/APICompatMacros.html)

Compatibility Macros in HDF5."["API](#)

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```



```
<code>H5Pget_nfilters<br />h5pget_nfilters_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Returns the number of filters in the pipeline.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pmodify_filter<br />h5pmodify_filter_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Modifies a filter in the filter pipeline.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Premove_filter<br />h5premove_filter_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Deletes one or more filters in the filter pipeline.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pset_deflate<br />h5pset_deflate_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Sets the deflate (GNU gzip) compression method and compression

level.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_filter
h5pset_filter_f</code>

</td><td> </td>

<td>

Adds a filter to the filter pipeline.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_fletcher32
h5pset_fletcher32_f</code>

</td><td> </td>

<td>

Sets up use of the Fletcher32 checksum filter.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_fletcher32
h5pset_fletcher32_f</code>

</td><td> </td>

<td>

Sets up use of the Fletcher32 checksum filter.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

```
<tr valign="top">
  <td>
    <code>H5Pset_link_phase_change<br />h5pset_link_phase_change_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets the parameters for conversion between compact and dense groups.
  </td>
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
  <td>
    <code>H5Pget_link_phase_change<br />h5pget_link_phase_change_f</code>
  </td><td>&nbsp;</td>
  <td>
    Queries the settings for conversion between compact and dense groups.
  </td>
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
  <td>
    <code>H5Pset_est_link_info<br />h5pset_est_link_info_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets estimated number of links and length of link names in a group.
  </td>
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
  <td>
    <code>H5Pget_est_link_info<br />h5pget_est_link_info_f</code>
  </td><td>&nbsp;</td>
```

<p><td></p> <p>Queries data required to estimate required local heap or object header size.</p> <p></td></p>	
</tr>	
<!-- NEW PAGE -->	
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>	
<tr valign="top">	
<p><td></p> <p><code>H5Pset_nlinks
h5pset_nlinks_f</code></code></p> <p></td><td>&nbsp;</td></p>	
<td>	
Sets maximum number of soft or user-defined link traversals.	
</td>	
</tr>	
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>	
<tr valign="top">	
<p><td></p> <p><code>H5Pget_nlinks
h5pget_nlinks_f</code></code></p> <p></td><td>&nbsp;</td></p>	
<td>	
Retrieves the maximum number of link traversals.	
</td>	
</tr>	
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>	
<tr valign="top">	
<p><td></p> <p><code>H5Pset_link_creation_order
h5pset_link_creation_order_f</code></code></p> <p></td><td>&nbsp;</td></p>	
<td>	

Sets creation order tracking and indexing for links in a group.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pget_link_creation_order

h5pget_link_creation_order_f</code>

</td><td> </td>

<td>

Queries whether link creation order is tracked and/or indexed
in a group.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_create_intermediate_group

h5pset_create_inter_group_f</code>

</td><td> </td>

<td>

Specifies in the property list whether to create missing intermediate
groups.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pget_create_intermediate_group
(none)</code>

</td><td> </td>

<td>

Determines whether the property is set to enable creating missing intermediate groups.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_char_encoding
h5pset_char_encoding_f</code>

</td><td> </td>

<td>

Sets the character encoding used to encode a string.

Use to set ASCII or UTF-8 character encoding for object names.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pget_char_encoding
h5pget_char_encoding_f</code>

</td><td> </td>

<td>

Retrieves the character encoding used to create a string.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

</table>

<table width="600" cellspacing="0" align="center" cellpadding="0">

```

<tr valign="bottom">
  <td colspan="3" align="left" valign="bottom">
    <b>Function Listing 4. Other external link functions
  </b></td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
<tr valign="top">
  <td>
    <b>C Function<br />Fortran Function</b>
  </td><td>&nbsp;</td>
  <td>
    <b>Purpose</b>
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset/get_elink_file_cache_size</code>
    <br />
    <code>(none)</code></td><td>&nbsp;</td>
  <td>
    Sets/retrieves the size of the external link open file cache
    from the specified file access property list. </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Fclear_elink_file_cache</code>
    <br />
    <code>(none)</code></td><td>&nbsp;</td>
  <td>

```

Clears the external link open file cache for a file.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

</table>

<SCRIPT language="JavaScript">

<!--

document.writeln ("

<div align=right>

(Top)

</div>

");

-->

</SCRIPT>

<!-- NEW PAGE -->

<h3 class=pagebefore>4.5. Programming Model: Working with Groups</h3>

<p>The programming model for working with groups is as follows:</p>

Create a new group or open an existing one.


```

<li>Perform the desired operations on the group. </li>
<ul>
  <li>Create new objects in the group.</li>
  <li>Insert existing objects as group members.</li>
<li>Delete existing members.</li>
<li>Open and close member objects.</li>
<li>Access information regarding member objects.</li>
<li>Iterate across group members.</li>
<li>Manipulate links.</li>
</ul>
<li>Terminate access to the group. (Close the group.)</li>
</ol>

```

```

<a name="ModelCreateGroup">

```

```

<h4>4.5.1 Creating a Group</h4></a>

```

```

<p>

```

To create a group, use `H5Gcreate`, specifying the location and the path of the new group.

The location is the identifier of the file or the group in a file with respect to which the new group is to be identified.

The path is a string that provides wither an absolute path or a relative path to the new group (see section 2.3,

“HDF5 Path Names”).

A path that begins with a slash (`/`) is an absolute path indicating that it locates the new group from the root group of the HDF5 file.

A path that begins with any other character is a relative path.

When the location is a file, a relative path is a path from that file's root group;

when the location is a group, a relative path is a path from that group.</p>

<p>

The sample code in the example below creates three groups.

The group `Data` is created in the root directory;

two groups are then created in `/Data`,

one with absolute path, the other with a relative path.</p>

```
<table width="600" cellpadding="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="left">
```

```
      <hr color="green" size="3"/>
```

```
<pre>
```

```
hid_t file;
```

```
file = H5Fopen(....);
```

```
group = H5Gcreate(file, "/Data", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
```

```
group_new1 = H5Gcreate(file, "/Data/Data_new1", H5P_DEFAULT, H5P_DEFAULT,
    H5P_DEFAULT);
```

```
group_new2 = H5Gcreate(group, "Data_new2", H5P_DEFAULT, H5P_DEFAULT,
    H5P_DEFAULT);
```

```
</pre>
```

```
  </td></tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
  <td align="left" >
```

```
    <b>Example 1. Creating three new groups</b>
```

```
    <hr color="green" size="3"/></td>
```

```
  </tr>
```

```
</table>
```

```
<br />
```

<p>

The third `H5Gcreate` parameter optionally specifies how much file space to reserve to store the names that will appear in this group. If a non-positive value is supplied, a default size is chosen. </p>

<!-- editingComment

(See XXX for an explanation of performance tuning.)

-->

<!-- NEW PAGE -->

<h4>4.5.2 Opening a Group and Accessing an Object in that Group</h4>

<p>Though it is not always necessary, it is often useful to explicitly open a group when working with objects in that group. Using the file created in the example above, the example below illustrates the use of a previously-acquired file identifier and a path relative to that file to open the group `Data`.</p>

<p>

Any object in a group can be also accessed by its absolute or relative path.

To open an object using a relative path, an application must first open the group or file on which that relative path is based.

To open an object using an absolute path, the application can use any location identifier in the same file as the target object;

the file identifier is commonly used, but object identifier for any object in that file will work.

Both of these approaches are illustrated in the example below. </p>

<p>

Using the file created in the examples above,
the example below provides sample code illustrating the use of both
relative and absolute paths to access an HDF5 data object.

The first sequence (two function calls) uses a previously-acquired
file identifier to open the group `Data`, and
then uses the returned group identifier and a relative path to open
the dataset `CData`.

The second approach (one function call) uses the same previously-acquired
file identifier and an absolute path to open the same dataset. </p>

<pre> group = H5Gopen(file, "Data", H5P_DEFAULT); dataset1 = H5Dopen(group, "CData", H5P_DEFAULT); dataset2 = H5Dopen(file, "/Data/CData", H5P_DEFAULT); </pre>	<p>Example 2. Open a dataset with relative and absolute paths</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------

4.5.3 Creating a Dataset in a Specific Group

Any dataset must be created in a particular group.

As with groups, a dataset may be created in a particular group by specifying its absolute path or a relative path.

The example below illustrates both approaches to creating a dataset in the group `/Data/`.

<pre> dataspace = H5Screate_simple(RANK, dims, NULL); dataset1 = H5Dcreate(file, "/Data/CData", H5T_NATIVE_INT, dataspace, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT); group = H5Gopen(file, "Data", H5P_DEFAULT); dataset2 = H5Dcreate(group, "Cdata2", H5T_NATIVE_INT, dataspace, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT); </pre>	<p>Example 3. Create a dataset with absolute and relative paths</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------

```
</tr>
</table>
<br />
```

```
<!-- NEW PAGE -->
```

```
<h4>4.5.4 Closing a Group</h4>
```

To ensure the integrity of HDF5 objects and to release system resources, an application should always call the appropriate close function when it is through working with an HDF5 object. In the case of groups, `H5Gclose` ends access to the group and releases any resources the HDF5 Library has maintained in support of that access, including the group identifier.

As illustrated in the example below, all that is required for an `H5Gclose` call is the group identifier acquired when the group was opened; there are no relative versus absolute path considerations.

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
</pre>
herr_t status;
status = H5Gclose(group);
</pre>
```

```

    </td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left" >
        <b>Example 4. Close a group</b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

```
<p>
```

A non-negative return value indicates that the group was successfully closed and the resources released; a negative return value indicates that the attempt to close the group or release resources failed.</p>

```
<!-- editingComment
```

```
<span class="editingComment">
```

```
    [[
```

Return values have been decidedly under-discussed prior to this mention.

Probably ought to be mentioned with the discussion of each new function call.

```
    ]]
```

```
</span>
```

```
-->
```

<h4>4.5.5 Creating Links </h4>

<p>As previously mentioned, every object is created in a specific group.

Once created, an object can be made a member of additional groups by means of links created with one of the <code>H5Lcreate_*</code> functions.</p>

<p>

A <link> is, in effect, a path by which the target object can be accessed; it therefore has a name which functions as a single path component.

A link can be removed with an `H5Ldelete` call, effectively removing the target object from the group that contained the link (assuming, of course, that the removed link was the only link to the target object in the group). </p>

<p>

Hard Links

There are two kinds of links, <hard links> and <symbolic links>.

Hard links are reference counted; symbolic links are not.

When an object is created, a hard link is automatically created.

An object can be deleted from the file by removing all the hard links to it.</p>

<p>

Working with the file from the previous examples, the code in the example below illustrates the creation of a hard link, named `Data_link`, in the root group, `/`, to the group `Data`.

Once that link is created, the dataset `Cdata` can be accessed via either of two absolute paths, `/Data/Cdata` or `/Data_Link/Cdata`.</p>

<table width="600" cellspacing="0" align="center">

<tr valign="top">


```

<td align="left">
  <hr color="green" size="3"/>
<pre>
status = H5Lcreate_hard(Data_loc_id, "Data", DataLink_loc_id, "Data_link",
                        H5P_DEFAULT, H5P_DEFAULT)

dataset1 = H5Dopen(file, "/Data_link/CData", H5P_DEFAULT);
dataset2 = H5Dopen(file, "/Data/CData", H5P_DEFAULT);
</pre>
</td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 5. Create a hard link</b>
    <hr color="green" size="3"/></td></tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

```

<p>

```

The example below shows example code to delete a link, deleting the hard link `Data` from the root group. The group `/Data` and its members are still in the file, but they can no longer be accessed via a path using the component `/Data`.

```

<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="left">

```

```

    <hr color="green" size="3"/>
<pre>
status = H5Ldelete(Data_loc_id, "Data", H5P_DEFAULT);

dataset1 = H5Dopen(file, "/Data_link/CData", H5P_DEFAULT);
    /* This call should succeed; all path component still exist*/
dataset2 = H5Dopen(file, "/Data/CData", H5P_DEFAULT);
    /* This call will fail; the path component '/Data' has been deleted*/
</pre>
</td></tr>

<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 6. Delete a link </b>
        <hr color="green" size="3"/></td></tr>
</table>
<br />

```

<p>

When the last hard link to an object is deleted, the object is no longer accessible. `H5Ldelete` will not prevent you from deleting the last link to an object. To see if an object has only one link, use the `H5Oget_info` function. If the value of the rc (reference count) field in the is greater than 1, then the link can be deleted without making the object inaccessible.</p>

<p>The example below shows `H5Oget_info` to the group originally called `Data`. </p>

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
</td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 7. Finding the number of links to an object</b>
    <hr color="green" size="3"/></td></tr>
</table>
<br />

```

It is possible to delete the last hard link to an object and not make the object inaccessible. Suppose your application opens a dataset, and then deletes the last hard link to the dataset. While the dataset is open, your application still has a connection to the dataset. If your application creates a hard link to the dataset before it closes the dataset, then the dataset will still be accessible.

Symbolic Links

Symbolic links are objects that assign a name in a group to a path.

Notably, the target object is determined only when the symbolic link is accessed, and may, in fact, not exist. Symbolic links are not reference counted,

so there may be zero, one, or more symbolic links to an object. </p>

<p>

The major types of symbolic links are soft links and external links. Soft links are symbolic links within an HDF5 file and are created with the <code>H5Lcreate_soft</code> function. Symbolic links to objects located in external files, in other words external links, can be created with the <code>H5Lcreate_external</code> function. Symbolic links are removed with the <code>H5Ldelete</code> function.</p>

<p>

The example below shows the creating two soft links to the group <code>/Data</code>. </p>

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

```
status = H5Lcreate_soft(path_to_target, link_loc_id, "Soft2", H5P_DEFAULT, H5P_DEFAULT);
```

```
status = H5Lcreate_soft(path_to_target, link_loc_id, "Soft3", H5P_DEFAULT, H5P_DEFAULT);
```

```
dataset = H5Dopen(file, "/Soft2/CData", H5P_DEFAULT);
```

</pre>

</td></tr>

<tr><td><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td align="left">

Example 8. Create a soft link

```

        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

<p>

With the soft links defined in the example above, the dataset `CData` in the group `/Data` can now be opened with any of the names `/Data/CData`, `/Soft2/CData`, or `/Soft3/CData`.

In release 1.8.7, a cache was added to hold the names of files accessed via external links. The size of this cache can be changed to help improve performance. For more information, see the entry in the [HDF5 Reference Manual](http://www.hdfgroup.org/hdf/5.0/./RM/RM_H5Front.html) for the `H5Pset_elink_file_cache_size` function call.

<p>

Note Regarding Hard Links and Soft Links

Note that an object's existence in a file is governed by the presence of at least one hard link to that object.

If the last hard link to an object is removed, the object is removed from the file and any remaining soft link becomes a

dangling link, a link whose target object does not exist.

<p>

Moving or Renaming Objects, and a Warning</p>

<p>An object can be renamed by changing the name of a link to it with
<code>H5Lmove</code>.

This has the same effect as creating a new link with the new name and
deleting the link with the old name.</p>

<p>

Exercise caution in the use of <code>H5Lmove</code>

and <code>H5Ldelete</code> as these functions each include a step that
unlinks a pointer to an HDF5 object.

If the link that is removed is on the only path leading to an HDF5 object,
that object will become permanently inaccessible in the file. </p>

<p>Scenario 1: Removing the Last Link</p>

<p>To avoid removing the last link to an object or otherwise making an
object inaccessible, use the <code>H5Oget_info</code>
function. Make sure that the value of the reference count field (rc) is
greater than 1.

</p>

<p>Scenario 2: Moving a Link that Isolates an Object</p>

<p>Consider the following example: assume that the group <code>group2</code>
can only be accessed via the following path, where <code>top_group</code>
is a member of the file's root group: </p>

<code>/top_group/group1/group2</code>

<p>

Using `H5Lmove`, `top_group` is renamed to be a member of `group2`. At this point, since `top_group` was the only route from the root group to `group1`, there is no longer a path by which one can access `group1`, `group2`, or any member datasets. And since `top_group` is now a member of `group2`, `top_group` itself and any member datasets have thereby also become inaccessible.</p>

Mounting a File

<p>An external link is a permanent connection between two files. A temporary connection can be set up with the `H5Fmount` function. For more information, see the “[The HDF5 File](08_TheFile.html)” chapter, and the `H5Fmount` function in the [<cite>HDF5 Reference Manual</cite>](RM/RM_H5Front.html).</p>

```
<SCRIPT language="JavaScript">
```

```
<!--
```

```
document.writeln ("
```

```
<a name="DiscoverInfo">
```

```
<div align=right>
```

```
<a href="#TOP"><font size="-1">(Top)</font></a>
```

```
</div>
```

```

</a>
");
-->
</SCRIPT>

```

```

<a name="DiscoverInfo">
<h4 class=pagebefore>4.5.6 Discovering Information about Objects</h4>
</a>

```

```

<p>There is often a need to retrieve information about a particular object.
The <code>H5Lget_info</code> and <code>H5Oget_info</code> functions fill
this niche by returning a description of the object or link in an
<code>H5L_info_t</code> or <code>H5O_info_t</code> structure.</p>

```

```

<!-- <p> ??????? the rest of section 6 has been commented out </p> -->

```

```

<!-- start of a section of 6 that has been commented out
These structures contains the following information: </p>

```

```

<p> ??????? what about the new structures? do they also contain the following information? ??????? </p>

```

```

<p> ??????? MEE: because more work needs to be done to translate the old version to the new version,
I think we should comment out most of this section starting with

```

```

the sentence above "These structures contain the following information:" and including everything down
to the start of the next section. The next section is

```

```

chapter 7 "Discovering Objects in a Group". ??????? </p>

```

```

<ul>

```

```

<li>The file and object identifiers, which together provide unique
identification of the object

```



```
-->
<!-- end of a section of 6 that has been commented out -->
<!-- editingComment
  <span class="editingComment"><br />
    [ [ [
Why are there 2 of each?
    ] ] ]
  </span>
-->
<!-- start of another section of 6 that has been commented out -->
<!--
  <li>The number of references, or hard links, to the object
  <li>The object type: group, dataset, named datatype, or soft link,
    returned as <code>H5G_GROUP</code>, <code>H5G_DATASET</code>,
<code>H5G_TYPE</code>, or <code>H5G_LINK</code>, respectively
  <li>The modification time (datasets only)
  <li>A link length value; the length of the path name of a symbolic
    link's target object
(returned for symbolic links, or soft links, only)
</ul>

<p>
The <code>H5G_stat_t</code> structure specification and the
<code>H5Gget_objinfo</code> function signature appear in Figure 17.
The <code>H5G_stat_t</code> structure elements are as listed above.
The <code>H5Gget_objinfo</code> function parameters are used follows:
<ul>
  <li><code><em>loc_id</em></code> specifies the object for which
    information being sought.
  <li>A path to the object is returned in <code><em>name</em></code>.
-->
```

<!-- end of a section of 6 that has been commented out -->

<!-- editingComment

"A path" -- Per Quincey, this tries to be the first path,

but it becomes problematic when an object has moved around much.

-->

<!-- beginning of a section of 6 that has been commented out

<code>follow_link</code> is a Boolean value specifying

whether to follow a soft link and open the target object

(<code>TRUE</code>) or not (<code>FALSE</code>).

The <code>H5G_stat_t</code> struct is returned in the

<code>statbuf</code> buffer.

<p>

<table x-use-null-cells

width="600"

cellspacing="0"

class="fullImgTable"

align="center">

<tr valign="top">

<td class="fullImgTableImgCell">

<pre>

```
typedef struct H5G_stat_t {
```

```
    unsigned long fileno[2];
```

```
    unsigned long objno[2];
```

```
    unsigned nlink;
```

```
    int type;
```

```
    time_t mtime;
```

```
    size_t linklen;
```

```
} H5G_stat_t
```

```
herr_t H5Gget_objinfo(hid_t loc_id, const char *name, hbool_t follow_link, H5G_stat_t *statbuf )
```

```
</pre>
```

```
</td></tr>
```

```
<tr>
```

```
<td align=center class="fullImgTableCapCell">
```

```
  <span class="figureNumber">Figure 17.
```

```
  The <code>H5G_stat_t</code> struct specification and
```

```
  the <code>H5Gget_objinfo</code> function signature</span>
```

```
</td></tr>
```

```
</table>
```

```
-->
```

```
<!-- end of another section of 6 that has been commented out -->
```

```
<!-- NEW <> PAGE -->
```

```
<!-- part of section 6 that is commented out <p>
```

Figure 18 provides a code example that prints the local paths

to the members of a group, following a soft link when it is found.

```
<p>
```

```
<table x-use-null-cells
```

```
width="600"
```

```
cellspacing="0"
```

```
    class="fullImgTable"
```

```
align="center">
```

```
<tr valign="top">
```

```
<td class="fullImgTableImgCell">
```

```
<pre><code>
```

```
    H5G_stat_t statbuf;
```

```
    H5Gget_objinfo(loc_id, name, FALSE, &statbuf);
```

```
    switch (statbuf.type) {
```

```
case H5G_GROUP:
    printf(" Object with name %s is a group \n", name);
    break;
case H5G_DATASET:
    printf(" Object with name %s is a dataset \n", name);
    break;
case H5G_TYPE:
    printf(" Object with name %s is a named datatype \n", name);
    break;
case H5G_LINK:
    lname = (char *)malloc(statbuf.linklen);

    H5Gget_linkval(loc_id, name, statbuf.linklen, lname);
    printf(" Object with name %s is a link to %s \n", name, lname);
    H5Gget_objinfo(loc_id, name, TRUE, &statbuf);
    switch (statbuf.type) {
        case H5G_GROUP:
            printf(" Target of link name %s is a group \n", name);
            break;
        case H5G_DATASET:
            printf(" Target of link name %s is a dataset \n", name);
            break;
        case H5G_TYPE:
            printf(" Target of link name %s is a named datatype \n", name);
            break;
        case H5G_LINK:
            printf(" Target of link name %s is a soft link \n", name);
            break;
        default:
            printf(" Unable to identify target ");
    }
}
```

```

        break;
    default:
        printf(" Unable to identify an object ");
    }
</code></pre>
</td></tr>
<tr>
<td align=center class="fullImgTableCapCell">
    <span class="figureNumber">Figure 18.
    Printing a specified object's name and type and,
    in the case of a link, opening the target object</span>
</td></tr>
</table>
<p> ??????? H5Gget_linkval is mentioned above. it was deprecated in 1.8. this section above needs to be
rewritten. ??????? </p>

-->

<!-- end of section 6 text that is commented out -->

<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="DiscoverGrObjs">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>

```


<h4 class=pagebefore>4.5.7 Discovering Objects in a Group</h4>

<!-- the paragraph below no longer mentions the previously included alternate way because the alternate way used deprecated functions. -->

<!-- the functions that replace the deprecated functions could be included if the text is edited. -->

<p>To examine all the objects or links in a group,
use the `H5Literate` or `H5Ovisit` functions to
examine the objects, and use
the `H5Lvisit` function to examine the links.
`H5Literate` is useful both with a single group and
in an iterative process that examines an entire file
or section of a file (such as the contents of a group or the contents
of all the groups that are members of that group)
and acts on objects as they are encountered. `H5Ovisit`
recursively visits all objects accessible from a specified object.
`H5Lvisit` recursively visits all the links starting from a
specified group. </p>

<!-- start of commenting out the rest of section 7. the commented out parts use deprecated functions to
look at objects. this section can be used if the deprecated functions
are replaced and the text edited.

<p>

An alternative approach is to determine the number of objects
in a group then approach them one at a time.

This is accomplished with the functions

`H5Gget_num_objs`,
`H5Gget_objname_by_idx`, and

```
<code>H5Gget_objtype_by_idx</code>.
```

??????? the three H5G functions above were deprecated in 1.8. this section needs to be rewritten
 ?????? </p>

<p>

`H5Gget_num_objs` retrieves the number of objects,
 say `n`, in the group.

The values from `0` through `n - 1`

can then be used as indices to access the members of the group.

For example,

an index value of `0` identifies the first member,

an index value of `1` identifies the second member, and

an index value of `n - 1` identifies the last member.

(Note that HDF5 objects do not have permanent indices;

these values are strictly transient and may be different each time a
 group is opened.)

<p>

Using the index described above, the name and object type can
 be retrieved using `H5Gget_objname_by_idx` and
`H5Gget_objtype_by_idx`, respectively.

With the name and object type, an application can proceed to
 operate as necessary on all or selected group members.

```
-->
```

```
<!-- end of commenting out of section 7 -->
```

```
<!-- editingComment
```

```
    Need examples (and maybe illustration?).
```

```
-->
```

```
<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="DiscoverAll">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>
```



```
<a name="DiscoverAll">
<h4 class=pagebefore>4.5.8 Discovering All the Objects in the File</h4>
</a>
```



```
<p>The structure of an HDF5 file is
<span class="termDefinition">self-describing</span>,
meaning that an application can navigate an HDF5 file
to discover and understand all the objects it contains.
This is an iterative process wherein the structure is traversed as a graph,
starting at one node and recursively visiting linked nodes.
To explore the entire file, the traversal should start at the root group.</p>
```



```
<!-- the rest of section 8 describes how to use H5Giterate. H5Literate replaces H5Giterate. the parame-
ters aren't exactly the same, so I don't want to
just do a 1:1 replacing of old with new. so, the rest of this section should be commented out.-->
<!-- start commenting out part of section 8
```


<p>

The function `H5Giterate`, used to discover the members of a group, is the key to the discovery process.

An application calls `H5Giterate` with a pointer to a callback function (see Figure 19).

The HDF5 Library iterates through the group specified by the `loc_id` and `name` parameters, calling the callback function once for each group member.

The callback function must have the signature defined by `H5G_iterate_t`.

When invoked, the arguments to the callback function are
 the group being iterated,
 the group member's name (the object name), and
 a pointer set by the user program.

The callback function is part of the application,
 so it can execute any actions the program requires
 to discover and store information about the objects.</p>

<p>

<table x-use-null-cells

width="600"

cellspacing="0"

 class="fullImgTable"

align="center">

<tr valign="top">

<td class="fullImgTableImgCell">

<pre>

```
typedef herr_t (*H5G_iterate_t)(hid_t group_id, const char *member_name,
```

```

    void *operator_data);
H5Giterate(hid_t loc_id, const char * name, int *idx, H5G_iterate_t operator,
    void *operator_data );
</pre>
</td></tr>
<tr>
<td align=center class="fullImgTableCapCell">
    <span class="figureNumber">Figure 19. </span>
</td></tr>
</table>

```

<p>

Note that the `H5Giterate` function follows the links from a single group and that

these links correspond to the components in a path name.

To iterate over an entire substructure, `H5Giterate` must be called recursively on every member of the original group that turns out to also be a group.

To iterate over an entire file, the first call to `H5Giterate` must iterate over the root group; subsequent calls to `H5Giterate` must then iterate over every subsequent group.

-->

<!-- end commented out part of section 8 -->

<!-- NEW <> PAGE -->

<!-- start commenting out part of section 8

<p>

Figure 20 illustrates the relationship between the calling module of the application, the callback function (`do_obj`), and calls to the HDF5 Library.

In this diagram, “Global Variables and Functions” symbolizes the fact that the callback function executes as part

of the application, and may therefore call functions and update data structures to describe the file and its objects.

```
<p>
<table x-use-null-cells
width="600"
cellspacing="0"
        class="fullImgTable"
align="center">
  <tr valign="top">
    <td class="fullImgTableImgCell" align="center">
      
    </td></tr>
  <tr>
    <td align=center class="fullImgTableCapCell">
      <span class="figureNumber">Figure 20.
        Relationships between a calling module, the callback function,
        and the callback function's calls back to the HDF5 library</span>
    </td></tr>
  </table>
-->
<!-- end commented out part of section 8 -->
```

```
<!-- NEW <> PAGE -->
```

```
<!-- start commenting out part of section 8
```

```
<p>
Figure 21 illustrates the sequence of events precipitated by an
<code>H5Giterate</code> call.
<ol>
  <li>The application first calls <code>H5Giterate</code>,</li>
```

passing a pointer to a callback function

(`do_obj` in the figure).

Note that the callback function is part of the application.

The HDF5 Library then iterates through the members of the group, calling the callback function in the application once for each group member.

When the iteration is complete, the `H5Giterate` call returns to the calling application.

<p>

Figure 22 shows the sequence of calls involved in one iteration of a callback function that employs the HDF5 function `H5Gget_objinfo` to discover properties of the object that is the subject of the current step of the iteration (e.g., the object's type and reference count). The HDF5 Library then calls the application's callback function `do_obj()`, which in turn calls the HDF5 Library to get the object information.

The callback function can process the information as needed, accessing any function or data structure of the application program, and it can call the HDF5 Library again to, for example, iterate through a group member that is itself a group.

<p>

```
<table x-use-null-cells
width="600"
cellspacing="0"
      class="fullImgTable"
align="center">
  <tr valign="top">
    <td class="fullImgTableImgCell" align="center">
      
    </td></tr>
  <tr>
    <td height="24" align=center class="fullImgTableCapCell">
      <span class="figureNumber">Figure 22. </span> </td>
    </tr>
</table>
```

-->

<!-- end commented out part of section 8 -->


```
</tr>
</table>
```

```
<p>
```

The overall sequence of calls can become quite complex, especially when the callback function in turn calls the HDF5 Library.

Figure 24 provides a sequence diagram for a case similar to the simple case described above:

```
<ol>
```

- The calling program invokes `H5Giterate` on a group,
- which calls `do_obj` once for each group members
(three group members in this case).

- The `do_obj` callback function in turn calls
`H5Gget_objinfo` each time it is invoked to discover
information about each object.

```
</ol>
```

```
<p>
```

```
<table x-use-null-cells
width="600"
cellspacing="0"
      class="fullImgTable"
align="center">
  <tr valign="top">
    <td class="fullImgTableImgCell" align="center">
      
    </td></tr>
  <tr>
    <td height="24" align=center class="fullImgTableCapCell">
      <span class="figureNumber">Figure 24. </span> </td>
    </tr>
```

</table>

-->

<!-- end commented out part of section 8 -->

<!-- NEW <> PAGE -->

<!-- start commenting out part of section 8

<p>

Recursively iterating through the members of every group will result in visiting an object once for each link to it.

This may result in visiting an object more than once.

The calling application must be prepared to recognize this case and handle it appropriately.

If an action should be undertaken only once per object, the application must make sure that it does not repeat the action for an object with two links.

For example, if the objects are being copied, it is important that an object with two names be copied once, not twice.

Figure 25 illustrates this case.

<p>

<table x-use-null-cells

width="600"

cellspacing="0"

class="fullImgTable"

align="center">

<tr valign="top">

<td colspan="2" align="center" class="fullImgTableImgTopCell">

a) The required action is to copy all the objects from one file to another.</td>


```

</tr>
<tr valign="top">
  <td align="center" class="fullImgTableImgBottomCell">
    
  </td>
  <td align="center" class="fullImgTableImgBottomCell">
    
  </td>
</tr>
<tr>
  <td align="center">
    b) A shared dataset should not be copied twice.
  </td>
  <td align="center">
    c) A shared dataset should be copied once and
    the appropriate link should be created.
  </td>
</tr>
</table>

```

```
-->
```

```
<!-- end commented out part of section 8 -->
```

```
<!--
```

```

<table x-use-null-cells
width="600"
cellspacing="0"
        class="fullImgTable"
align="center">
    <tr valign="top">
        <td class="fullImgTableImgCell" align="center">
            
        </td></tr>
    <tr>
        <td height="24" align=center class="fullImgTableCapCell">
            <span class="figureNumber">Figure 26. </span> </td>
        </tr>
</table>
-->

```

```

<!-- editingComment Revisit right-alignment of following table.
    It's centered for now simply because right-alignment yields an
    over-write that I don't have time to fix right now.
    Remove the width definition at that point.      -->

```

```

    <!-- start commenting out part of section 8
<table x-use-null-cells
cellspacing="0"
width="600"
align="center">
    <tr valign="top">
        <td class="fullImgTableImgCell" align="center">
            
        </td></tr>
    <tr>

```

```

<td height="24" align=center class="fullImgTableCapCell">
  <span class="figureNumber">Figure 26. </span> </td>
</tr>
</table>

```

There is a second important case when the twice-visited member is a group.

Any group with more than one link to it can potentially be part of a circular path.

I.e., recursively iterating through member groups may eventually bring the iteration back to the current group and may generate an infinite path within the file's linked structure.

To embark upon the resulting infinite iteration would clearly be unacceptable in the general case.

Figure 26 illustrates an HDF5 file with such potential.

<p>

In such a case, the callback function should check the reference count in the `H5G_stat_t` buffer as returned by `H5Gget_objinfo`.

If the count is greater than one, there is more than one path to the object in question and it may be in a loop; the program should act accordingly.

For example, it may be necessary to construct a global table of all the objects visited.

Note that the object's name is not unique, but the full path and the object number (found in the above-mentioned `H5G_stat_t` buffer) are unique within an individual HDF5 file.

-->

<!-- end commented out part of section 8 -->

```
<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="Examples">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>
<br />
<!-- NEW PAGE -->
<a name="Examples">
<h3 class=pagebefore>4.6. Examples of File Structures</h3></a>
<p>This section presents several samples of HDF5 file structures.</p>
```

```
<table width="600" cellspacing="0" align="center">  
    <tr><td colspan="3"><hr color="green" size="3" /></td></tr>  
    <tr valign="top">  
        <td align="center">  
              
        </td>  
        <td align="center" valign="top">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>  
        <td align="center">  
              
        </td>
```

```

    </tr>
<tr>
    <td align="center" width="50%">
        a) The file contains three groups:
        the root group, <code>/group1</code>, and <code>/group2</code>.
    </td>
    <td align="center" valign="top">&nbsp;&nbsp;&nbsp;</td>
    <td align="center" width="50%">
        b) The dataset <code>dset1</code> (or <code>/group1/dset1</code>)
        is created in <code>/group1</code>.
    </td>
</tr>
<tr valign="top">
    <td align="center">
        
    </td>
    <td align="center" valign="top">&nbsp;&nbsp;&nbsp;</td>
    <td align="center">
        
    </td>
</tr>
<tr>
    <td align="center" valign="top">
        c) A link named <code>dset2</code> to the same dataset
        is created in <code>/group2</code>.
    </td>
    <td align="center" valign="top">&nbsp;&nbsp;&nbsp;</td>
    <td align="center">
        d) The link from <code>/group1</code> to <code>dset1</code> is removed.
        The dataset is still in the file, but can be accessed only as
        <code>/group2/dset2</code>.
    </td>

```

```

</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr>
<td colspan="3" align="left">
<b>Figure 9. Some file structures</b></td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

```

<p>

```

The figure above shows examples of the structure of a file with three groups and one dataset. The file in Figure 9a contains three groups: the root group and two member groups.

In Figure 9b, the dataset `dset1` has been created in `/group1`.

In Figure 9c, a link named `dset2` from `/group2` to the dataset has been added.

Note that there is only one copy of the dataset; there are two links to it and it can be accessed either as `/group1/dset1` or as `/group2/dset2`.

```

<p>

```

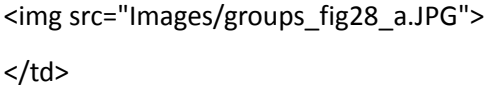
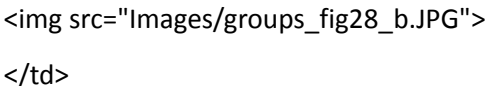
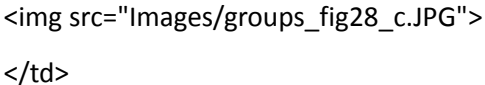
Figure 9d above illustrates that one of the two links to the dataset can be deleted.

In this case, the link from `/group1` has been removed.

The dataset itself has not been deleted; it is still in the file

but can only be accessed as `/group1/dset2`.

<!-- NEW PAGE -->

<hr style="border: 1px solid green;"/>		
<div style="text-align: center;">  </div>	 	<div style="text-align: center;">  </div>
<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>a) <code>dset1</code> has two names:</p> <p><code>/group2/dset1</code> and <code>/group1/GXX/dset1</code>.</p> </div> <div style="width: 45%;"> <p>b) <code>dset1</code> again has two names:</p> <p><code>/group1/dset1</code> and <code>/group1/dset2</code>.</p> </div> </div>		
<div style="text-align: center;">  </div>	 	

```

<td align="center">

</td>
</tr>
<tr>
<td align="center" valign="top">
c) <code>dset1</code> has three names:
<code>/group1/dset1</code>, <code>/group2/dset2</code>,
and <code>/group1/GXX/dset2</code>.
</td>
<td align="center" valign="top">&nbsp;&nbsp;&nbsp;</td>
<td align="center">
d) <code>dset1</code> has an infinite number of available path names.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr>
<td colspan="3" align="left">
<b>Figure 10. More sample file structures</b>
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

```

<p>

```

The figure above illustrates loops in an HDF5 file structure.

The file in Figure 10a contains three groups and a dataset;

<code>group2</code> is a member of the root group and of the

root group's other member group, `group1`.

`group2` thus can be accessed by either of two paths:

`/group2` or `/group1/GXX`.

Similarly, the dataset can be accessed either as

`/group2/dset1` or as `/group1/GXX/dset1`.

Figure 10b illustrates a different case: the dataset is a

member of a single group but with two links, or names, in that group.

In this case, the dataset again has two names,

`/group1/dset1` and `/group1/dset2`.

In Figure 10c, the dataset `dset1` is a member of two groups, one of which can be accessed by either of two names.

The dataset thus has three path names: `/group1/dset1`, `/group2/dset2`, and `/group1/GXX/dset2`.

And in Figure 10d, two of the groups are members of each other and the dataset is a member of both groups.

In this case, there are an infinite number of paths to the dataset because `GXX` and `GYY` can be traversed any number of times on the way from the root group, `/`, to the dataset.

This can yield a path name such as

`/group1/GXX/GYY/GXX/GYY/GXX/dset2`.

<!-- NEW PAGE -->

```

<table width="600" cellpadding="0" align="center">
  <tr><td colspan="3"><hr color="green" size="3" /></td></tr>
  <tr valign="top">
    <td align="center">
      </td>
    <td align="center" valign="top">&nbsp;&nbsp;&nbsp;</td>
    <td align="center">
      </td>
    </tr>
  <tr>
    <td align="center" valign="top" width="48%">
      a) The file contains only hard links.</td>
    <td align="center" valign="top" width="48%">
      b) A soft link is added from <code>group2</code>
      to <code>/group1/dset1</code>.</td>
    </tr>
  <tr valign="top">
    <td align="center">
      </td>
    <td align="center">
      </td>
    </tr>
  <tr>
    <td align="center" valign="top">
      c) A soft link named <code>dset3</code> is added with a target
      that does not yet exist.</td>
    <td align="center" valign="top">&nbsp;&nbsp;&nbsp;</td>
    <td align="center" valign="top">
      d) The target of the soft link is created or linked.</td>
  </tr>

```

```

    </tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr>
    <td colspan="3" align="left">
        <b>Figure 11. Hard and soft links</b></td>
    </tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```
<!-- NEW PAGE -->
```

```
<p>
```

The figure above takes us into the realm of soft links.

The original file, in Figure 11a, contains only three hard links.

In Figure 11b, a soft link named `dset2` from `group2` to `/group1/dset1` has been created, making this dataset accessible as `/group2/dset2`.

```
<p>
```

In Figure 11c, another soft link has been created in `group2`.

But this time the soft link, `dset3`, points to a target object that does not yet exist.

That target object, `dset`, has been added in Figure 11d and is now accessible as either `/group2/dset` or `/group2/dset3`.

```
<SCRIPT language="JavaScript">
```

```
<!--
document.writeln ("
<a name="Appendix">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>

<!-- editingComment -- Removed until we can make sure it's (1) finished and (2) right.

<h3 class=pagebefore>10. Appendix: Mapping HDF5 to a Graph</h3>

<div>

  <span class="editingComment">
    [ [ [
Unless someone says it ought to be otherwise,
this section will be commented out for this release.
    ] ] ]
  </span>

  <p><em>Is this material wanted? It will take some effort to flesh this out
and make sure it is correct.</em></p>

<table x-use-null-cells
class="ColumnTable"
width="600"
cellspacing="0"
align="center">
```

```

<tr valign="top">
  <td class="ColumnHdr">
    <p><span class="TableHead">Box 2: Applying graph algorithms to HDF5</span>
    <p>It follows from the observation that an HDF5 can be mapped to a rooted directed graph, that graph
    algorithms can be applied to the structure of the HDF5 file. For example, the objects of the HDF5 file can
    be visited by "traversing" the graph.
    <h4><em>Algorithm 1: Depth First Traversal</em></h4>
    <pre>
      visit_df (Group g) {
        process (g); // a pre-order traversal
        for all m in g.members() {
          if (m is a group) {
            visit_df(m); // recur
          } else {
            process (m); // m is a dataset or named datatype
          }
        }
      }
    </pre>
  </td>
</tr>
</table>

<div align="right">
  <a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</div>

-->

<br /><br /><br />
<!-- FOR USE WITH PRINT VERSION -----
<br /><br /><br /><br /><br /><br /><br /><br /><br />

```

```
<br /><br /><br /><br /><br /><br /><br /><br /><br />
<br /><br /><br /><br /><br /><br /><br /><br /><br />
<br /><br /><br /><br /><br />
<!-- FOR USE WITH PRINT VERSION ----->
```

```
<a name="FootNote"><hr width="200px" align="left"></a>
```

```
<p><font size="-1"><sup><a href="#SupScript1">1</a></sup>It could be said
that HDF5 extends the organizing concepts of a file system to the
internal structure of a single file.</font></p>
```

```
<!-- HEADER RIGHT " " -->
```

```
</body>
```

```
</html>
```

I

```

<!doctype HTML public "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
<head>
<title>Chapter 5: HDF5 Datasets</title>

<!--(Meta)=====-->

<!--(Links)=====-->

<!--( Begin styles definition )=====-->
<link href="ed_styles/NewUGelect.css" rel="stylesheet" type="text/css">
<!--( End styles definition )=====-->

</head>

<body>

<!-- #BeginLibraryItem "/ed_libs/styles_UG.lbi" -->
<!--
* * * * *
* Copyright by The HDF Group. *
* Copyright by the Board of Trustees of the University of Illinois. *
* All rights reserved. *
*
* This file is part of HDF5. The full HDF5 copyright notice, including *
* terms governing use, modification, and redistribution, is contained in *
* the files COPYING and Copyright.html. COPYING can be found at the root *
* of the source code distribution tree; Copyright.html can be found at the *
* root level of an installed copy of the electronic HDF5 document set and *
* is linked from the top-level documents page. It can also be found at *
* http://www.hdfgroup.org/HDF5/doc/Copyright.html. If you do not have *

```

```

* access to either file, you may request a copy from help@hdfgroup.org.  *
* * * * *
-->

<!-- #EndLibraryItem --><!-- HEADER LEFT "HDF5 User's Guide" -->
<!-- HEADER RIGHT "HDF5 Datasets" -->

<!-- ( TOC )=====-->
<!--<SCRIPT language="JavaScript">-->
<!--
document.writeln ('\
<table x-use-null-cells\
        align=right\
width=240\
cellspacing="0"\
class="tocTable">\
-->
<!-- Table Version 3 --><!-- \ -->
<!--
<tr valign="top"> \
    <td class="tocTableHeaderCell" colspan="2"> \
        <span class="TableHead">Chapter Contents</span></td>\
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#Intro">1.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#Intro">Introduction</a></td> \
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#FileFunctSums">2.</a></td>\

```

```

    <td class="tocTableContentCell3">\
    <a href="#FileFuncSums">Dataset (H5D) Function Summaries</a></td>\
  </tr>\
  <tr valign="top"> \
    <td class="tocTableContentCell2"> \
      <a href="#PModel">3.</a></td>\
    <td class="tocTableContentCell3">\
    <a href="#PModel">Programming Model</a></td> \
  </tr>\
  <tr valign="top"> \
    <td class="tocTableContentCell2"> \
      <a href="#DTransfer">4.</a></td>\
    <td class="tocTableContentCell3">\
    <a href="#DTransfer">Data Transfer</a></td>\
  </tr>\
  <tr valign="top"> \
    <td class="tocTableContentCell2"> \
      <a href="#Allocation">5.</a></td>\
    <td class="tocTableContentCell3">\
    <a href="#Allocation">Allocation of Space</a></td>\
  </tr>\
  <tr valign="top"> \
    <td class="tocTableContentCell"> \
      <a href="#UseFilters">6.</a></td>\
    <td class="tocTableContentCell4">\
    <a href="#UseFilters">Specialized Filters</a>\
      <br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
    <a href="#N-Bit">N-bit</a>\
      <br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;\
    <a href="#ScaleOffset">Scale-offset</a></td>\
  </tr>

```

```
</td></tr>\n</table>\n')\n-->\n<!--</SCRIPT>-->\n<!--(End TOC)=====-->\n\n<!-- editingComment\n  <span class="editingComment">[ [ [\n    ] ] ]</span>\n-->\n\n<!-- editingComment\n-->\n\n<div align="center">\n<a name="TOP">
```

6. HDF5 Datasets

```
</div>
```

```
<!-- editingComment
```

```
<span class="editingComment">[ [ [
```

```
Original title. Which is proper?
```

```
<h2>10. Datasets I/O</h2>
```

```
] ] ]</span>
```

```
-->
```

```
<br />
```

```
<a name="Intro">
```

```
<h3>5.1. Introduction</h3>
```

```
</a>
```

An HDF5 dataset is an object composed of a collection of data elements, or raw data, and metadata that stores a description of the data elements, data layout, and all other information necessary to write, read, and interpret the stored data. From the viewpoint of the application the raw data is stored as a one-dimensional or multi-dimensional array of elements (the *raw data*), those elements can be any of several numerical or character types, small arrays, or even compound types similar to C structs. The dataset object may have attribute objects. See the figure below.

```
<table width="600" cellspacing="0" align="center">
```

```
<tr valign="top">
```

```
<td align="center">
```

```
<hr color="green" size="3"/>
```

```

    
  </td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 1. Application view of a dataset</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

<!-- editingComment
<span class="editingComment">
<p>Datatypes are described in [ [ [ "Datatypes" ] ] ], and in the [ [ [ "HDF5
Datatypes" chapter in this guide ] ] ], Dataspace objects are described in
[ [ [ Dataspace ] ] ], and Attributes are described in [ [ [ Attributes ] ] ].
</span>
-->

```

A dataset object is stored in a file in two parts: a header and a data array. The header contains information that is needed to interpret the array portion of the dataset, as well as metadata (or pointers to metadata) that describes or annotates the dataset. Header information includes the name of the object, its dimensionality, its number-type, information about how the data itself is stored on disk (the *storage layout*), and other information used by the library to speed up access to the dataset or maintain the file's integrity.

The HDF5 dataset interface, comprising the H5D functions, provides a mechanism for managing HDF5 datasets including the transfer of data

between memory and disk and the description of dataset properties. </p>

<p>A dataset is used by other HDF5 APIs, either by name or by an identifier (e.g., returned by <code>H5Dopen</code>).</p>

<h4>Link/Unlink</h4>

<p>A dataset can be added to a group with one of the <code>H5Lcreate</code> calls, and deleted from a group with <code>H5Ldelete</code>. The link and unlink operations use the name of an object, which may be a dataset. The dataset does not have to open to be linked or unlinked.</p>

<h4>Object reference</h4>

<p>A dataset may be the target of an object reference. The object reference is created by <code>H5Rcreate</code> with the name of an object which may be a dataset and the reference type <code>H5R_OBJECT</code>. The dataset does not have to be open to create a reference to it.</p>

<p>An object reference may also refer to a region (selection) of a dataset. The reference is created with <code>H5Rcreate</code> and a reference type of <code>H5R_DATASET_REGION</code>.</p>

<p>An object reference can be accessed by a call to <code>H5Rdereference</code>. When the reference is to a dataset or dataset region, the <code>H5Rdereference</code> call returns an identifier to the dataset just as if <code>H5Dopen</code> has been called.</p>

<h4>Adding attributes</h4>

A dataset may have user-defined attributes which are created with `H5Acreate` and accessed through the H5A API. To create an attribute for a dataset, the dataset must be open, and the identifier is passed to `H5Acreate`. The attributes of a dataset are discovered and opened using `H5Aopen_name`, `H5Aopen_idx`, or `H5Aiterate`; these functions use the identifier of the dataset. An attribute can be deleted with `H5Adelete` which also uses the identifier of the dataset.

<!-- editingComment

<p>The remaining sections of this chapter discuss... [To be written last.]</p>

-->

<SCRIPT language="JavaScript">

<!--

document.writeln ("

<div align=right>

(Top)

</div>

");

-->

</SCRIPT>

<!-- NEW PAGE -->

5.2. Dataset Function Summaries

Functions that can be used with datasets (H5D functions) and property list functions that can be used with datasets (H5P functions) are listed below.

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Dcreate_anon
h5dcreate_anon_f</code>

</td><td> </td>

<td>

Creates a dataset in a file without linking it into the file structure.

</td>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Dopen
h5dopen_f</code>

</td><td> </td>

<td>

Opens an existing dataset. The C function is a

macro: see “API

Compatibility Macros in HDF5.”

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Dclose
h5dclose_f</code>

</td><td> </td>

<td>

Closes the specified dataset.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<div data-bbox="240 182 841 327"> <code><td></code> <code><code>H5Dget_space
h5dget_space_f</code></code> <code></td><td>&nbsp;</td></code> </div>	<div data-bbox="240 352 974 497"> <code><td></code> Returns an identifier for a copy of the dataspace for a dataset. <code></td></code> </div>
<div data-bbox="240 522 305 552"> <code></tr></code> </div>	
<div data-bbox="215 577 941 606"> <code><tr><td colspan="3"><hr color="green" size="1" /></td></tr></code> </div>	
<div data-bbox="215 632 420 661"> <code><tr valign="top"></code> </div>	
<div data-bbox="240 686 1011 831"> <code><td></code> <code><code>H5Dget_space_status
h5dget_space_status_f</code></code> <code></td><td>&nbsp;</td></code> </div>	<div data-bbox="240 856 951 1001"> <code><td></code> Determines whether space has been allocated for a dataset. <code></td></code> </div>
<div data-bbox="240 1026 305 1056"> <code></tr></code> </div>	
<div data-bbox="215 1081 941 1110"> <code><tr><td colspan="3"><hr color="green" size="1" /></td></tr></code> </div>	
<div data-bbox="215 1136 420 1165"> <code><tr valign="top"></code> </div>	
<div data-bbox="240 1190 812 1335"> <code><td></code> <code><code>H5Dget_type
h5dget_type_f</code></code> <code></td><td>&nbsp;</td></code> </div>	<div data-bbox="240 1360 961 1505"> <code><td></code> Returns an identifier for a copy of the datatype for a dataset. <code></td></code> </div>
<div data-bbox="240 1530 305 1560"> <code></tr></code> </div>	
<div data-bbox="215 1585 941 1614"> <code><tr><td colspan="3"><hr color="green" size="1" /></td></tr></code> </div>	
<div data-bbox="215 1640 420 1669"> <code><tr valign="top"></code> </div>	
<div data-bbox="240 1694 984 1839"> <code><td></code> <code><code>H5Dget_create_plist
h5dget_create_plist_f</code></code> <code></td><td>&nbsp;</td></code> </div>	<div data-bbox="240 1864 305 1894"> <code><td></code> </div>

Returns an identifier for a copy of the dataset creation property

list for a dataset.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Dget_access_plist
(none)</code>

</td><td> </td>

<td>

Returns the dataset access property list associated with a dataset.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Dget_offset
h5dget_offset_f</code>

</td><td> </td>

<td>

Returns the dataset address in a file.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Dget_storage_size
h5dget_storage_size_f</code>

</td><td> </td>

<td>

Returns the amount of storage required for a dataset.

</td>

</tr>

<hr/>		
	<code>H5Dvlen_get_buf_size</code> <code>h5dvlen_get_max_len_f</code>	
	<p>Determines the number of bytes required to store variable-length (VL) data.</p>	

<hr/>		
	<code>H5Dvlen_reclaim</code> <code>h5dvlen_reclaim_f</code>	
	<p>Reclaims VL datatype memory buffers.</p>	

<!-- NEW PAGE -->

<hr/>		
	<code>H5Dread</code> <code>h5dread_f</code>	
	<p>Reads raw data from a dataset into a buffer.</p>	

<hr/>		

<td>

<code>H5Dwrite
h5dwrite_f</code>

</td><td> </td>

<td>

Writes raw data from a buffer to a dataset.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Diterate
(none)</code>

</td><td> </td>

<td>

Iterates over all selected elements in a dataspace.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Dgather
(none)</code>

</td><td> </td>

<td>

Gathers data from a selection within a memory buffer.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Dscatter
(none)</code>

</td><td> </td>

<td>

Scatters data into a selection within a memory buffer.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Dfill
h5dfill_f</code>

</td><td> </td>

<td>

Fills dataspace elements with a fill value in a memory buffer.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Dset_extent
h5dset_extent_f</code>

</td><td> </td>

<td>

Changes the sizes of a dataset's dimensions.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

</table>

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center" cellpadding="0">

```

<tr valign="bottom">
  <td colspan="3" align="left" valign="bottom">
    <b>Function Listing 2. Dataset creation property list functions (H5P)
  </b></td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
<tr valign="top">
  <td>
    <b>C Function<br>Fortran Function</b>
  </td><td>&nbsp;</td>
  <td>
    <b>Purpose</b>
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset_layout<br>h5pset_layout_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets the type of storage used to store the raw data for a dataset.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pget_layout<br>h5pget_layout_f</code>
  </td><td>&nbsp;</td>
  <td>
    Returns the layout of the raw data for a dataset.
  </td>

```



```

</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset_chunk<br>h5pset_chunk_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets the size of the chunks used to store a chunked layout dataset.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pget_chunk<br>h5pget_chunk_f</code>
  </td><td>&nbsp;</td>
  <td>
    Retrieves the size of chunks for the raw data of a chunked layout
    dataset.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset_deflate<br>h5pset_deflate_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets compression method and compression level.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">

```

<div data-bbox="240 182 911 548"> <div data-bbox="240 182 911 327"> <div data-bbox="240 182 911 237"><td></div> <div data-bbox="240 237 911 291"><code>H5Pset_fill_value
h5pset_fill_value_f</code></div> <div data-bbox="240 291 911 346"></td><td>&nbsp;</td></div> </div> <div data-bbox="240 346 911 491"> <div data-bbox="240 346 911 401"><td></div> <div data-bbox="240 401 911 455">Sets the fill value for a dataset.</div> <div data-bbox="240 455 911 510"></td></div> </div> <div data-bbox="240 510 911 548"></tr></div> </div>	<div data-bbox="215 573 943 606"><tr><td colspan="3"><hr color="green" size="1" /></td></tr></div>	
<div data-bbox="215 630 919 1050"> <div data-bbox="215 630 919 827"> <div data-bbox="215 630 919 684"><tr valign="top"></div> <div data-bbox="215 684 919 827"> <div data-bbox="215 684 919 739"><td></div> <div data-bbox="215 739 919 793"><code>H5Pget_fill_value
h5pget_fill_value_f</code></div> <div data-bbox="215 793 919 848"></td><td>&nbsp;</td></div> </div> <div data-bbox="215 848 919 993"> <div data-bbox="215 848 919 903"><td></div> <div data-bbox="215 903 919 957">Retrieves a dataset fill value.</div> <div data-bbox="215 957 919 1012"></td></div> </div> <div data-bbox="215 1012 919 1050"></tr></div> </div> </div>	<div data-bbox="215 1075 943 1108"><tr><td colspan="3"><hr color="green" size="1" /></td></tr></div>	
<div data-bbox="215 1131 820 1551"> <div data-bbox="215 1131 820 1329"> <div data-bbox="215 1131 820 1186"><tr valign="top"></div> <div data-bbox="215 1186 820 1329"> <div data-bbox="215 1186 820 1241"><td></div> <div data-bbox="215 1241 820 1295"><code>H5Pfill_value_defined
(none)</code></div> <div data-bbox="215 1295 820 1350"></td><td>&nbsp;</td></div> </div> <div data-bbox="215 1350 820 1495"> <div data-bbox="215 1350 820 1404"><td></div> <div data-bbox="215 1404 820 1459">Determines whether the fill value is defined.</div> <div data-bbox="215 1459 820 1514"></td></div> </div> <div data-bbox="215 1514 820 1551"></tr></div> </div> </div>	<div data-bbox="215 1577 943 1610"><tr><td colspan="3"><hr color="green" size="1" /></td></tr></div>	
<div data-bbox="215 1633 894 1881"> <div data-bbox="215 1633 894 1831"> <div data-bbox="215 1633 894 1688"><tr valign="top"></div> <div data-bbox="215 1688 894 1831"> <div data-bbox="215 1688 894 1743"><td></div> <div data-bbox="215 1743 894 1797"><code>H5Pset_fill_time
h5pset_fill_time_f</code></div> <div data-bbox="215 1797 894 1852"></td><td>&nbsp;</td></div> </div> <div data-bbox="215 1852 894 1881"><td></div> </div> </div>		

Sets the time when fill values are written to a dataset.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pget_fill_time
h5pget_fill_time_f</code>

</td><td> </td>

<td>

Retrieves the time when fill value are written to a dataset.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_alloc_time
h5pset_alloc_time_f</code>

</td><td> </td>

<td>

Sets the timing for storage space allocation.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pget_alloc_time
h5pget_alloc_time_f</code>

</td><td> </td>

<td>

Retrieves the timing for storage space allocation.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

```

<tr valign="top">
  <td>
    <code>H5Pset_filter<br>h5pset_filter_f</code>
  </td><td>&nbsp;</td>
  <td>
    Adds a filter to the filter pipeline.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pall_filters_avail<br>(none)</code>
  </td><td>&nbsp;</td>
  <td>
    Verifies that all required filters are available.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pget_nfilters<br>h5pget_nfilters_f</code>
  </td><td>&nbsp;</td>
  <td>
    Returns the number of filters in the pipeline.
  </td>
</tr>
<!-- NEW PAGE -->
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pget_filter<br>h5pget_filter_f</code>

```

</td><td> </td>

<td>

Returns information about a filter in a pipeline. The C function is a macro: see “API Compatibility Macros in HDF5.”

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pget_filter_by_id
h5pget_filter_by_id_f</code>

</td><td> </td>

<td>

Returns information about the specified filter. The C function is a macro: see “API Compatibility Macros in HDF5.”

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pmodify_filter
h5pmodify_filter_f</code>

</td><td> </td>

<td>

Modifies a filter in the filter pipeline.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Premove_filter
h5premove_filter_f</code>

</td><td> </td>

<td>

Deletes one or more filters in the filter pipeline.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_fletcher32
h5pset_fletcher32_f</code>

</td><td> </td>

<td>

Sets up use of the Fletcher32 checksum filter.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_nbit
h5pset_nbit_f</code>

</td><td> </td>

<td>

Sets up use of the n-bit filter.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_scaleoffset
h5pset_scaleoffset_f</code>

</td><td> </td>

<td>

Sets up use of the scale-offset filter.

</td>

```

    </tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Pset_shuffle<br>h5pset_shuffle_f</code>
    </td><td>&nbsp;</td>
    <td>
        Sets up use of the shuffle filter.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Pset_szip<br>h5pset_szip_f</code>
    </td><td>&nbsp;</td>
    <td>
        Sets up use of the Szip compression filter.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Pset_external<br>h5pset_external_f</code>
    </td><td>&nbsp;</td>
    <td>
        Adds an external file to the list of external files.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>

```

```
<code>H5Pget_external_count<br>h5pget_external_count_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Returns the number of external files for a dataset.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pget_external<br>h5pget_external_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Returns information about an external file.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pset_char_encoding<br>h5pset_char_encoding_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Sets the character encoding used to encode a string.

Use to set ASCII or UTF-8 character encoding for object names.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pget_char_encoding<br>h5pget_char_encoding_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```


Retrieves the character encoding used to create a string.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

</table>

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="3" align="left" valign="bottom">

Function Listing 3. Dataset access property list functions (H5P)

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td>

C Function
Fortran Function

</td><td> </td>

<td>

Purpose

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_buffer
h5pset_buffer_f</code>

</td><td> </td>

<td>

Sets type conversion and background buffers.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pget_buffer
h5pget_buffer_f</code>

</td><td> </td>

<td>

Reads buffer settings.

</td>

</tr>

<!-- 8.10.10, MEE: I removed two dataset access property list functions:

H5Pset_preserve and H5Pget_preserve. -->

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_chunk_cache
h5pset_chunk_cache_f</code>

</td><td> </td>

<td>

Sets the raw data chunk cache parameters.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pget_chunk_cache
h5pget_chunk_cache_f</code>

</td><td> </td>

<td>

Retrieves the raw data chunk cache parameters.

</td>

```

    </tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Pset_edc_check<br>h5pset_edc_check_f</code>
    </td><td>&nbsp;</td>
    <td>
        Sets whether to enable error-detection when reading a dataset.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Pget_edc_check<br>h5pget_edc_check_f</code>
    </td><td>&nbsp;</td>
    <td>
        Determines whether error-detection is enabled for dataset reads.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Pset_filter_callback<br>(none)</code>
    </td><td>&nbsp;</td>
    <td>
        Sets user-defined filter callback function.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>

```

<code>H5Pset_data_transform</code> <code>h5pset_data_transform_f</code>	 Sets a data transform expression.
<hr/>	
<code>H5Pget_data_transform</code> <code>h5pget_data_transform_f</code>	 Retrieves a data transform expression.
<hr/>	
<code>H5Pset_type_conv_cb</code> <code>(none)</code>	 Sets user-defined datatype conversion callback function.
<hr/>	
<code>H5Pget_type_conv_cb</code> <code>(none)</code>	 Gets user-defined datatype conversion callback function.

```

</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset_hyper_vector_size<br>h5pset_hyper_vector_size_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets number of I/O vectors to be read/written in hyperslab I/O.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pget_hyper_vector_size<br>h5pget_hyper_vector_size_f</code>
  </td><td>&nbsp;</td>
  <td>
    Retrieves number of I/O vectors to be read/written in hyperslab I/O.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset_btree_ratios<br>h5pset_btree_ratios_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets B-tree split ratios for a dataset transfer property list.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">

```

```

<td>
<code>H5Pget_btree_ratios<br>h5pget_btree_ratios_f</code>
</td><td>&nbsp;</td>
<td>
Gets B-tree split ratios for a dataset transfer property list.
</td>
</tr>
<!-- NEW PAGE -->
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5Pset_vlen_mem_manager<br>(none)</code>
</td><td>&nbsp;</td>
<td>
Sets the memory manager for variable-length datatype allocation in
<code>H5Dread</code> and <code>H5Dvlen_reclaim</code>.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5Pget_vlen_mem_manager<br>(none)</code>
</td><td>&nbsp;</td>
<td>
Gets the memory manager for variable-length datatype allocation in
<code>H5Dread</code> and <code>H5Dvlen_reclaim</code>.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>

```

<code>H5Pset_dxpl_mpio</code> <code>h5pset_dxpl_mpio_f</code>	 Sets data transfer mode.
<hr/>	
<code>H5Pget_dxpl_mpio</code> <code>h5pget_dxpl_mpio_f</code>	 Returns the data transfer mode.
<hr/>	
<code>H5Pset_dxpl_mpio_chunk_opt</code> <code>(none)</code>	 Sets a flag specifying linked-chunk I/O or multi-chunk I/O.
<hr/>	
<code>H5Pset_dxpl_mpio_chunk_opt_num</code> <code>(none)</code>	 Sets a numeric threshold for linked-chunk I/O.

```

</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset_dxpl_mpio_chunk_opt_ratio<br>(none)</code>
  </td><td>&nbsp;</td>
  <td>
    Sets a ratio threshold for collective I/O.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset_dxpl_mpio_collective_opt<br>(none)</code>
  </td><td>&nbsp;</td>
  <td>
    Sets a flag governing the use of independent versus collective I/O.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset_multi_type<br>(none)</code>
  </td><td>&nbsp;</td>
  <td>
    Sets the type of data property for the MULTI driver.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">

```



```
<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="PModel">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>

<!-- NEW PAGE -->
<a name="PModel">
<h3 class=pagebefore>5.3. Programming Model</h3>
</a>

<p>This section explains the programming model for datasets.</p>
<br />
<h4>5.3.1. General Model</h4>

<p>The programming model for using a dataset has three main phases:</p>

<ul>
<li>Obtain access to the dataset </li>
<li>Operate on the dataset using the dataset identifier returned
```

at access

Release the dataset

<p>These three phases or steps are described in more detail below the figure.</p>

<p>A dataset may be opened several times and operations performed with several different identifiers to the same dataset. All the operations affect the dataset although the calling program must synchronize if necessary to serialize accesses.</p>

<p>Note that the dataset remains open until every identifier is closed. The figure below shows the basic sequence of operations.</p>

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 2. Dataset programming sequence</b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />
```

Creation and data access operations may have optional parameters which are set with property lists. The general programming model is:

- Create property list of appropriate class (dataset create, dataset transfer)

- Set properties as needed; each type of property has its own format and datatype

- Pass the property list as a parameter of the API call

The steps below describe the programming phases or steps for using a dataset.

Step 1. Obtain Access

A new dataset is created by a call to `H5Dcreate`. If successful, the call returns an identifier for the newly created dataset.

Access to an existing dataset is obtained by a call to `H5Dopen`. This call returns an identifier for the existing dataset.

An object reference

```
<!-- editingComment
```

```
<span class="editingComment">[ [
(Chapter ???)
]]</span>
```

```
-->
```

may be dereferenced to obtain an identifier to the dataset it points to.

In each of these cases, the successful call returns an identifier

to the dataset. The identifier is used in subsequent operations until the dataset is closed.

Step 2. Operate on the Dataset

The dataset identifier can be used to write and read data to the dataset, to query and set properties, and to perform other operations such as adding attributes, linking in groups, and creating references.

The dataset identifier can be used for any number of operations until the dataset is closed.

Step 3. Close the Dataset

When all operations are completed, the dataset identifier should be closed. This releases the dataset.

- Dataspace
- Datatype
- Storage properties

<p>These properties cannot be changed for the life of the dataset, although the dataspace may be expanded up to its maximum dimensions.</p>

<h4>Name</h4>

<p>A dataset name is a sequence of alphanumeric ASCII characters. The full name would include a tracing of the group hierarchy from the root group of the file, e.g., /rootGroup/groupA/subgroup23/dataset1. The local name or relative name within the lowest-level group containing the dataset would include none of the group hierarchy. e.g., Dataset1.</p>

<h4>Dataspace</h4>

<p>The dataspace of a dataset defines the number of dimensions and the size of each dimension.

<!-- editingComment

[[[[Dataspace]].]]]

-->

The dataspace defines the number of dimensions, and the maximum dimension sizes and current size of each dimension. The maximum dimension size can be a fixed value or the constant <code>H5D_UNLIMITED</code>, in which case the actual dimension size can be changed with calls to <code>H5Dset_extent</code>, up to the

maximum set with the `maxdims` parameter in the [H5Screate_simple](../RM/RM_H5S.html#Dataspace-CreateSimple) call that established the dataset's original dimensions. The maximum dimension size is set when the dataset is created and cannot be changed.

Datatype

Raw data has a datatype which describes the layout of the raw data stored in the file.

`<!-- editingComment`

`[[[`
`(See [[Datatype]].`
`]]]`

`-->`

The datatype is set when the dataset is created and can never be changed. When data is transferred to and from the dataset, the HDF5 Library will assure that the data is transformed to and from the stored format.

Storage Properties

Storage properties of the dataset are set when it is created. The required inputs table below shows the categories of storage properties. The storage properties cannot be changed after the dataset is created.

`<!-- editingComment`

`[[[`
`in [[storage properties]]`
`]]]`

`-->`

Filters

When a dataset is created, optional filters are specified. The filters are added to the data transfer pipeline when data is read or written. The standard library includes filters to implement compression, data shuffling, and error detection code. Additional user-defined filters may also be used.


```

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
<tr valign="top">
  <td width="25%"><b>Required Inputs</b></td>
  <td width="75%"><b>Description</b></td>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Dataspace</td>
  <td>The shape of the array.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Datatype</td>
  <td>The layout of the stored elements.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Name</td>
  <td>The name of the dataset in the group.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="2" align="left" valign="bottom">
      <b>Table 2. Optional inputs</b></td>
    </tr>
  <tr><td colspan="2"><hr color="green" size="3" /></td></tr>

```

```

<tr valign="top">
  <td width="25%"><b>Optional Inputs</b></td>
  <td width="75%"><b>Description</b></td>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Storage Layout</td>
  <td>How the data is organized in the file including chunking.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Fill Value</td>
  <td>The behavior and value for uninitialized data.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>External Storage</td>
  <td>Option to store the raw data in an external file.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Filters</td>
  <td>Select optional filters to be applied. One of the filters
    that might be applied is compression.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

<!-- NEW PAGE -->
<h4><em>Example</em></h4>

```

To create a new dataset, go through the following general steps:

- Set dataset characteristics (optional where default settings are acceptable)

 - Datatype

 - Dataspace

 - Dataset creation property list

- Create the dataset

- Close the datatype, dataspace, and property list (as necessary)

- Close the dataset

Example 1 below shows example code to create an empty dataset. The dataspace is 7 x 8, and the datatype is a big-endian integer. The dataset is created with the name "dset1" and is a member of the root group, "/".

--

--

--

--

```


```

```

hid_t dataset, datatype, dataspace;

```

```

/*
 * Create dataspace: Describe the size of the array and
 * create the dataspace for fixed-size dataset.
 */
dimsf[0] = 7;
dimsf[1] = 8;
dataspace = H5Screate_simple(2, dimsf, NULL);
/*
 * Define datatype for the data in the file.
 * For this example, store little-endian integer numbers.
 */
datatype = H5Tcopy(H5T_NATIVE_INT);
status = H5Tset_order(datatype, H5T_ORDER_LE);
/*
 * Create a new dataset within the file using defined
 * dataspace and datatype. No properties are set.
 */
dataset = H5Dcreate(file, "/dset", datatype, dataspace, H5P_DEFAULT,
                    H5P_DEFAULT, H5P_DEFAULT);

H5Dclose(dataset);
H5Sclose(dataspace);
H5Tclose(datatype);
```

</tr>

<tr><td><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td align="left">

Example 1. Create an empty dataset

<hr color="green" size="3"/></td>

</tr>

</table>

<!-- NEW PAGE -->

<p>Example 2 below shows example code to create a similar dataset with a fill value of ‘-1’.

This code has the same steps as in the example above, but uses a non-default property list. A file creation property list is created, and then the fill value is set to the desired value. Then the property list is passed to the <code>H5Dcreate</code> call.</p>

```
<table width="600" cellpadding="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="left">
```

```
      <hr color="green" size="3"/>
```

```
      <pre>
```

```
hid_t  dataset, datatype, dataspace;
```

```
hid_t plist; /* property list */
```

```
int fillval = -1;
```

```
dimsf[0] = 7;
```

```
dimsf[1] = 8;
```

```
dataspace = H5Screate_simple(2, dimsf, NULL);
```

```
datatype = H5Tcopy(H5T_NATIVE_INT);
```

```
status = H5Tset_order(datatype, H5T_ORDER_LE);
```

```
/*
```

```
 * Example of Dataset Creation property list: set fill value to '-1'
```

```
 */
```

```
plist = H5Pcreate(H5P_DATASET_CREATE);
```

```
status = H5Pset_fill_value(plist, datatype, &fillval);
```

```

/* Same as above, but use the property list */
dataset = H5Dcreate(file, "/dset", datatype, dataspace, H5P_DEFAULT,
    plist, H5P_DEFAULT);

H5Dclose(dataset);
H5Sclose(dataspace);
H5Tclose(datatype);
H5Pclose(plist);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 2. Create a dataset with fill value set</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

After this code is executed, the dataset has been created and written to the file. The data array is uninitialized. Depending on the storage strategy and fill value options that have been selected, some or all of the space may be allocated in the file, and fill values may be written in the file.</p>

```

<!-- editingComment
<span class="editingComment">[ [ [
  See &lt;&lt;below&gt;&gt;.
] ] ]</span>
-->

```

5.3.3. Data Transfer Operations on a Dataset</h3>

Data is transferred between memory and the raw data array of the dataset through `H5Dwrite` and `H5Dread` operations. A data transfer has the following basic steps:

- Allocate and initialize memory space as needed

- Define the datatype of the memory elements

- Define the elements to be transferred (a selection, or all the elements)

- Set data transfer properties (including parameters for filters or file drivers) as needed

- Call the H5D API

Note that the location of the data in the file, the datatype of the data in the file, the storage properties, and the filters do not need to be specified because these are stored as a permanent part of the dataset. A selection of elements from the dataspace is specified; the selected elements may be the whole dataspace.

<!-- NEW PAGE -->

The figure below shows a diagram of a write operation which transfers a data array from memory to a dataset in the file (usually on disk). A read operation has similar parameters with the data flowing the other direction.

--

--

--

--

```

    
  </td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 3. A write operation</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

<h4>Memory Space</h4>

<p>The calling program must allocate sufficient memory to store the data elements to be transferred. For a write (from memory to the file), the memory must be initialized with the data to be written to the file. For a read, the memory must be large enough to store the elements that will be read. The amount of storage needed can be computed from the memory datatype (which defines the size of each data element) and the number of elements in the selection.</p>

```
<!-- NEW PAGE -->
```

<h4>Memory Datatype</h4>

<p>The memory layout of a single data element is specified by the memory datatype. This specifies the size, alignment, and byte order of the element as well as the datatype class. Note that the memory datatype must be the same datatype class as the file, but may have different byte order and other properties. The HDF5 Library automatically transforms data elements between the source and destination layouts. See the chapter

“HDF5 Datatypes”
for more details.</p>

<p>For a write, the memory datatype defines the layout of the data to be written; an example is IEEE floating-point numbers in native byte order. If the file datatype (defined when the dataset is created) is different but compatible, the HDF5 Library will transform each data element when it is written. For example, if the file byte order is different than the native byte order, the HDF5 Library will swap the bytes.</p>

<p>For a read, the memory datatype defines the desired layout of the data to be read. This must be compatible with the file datatype, but should generally use native formats, e.g., byte orders. The HDF5 Library will transform each data element as it is read.</p>

<h4>Selection</h4>

<p>The data transfer will transfer some or all of the elements of the dataset depending on the dataspace selection. The selection has two dataspace objects: one for the source, and one for the destination. These objects describe which elements of the dataspace to be transferred. Some (partial I/O) or all of the data may be transferred. Partial I/O is defined by defining hyperslabs or lists of elements in a dataspace object.</p>

<p>The dataspace selection for the source defines the indices of the elements to be read or written. The two selections must define the same number of points, but the order and layout may be different. The HDF5 Library automatically selects and distributes the elements according to the selections. It might, for example, perform a scatter-gather or sub-set of the data. </p>

```
<!-- editingComment
```

```
<span class="editingComment">[ [ [
```

```
See [[Selections]].
```

```
] ] ]</span>
```

```
-->
```

```
<h4><em>Data Transfer Properties</em></h4>
```

For some data transfers, additional parameters should be set using the transfer property list. The table below lists the categories of transfer properties. These properties set parameters for the HDF5 Library and may be used to pass parameters for optional filters and file drivers. For example, transfer properties are used to select independent or collective operation when using MPI-I/O.

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<td colspan="2" align="left" valign="bottom">
```

```
<b>Table 3. Categories of transfer properties</b></td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
<td><b>Properties</b></td>
```

```
<td><b>Description</b></td>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>Library parameters</td>
```

```
<td>Internal caches, buffers, B-Trees, etc.</td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```

<tr valign="top">
  <td>Memory management</td>
  <td>Variable-length memory management, data overwrite</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>File driver management</td>
  <td>Parameters for file drivers</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Filter management</td>
  <td>Parameters for filters</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

<h4>Data Transfer Operation (Read or Write)</h4>

<p>The data transfer is done by calling <code>H5Dread</code> or <code>H5Dwrite</code> with the parameters described above. The HDF5 Library constructs the required pipeline, which will scatter-gather, transform datatypes, apply the requested filters, and use the correct file driver.</p>

<p>During the data transfer, the transformations and filters are applied to each element of the data in the required order until all the data is transferred.</p>

```
<!-- editingComment
```

```
<span class="editingComment">[ [ [
<p>[[See Data Transfer Below]]
] ] ]</span>
-->
```

```
<h4><em>Summary</em></h4>
```

```
<p>To perform a data transfer, it is necessary to allocate and initialize
memory, describe the source and destination, set required and optional
transfer properties, and call the H5D API. </p>
```

```
<h4><em>Examples</em></h4>
```

```
<p>The basic procedure to <b>write</b> to a dataset is the following:</p>
```

```
<div>
```

```
Open the dataset.<br>
```

```
Set the dataset dataspace for the write (optional if dataspace is
```

```
<code>H5S_SELECT_ALL</code>).<br>
```

```
Write data.<br>
```

```
Close the datatype, dataspace, and property list (as necessary).<br>
```

```
Close the dataset.<br>
```

```
</div>
```

```
<p>Example 3 below shows example code to write a 4 x 6 array of integers.
```

```
In the example, the data is initialized in the memory array dset_data.
```

```
The dataset has already been created in the file, so it is opened
```

```
with <code>H5Dopen</code>.</p>
```

```
<p>The data is written with <code>H5Dwrite</code>. The arguments are the
dataset identifier, the memory datatype (<code>H5T_NATIVE_INT</code>), the
```

memory and file selections (`H5S_ALL` in this case: the whole array), and the default (empty) property list. The last argument is the data to be transferred.

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

```
hid_t    file_id, dataset_id; /* identifiers */
```

```
herr_t    status;
```

```
int       i, j, dset_data[4][6];
```

```
/* Initialize the dataset. */
```

```
for (i = 0; i < 4; i++)
```

```
    for (j = 0; j < 6; j++)
```

```
        dset_data[i][j] = i * 6 + j + 1;
```

```
/* Open an existing file. */
```

```
file_id = H5Fopen("dset.h5", H5F_ACC_RDWR, H5P_DEFAULT);
```

```
/* Open an existing dataset. */
```

```
dataset_id = H5Dopen(file_id, "/dset", H5P_DEFAULT);
```

```
/* Write the entire dataset, using 'dset_data':
```

```
    memory type is 'native int'
```

```
    write the entire dataspace to the entire dataspace,
```

```
    no transfer properties,
```

```
*/
```

```
status = H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL,
```

```

H5S_ALL, H5P_DEFAULT, dset_data);

status = H5Dclose(dataset_id);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 3. Write an array of integers</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

Example 4 below shows a similar write except for setting a non-default value for the transfer buffer.

```

<!-- editingComment
<span class="editingComment">[ [
  &lt;&lt;explain what this does&gt;&gt;.
  ] ]</span>
-->

```

The code is the same as Example 3, but a transfer property list is created, and the desired buffer size is set. The `H5Dwrite` function has the same arguments, but uses the property list to set the buffer.</p>

```

<!-- NEW PAGE -->
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
    </td>
  </tr>
</table>

```

```

<pre>
hid_t    file_id, dataset_id;
hid_t    xferplist;
herr_t    status;
int      i, j, dset_data[4][6];

file_id = H5Fopen("dset.h5", H5F_ACC_RDWR, H5P_DEFAULT);

dataset_id = H5Dopen(file_id, "/dset", H5P_DEFAULT);

/*
 * Example: set type conversion buffer to 64MB
 */
xferplist = H5Pcreate(H5P_DATASET_XFER);
status = H5Pset_buffer( xferplist, 64 * 1024 *1024, NULL, NULL);

/* Write the entire dataset, using 'dset_data':
   memory type is 'native int'
   write the entire dataspace to the entire dataspace,
   set the buffer size with the property list,
 */
status = H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL,
                  H5S_ALL, xferplist, dset_data);

status = H5Dclose(dataset_id);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 4. Write an array using a property list</b>
    <hr color="green" size="3"/></td>

```

```

    </tr>
</table>
<br />

<!-- editingComment
  <span class="editingComment">[ [ [
    <p>Partial writes (i.e., of selected data elements, such as a hyperslab)
    are explained below [[partial I/O]]
  ] ] ]</span>
-->

```

<p>The basic procedure to read from a dataset is the following:</p>

```

<dir>
  Define the memory dataspace of the read (optional if dataspace is
  <code>H5S_SELECT_ALL</code>).<br>
  Open the dataset.<br>
  Get the dataset dataspace (if using <code>H5S_SELECT_ALL</code> above).<br>

</dir>Else define dataset dataspace of read.</dir>
  Define the memory datatype (optional).<br>
  Define the memory buffer.<br>
  Open the dataset.<br>
  Read data.<br>
  Close the datatype, dataspace, and property list (as necessary).<br>
  Close the dataset.
</dir>

```

<p>The example below shows code that reads a 4 x 6 array of integers from

a dataset called "dset1". First, the dataset is opened.

The `H5Dread` call has parameters:

<!-- NEW PAGE -->

The dataset identifier (from `H5Dopen`)

The memory datatype (`H5T_NATIVE_INT`)

The memory and file dataspace (`H5S_ALL`, the whole array)

A default (empty) property list

The memory to be filled

<table width="600" cellpadding="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

hid_t file_id, dataset_id;

herr_t status;

int i, j, dset_data[4][6];

/* Open an existing file. */

file_id = H5Fopen("dset.h5", H5F_ACC_RDWR, H5P_DEFAULT);

/* Open an existing dataset. */

dataset_id = H5Dopen(file_id, "/dset", H5P_DEFAULT);

/* read the entire dataset, into 'dset_data':

memory type is 'native int'

read the entire dataspace to the entire dataspace,

```

        no transfer properties,
    */
    status = H5Dread(dataset_id, H5T_NATIVE_INT, H5S_ALL,
        H5S_ALL, H5P_DEFAULT, dset_data);

    status = H5Dclose(dataset_id);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 5. Read an array from a dataset</b>
        <hr color="green" size="3" />
    </td>
</tr>
</table>
<br />

```

<h4>5.3.4. Retrieve the Properties of a Dataset</h4>

The functions listed below allow the user to retrieve information regarding a dataset including the datatype, the dataspace, the dataset creation property list, and the total stored size of the data.

| |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <tr valign="bottom"> <td colspan="3" align="left" valign="bottom"> Function Listing 4. Retrieve dataset information </td> </tr> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
<tr valign="top">
  <td>
    <b>Query Function</b></td><td>&nbsp;</td>
  <td>
    <b>Description</b></td></tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td> <code>H5Dget_space</code></td><td>&nbsp;</td>
  <td> Retrieve the dataspace of the dataset
    as stored in the file.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td> <code>H5Dget_type</code></td><td>&nbsp;</td>
  <td> Retrieve the datatype of the dataset
    as stored in the file.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td> <code>H5Dget_create_plist</code></td><td>&nbsp;</td>
  <td> Retrieve the
    dataset creation properties.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td> <code>H5Dget_storage_size</code></td><td>&nbsp;</td>
  <td>
    Retrieve the total bytes for all the data of the dataset.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

```

```

<tr valign="top">
  <td> <code>H5Dvlen_get_buf_size</code></td><td>&nbsp;</td>
  <td>
    Retrieve the total bytes for all the variable-length
    data of the dataset.</td>
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

```

<p>The example below illustrates how to retrieve dataset information.</p>

```

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
hid_t    file_id, dataset_id;
hid_t    dspace_id, dtype_id, plist_id;
herr_t    status;

/* Open an existing file. */
file_id = H5Fopen("dset.h5", H5F_ACC_RDWR, H5P_DEFAULT);

/* Open an existing dataset. */
dataset_id = H5Dopen(file_id, "/dset", H5P_DEFAULT);

dspace_id = H5Dget_space(dataset_id);
dtype_id = H5Dget_type(dataset_id);

```

```

plist_id = H5Dget_create_plist(dataset_id);

/* use the objects to discover the properties of the dataset */

status = H5Dclose(dataset_id);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 6. Retrieve dataset</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

```

<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="DTransfer">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>

```

```

<a name="DTransfer">
<h3 class=pagebefore>5.4. Data Transfer</h3>
</a>

```

The HDF5 Library implements data transfers through a pipeline which implements data transformations (according to the datatype and selections), chunking (as requested), and I/O operations using different mechanisms (file drivers). The pipeline is automatically configured by the HDF5 Library. Metadata is stored in the file so that the correct pipeline can be constructed to retrieve the data. In addition, optional filters such as compression may be added to the standard pipeline.

The figure below illustrates data layouts for different layers of an application using HDF5. The application data is organized as a multidimensional array of elements. The HDF5 format specification

<!-- editingComment

[[[[cite it]]]]]

-->

defines the stored layout of the data and metadata. The storage layout properties define the organization of the abstract data. This data is written and read to and from some storage medium.

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center">

<tr valign="top">

<td align="center">

<hr color="green" size="3"/>

</td>

</tr>

<tr><td><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td align="left" >

Figure 4. Data layouts in an application

<hr color="green" size="3"/></td>

</tr>

</table>

<p>The last stage of a write (and first stage of a read) is managed by an HDF5 file driver module. The virtual file layer of the HDF5 Library implements a standard interface to alternative I/O methods, including memory (AKA “core”) files, single serial file I/O, multiple file I/O, and parallel I/O. The file driver maps a simple abstract HDF5 file to the specific access methods.</p>

<p>The raw data of an HDF5 dataset is conceived to be a multidimensional array of data elements. This array may be stored in the file according to several storage strategies:</p>

Contiguous

Chunked

Compact

<p>The storage strategy does not affect data access methods except that certain operations may be more or less efficient depending on the storage strategy and the access patterns.</p>

<p>Overall, the data transfer operations (<code>H5Dread</code> and

`H5Dwrite`) work identically for any storage method, for any file driver, and for any filters and transformations. The HDF5 Library automatically manages the data transfer process. In some cases, transfer properties should or must be used to pass additional parameters such as MPI/IO directives when used the parallel file driver.

5.4.1. The Data Pipeline

When data is written or read to or from an HDF5 file, the HDF5 Library passes the data through a sequence of processing steps which are known as the HDF5 data pipeline. This data pipeline performs operations on the data in memory such as byte swapping, alignment, scatter-gather, and hyperslab selections. The HDF5 Library automatically determines which operations are needed and manages the organization of memory operations such as extracting selected elements from a data block. The data pipeline modules operate on data buffers: each module processes a buffer and passes the transformed buffer to the next stage.

The table below lists the stages of the data pipeline. The figure below the table shows the order of processing during a read or write.

| | |
|--------------------------------------------------------------------------|--|
| <div> <div></div> <div>Table 4. Stages of the data pipeline</div> </div> | |
| <hr/> | |

| <td width="35%">Layers</td> | <td width="65%">Description</td> |
|-------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <tr><td colspan="2"><hr color="green" size="1" /></td></tr> | |
| <tr valign="top"> | <td>I/O initiation</td>
<td>Initiation of HDF5 I/O activities (<code>H5Dwrite</code> and
<code>H5Dread</code>) in a user's application program. </td>
</tr> |
| <tr><td colspan="2"><hr color="green" size="1" /></td></tr> | |
| <tr valign="top"> | <td>Memory hyperslab operation</td>
<td>Data is scattered to (for read), or gathered from (for write)
the application's memory buffer (bypassed if no datatype
conversion is needed).</td>
</tr> |
| <tr><td colspan="2"><hr color="green" size="1" /></td></tr> | |
| <tr valign="top"> | <td>Datatype conversion</td>
<td>Datatype is converted if it is different between memory and
storage (bypassed if no datatype conversion is needed).</td>
</tr> |
| <tr><td colspan="2"><hr color="green" size="1" /></td></tr> | |
| <tr valign="top"> | <td>File hyperslab operation</td>
<td>Data is gathered from (for read), or scattered to (for write) to
file space in memory (bypassed if no datatype conversion is needed).</td>
</tr> |
| <tr><td colspan="2"><hr color="green" size="1" /></td></tr> | |
| <tr valign="top"> | <td>Filter pipeline</td>
<td>Data is processed by filters when it passes. Data can be |

```

        modified and restored here (bypassed if no datatype conversion
        is needed, no filter is enabled, or dataset is not chunked).</td>
    </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>Virtual File Layer</td>
    <td>Facilitate easy plug-in file drivers such as MPIO or
    POSIX I/O.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>Actual I/O</td>
    <td>Actual file driver used by the library such as MPIO or STDIO.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<!-- NEW PAGE -->

```

```

<table width="600" cellspacing="0" align="center">
    <tr valign="top">
        <td align="center">
            <hr color="green" size="3"/>
            <br>
        </td>
    </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left" >

```

Figure 5. The processing order in the data pipeline

The HDF5 Library automatically applies the stages as needed.

When the memory dataspace selection is other than the whole dataspace, the memory hyperslab stage scatters/gathers the data elements between the application memory (described by the selection) and a contiguous memory buffer for the pipeline. On a write, this is a gather operation; on a read, this is a scatter operation.

When the memory datatype is different from the file datatype, the datatype conversion stage transforms each data element. For example, if data is written from 32-bit big-endian memory, and the file datatype is 32-bit little-endian, the datatype conversion stage will swap the bytes of every elements. Similarly, when data is read from the file to native memory, byte swapping will be applied automatically when needed.

The file hyperslab stage is similar to the memory hyperslab stage, but is managing the arrangement of the elements according to the dataspace selection. When data is read, data elements are gathered from the data blocks from the file to fill the contiguous buffers which are then processed by the pipeline. When data is read, the elements from a buffer are scattered to the data blocks of the file.

5.4.2. Data Pipeline Filters

In addition to the standard pipeline, optional stages, called filters,

can be inserted in the pipeline.

```
<!-- editingComment
```

```
<span class="editingComment">[ [
```

```
see [[chunked]])
```

```
] ] ]</span>
```

```
-->
```

The standard distribution includes optional filters to implement compression and error checking. User applications may add custom filters as well.</p>

<p>The HDF5 Library distribution includes or employs

several optional filters. These are listed in the table below.

The filters are applied in the pipeline between the virtual file layer and the file hyperslab operation. See the figure above. The application can use any number of filters in any order.</p>

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<td colspan="2" align="left" valign="bottom">
```

```
<b>Table 5. Data pipeline filters</b></td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
<td width="35%"><b>Filter</b></td>
```

```
<td width="65%"><b>Description</b></td>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>gzip compression</td>
```

```
<td>Data compression using <code>zlib</code>.</td>
```

```
</tr>
```

```

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Szip compression</td>
  <td>Data compression using the Szip library. See The HDF Group
  website for more information regarding the
  <a href="http://www.hdfgroup.org/doc_resource/SZIP/"
  target="Ext1">Szip</a> filter.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>N-bit compression</td>
  <td>Data compression using an algorithm specialized for
  n-bit datatypes.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Scale-offset compression</td>
  <td>Data compression using using a "scale and
  offset" algorithm.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Shuffling</td>
  <td>To improve compression performance, data is regrouped by
  its byte position in the data unit. In other words, the
  1<sup><font size="-1">st</font></sup>,
  2<sup><font size="-1">nd</font></sup>,
  3<sup><font size="-1">rd</font></sup>, and
  4<sup><font size="-1">th</font></sup> bytes of integers are
  stored together respectively.</td>
</tr>

```

```

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Fletcher32</td>
  <td>Fletcher32 checksum for error-detection.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

Filters may be used only for chunked data and are applied to chunks of data between the file hyperslab stage and the virtual file layer. At this stage in the pipeline, the data is organized as fixed-size blocks of elements, and the filter stage processes each chunk separately.

Filters are selected by dataset creation properties, and some behavior may be controlled by data transfer properties. The library determines what filters must be applied and applies them in the order in which they were set by the application. That is, if an application calls `H5Pset_shuffle` and then `H5Pset_deflate` when creating a dataset's creation property list, the library will apply the shuffle filter first and then the deflate filter.

Information regarding the n-bit and scale-offset filters can be found in the [Using the N-bit Filter](#N-Bit) and [Using the Scale-offset Filter](#ScaleOffset) sections, respectively.

5.4.3. File Drivers

I/O is performed by the HDF5 virtual file layer. The file driver

interface writes and reads blocks of data; each driver module implements the interface using different I/O mechanisms. The table below lists the file drivers currently supported. Note that the I/O mechanisms are separated from the pipeline processing: the pipeline and filter operations are identical no matter what data access mechanism is used.

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<td colspan="2" align="left" valign="bottom">
```

```
<b>Table 6. I/O file drivers</b></td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
<td><b>File Driver</b></td>
```

```
<td><b>Description</b></td>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td><code>H5FD_CORE</code></td>
```

```
<td>Store in memory (optional backing store to disk file).</td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td><code>H5FD_FAMILY</code></td>
```

```
<td>Store in a set of files.</td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td><code>H5FD_LOG</code></td>
```

```
<td>Store in logging file.</td>
```

```
</tr>
```

```

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5FD_MPIO</code></td>
  <td>Store using MPI/IO.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5FD_MULTI</code></td>
  <td>Store in multiple files. There are several options to control
  layout.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5FD_SEC2</code></td>
  <td>Serial I/O to file using Unix &ldquo;section 2&rdquo; functions.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5FD_STDIO</code></td>
  <td>Serial I/O to file using Unix &ldquo;stdio&rdquo; functions.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

Each file driver writes/reads contiguous blocks of bytes from a logically contiguous address space. The file driver is responsible for managing the details of the different physical storage methods.

In serial environments, everything above the virtual file layer tends to work identically no matter what storage method is used.

Some options may have substantially different performance depending on the file driver that is used. In particular, multi-file and parallel I/O may perform considerably differently from serial drivers depending on chunking and other settings.

5.4.4. Data Transfer Properties to Manage the Pipeline

Data transfer properties set optional parameters that control parts of the data pipeline. The function listing below shows transfer properties that control the behavior of the library.

!-- editingComment

```
<span class="editingComment">[ [
&lt;&lt;Developers: explain what these do!&gt;&gt;
] ]</span>
```

-->

!-- NEW PAGE -->

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<td colspan="3" align="left" valign="bottom">
```

```
<b>Function Listing 5. Data transfer property list functions
```

```
</b></td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<b>Property</b></td><td>&nbsp;</td>
```

```
<td>
```

```
<b>Description</b></td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```

<tr valign="top">
  <td> <code>H5Pset_buffer</code></td><td>&nbsp;</td>
  <td>
    Maximum size for the type conversion buffer and the background
    buffer. May also
    supply pointers to application-allocated buffers.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td> <code>H5Pset_hyper_cache</code></td><td>&nbsp;</td>
  <td>
    Whether to cache hyperslab blocks during I/O.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td> <code>H5Pset_btree_ratios</code></td><td>&nbsp;</td>
  <td>
    Set the B-tree split ratios for a dataset transfer property list.
    The split ratios determine what percent of children go in the
    first node when a node splits.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

Some filters and file drivers require or use additional parameters from the application program. These can be passed in the data transfer property list. The table below shows file driver property list functions.

```

<table width="600" cellspacing="0" align="center" cellpadding="0">

```

```

<tr valign="bottom">
  <td colspan="3" align="left" valign="bottom">
    <b>Function Listing 6. File driver property list functions
  </b></td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
<tr valign="top">
  <td>
    <b>Property</b></td><td>&nbsp;</td>
  <td>
    <b>Description</b></td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5Pset_dxpl_mpio</code></td><td>&nbsp;</td>
  <td>
    Control the MPI I/O transfer mode (independent or collective)
    during data I/O operations.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5Pset_small_data_block_size</code></td><td>&nbsp;</td>
  <td>
    Reserves blocks of size bytes for the contiguous storage of the raw
    data portion of small datasets. The HDF5 Library then writes the raw data
    from small datasets to this reserved space which reduces unnecessary
    discontinuities within blocks of metadata and
    improves I/O performance.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">

```

```

<td> <code>H5Pset_edc_check</code></td><td>&nbsp;</td>
<td>
Disable/enable EDC checking for read. When selected, EDC
is always written.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

The transfer properties are set in a property list which is passed as a parameter of the `H5Dread` or `H5Dwrite` call. The transfer properties are passed to each pipeline stage. Each stage may use or ignore any property in the list. In short, there is one property list that contains all the properties.

5.4.5. Storage Strategies

The raw data is conceptually a multi-dimensional array of elements that is stored as a contiguous array of bytes. The data may be physically stored in the file in several ways. The table below lists the storage strategies for a dataset.

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="2" align="left" valign="bottom">
      <b>Table 7. Dataset storage strategies</b></td>
    </tr>
    <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
    <tr valign="top">

```

```

    <td><b>Storage Strategy</b></td>
    <td><b>Description</b></td>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>Contiguous</td>
    <td>The dataset is stored as one continuous array of bytes.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>Chunked</td>
    <td>The dataset is stored as fixed-size chunks.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>Compact</td>
    <td>A small dataset is stored in the metadata header.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

The different storage strategies do not affect the data transfer operations of the dataset: reads and writes work the same for any storage strategy.

```

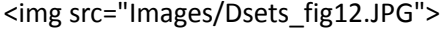
<!-- editingComment
<span class="editingComment">[ [
<p>&lt;&lt;Relationship between storage strategies, and pipeline, filters,
and file drivers.??&gt;&gt;
] ]</span>
-->

```

These strategies are described in the following sections.

Contiguous

A contiguous dataset is stored in the file as a header and a single continuous array of bytes. See the figure below. In the case of a multi-dimensional array, the data is serialized in row major order. By default, data is stored contiguously.

<hr/> 
<hr/>
Figure 6. Contiguous data storage <hr/>

Contiguous storage is the simplest model. It has several limitations. First, the dataset must be a fixed-size: it is not possible to extend the limit of the dataset or to have unlimited dimensions. In other words, if the number of dimensions of the array might change over

time, then chunking storage must be used instead of contiguous.

Second, because data is passed through the pipeline as fixed-size blocks, compression and other filters cannot be used with contiguous data.

<!-- NEW PAGE -->

<h4>Chunked</h4>

<p>The data of a dataset may be stored as fixed-size chunks. See the figure below.

A chunk is a hyper-rectangle of any shape.

When a dataset is chunked, each chunk is read or written as a single I/O operation, and individually passed from stage to stage of the data pipeline.

</p>

--

| |
 |

| --- |
| |
 |

Figure 7. Chunked data storage

Chunks may be any size and shape that fits in the dataspace of the dataset. For example, a three dimensional dataspace can be chunked as 3-D cubes, 2-D planes, or 1-D lines. The chunks may extend beyond the size of the dataspace. For example, a 3 x 3 dataset might be chunked in 2 x 2 chunks. Sufficient chunks will be allocated to store the array, and any extra space will not be accessible. So, to store the 3 x 3 array, four 2 x 2 chunks would be allocated with 5 unused elements stored.

Chunked datasets can be unlimited in any direction and can be compressed or filtered.

Since the data is read or written by chunks, chunking can have a dramatic effect on performance by optimizing what is read and written. Note, too, that for specific access patterns such as parallel I/O, decomposition into chunks can have a large impact on performance.

Two restrictions have been placed on chunk shape and size:

- The rank of a chunk must be less than or equal to the rank of the dataset
- Chunk size cannot exceed the size of a fixed-size dataset; for example, a dataset consisting of a 5 x 4 fixed-size array cannot be defined with 10 x 10 chunks

<!-- NEW PAGE -->

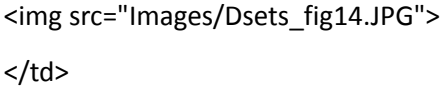
<h4>Compact</h4>

For contiguous and chunked storage, the dataset header information and data are stored in two (or more) blocks. Therefore, at least two I/O operations are required to access the data: one to access the header, and one (or more) to access data. For a small dataset, this is considerable overhead.

A small dataset may be stored in a continuous array of bytes in the header block using the compact storage option. This dataset can be read entirely in one operation which retrieves the header and data.

The dataset must fit in the header. This may vary depending on the metadata that is stored. In general, a compact dataset should be approximately 30 KB or less total size.

See the figure below.

<hr/> 
<hr/>
Figure 8. Compact data storage <hr/>

5.4.6. Partial I/O Sub-setting and Hyperslabs

Data transfers can write or read some of the data elements of the dataset.

This is controlled by specifying two selections: one for the source and one for the destination. Selections are specified by creating a dataspace with selections.

```

<!-- editingComment
<span class="editingComment">[ [ [
(see [[dataspace chapter]])
] ] ]</span>
-->

```

Selections may be a union of hyperslabs or a list of points.

A hyperslab is a contiguous hyper-rectangle from the dataspace.

Selected fields of a compound datatype may be read or written.

In this case, the selection is controlled by the memory and file datatypes.

Summary of procedure:

- Open the dataset
- Define the memory datatype
- Define the memory dataspace selection and file dataspace selection
- Transfer data (`H5Dread` or `H5Dwrite`)

For a detailed explanation of selections, see the chapter

“[HDF5 Dataspaces and Partial I/O](12_Dataspaces.html)”.

”</p>

```
<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="Allocation">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>
```

```
<br />
<!-- NEW PAGE -->
<a name="Allocation">
<h3 class=pagebefore>5.5. Allocation of Space in the File</h3>
</a>
```

<p>When a dataset is created, space is allocated in the file for its header and initial data. The amount of space allocated when the dataset is created depends on the storage properties. When the dataset is modified (data is written, attributes added, or other changes),

additional storage may be allocated if necessary.</p>

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<td colspan="2" align="left" valign="bottom">
```

```
<b>Table 8. Initial dataset size</b></td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
<td width="15%"><b>Object</b></td>
```

```
<td width="85%"><b>Size</b></td>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>Header</td>
```

```
<td>Variable, but typically around 256 bytes at the creation of  
a simple dataset with a simple datatype.</td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>Data</td>
```

```
<td>Size of the data array (number of elements x size of element).
```

```
Space allocated in the file depends on the storage strategy  
and the allocation strategy.</td>
```

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
```

```
</table>
```

```
<br />
```

```
<h4><em>Header</em></h4>
```

<p>A dataset header consists of one or more header messages containing

persistent metadata describing various aspects of the dataset.

These records are defined in the [HDF5 File Format Specification](http://hdf5format.html). The amount of storage required for the metadata depends on the metadata to be stored. The table below summarizes the metadata.

Table 9. Metadata storage sizes	
Header Information	Approximate Storage Size
Datatype (required)	Bytes or more. Depends on type.
Dataspace (required)	Bytes or more. Depends on number of dimensions and hsize_t.
Layout (required)	Points to the stored data. Bytes or more. Depends on hsize_t and number of dimensions.

```

    </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>Filters</td>
    <td>Depends on the number of filters. The size of the filter message
    depends on the name and data that will be passed.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

The header blocks also store the name and values of attributes, so the total storage depends on the number and size of the attributes.

In addition, the dataset must have at least one link, including a name, which is stored in the file and in the group it is linked from.

The different storage strategies determine when and how much space is allocated for the data array. See the discussion of fill values below

```
<!-- editingComment
```

```

<span class="editingComment">[ [ [
Link
] ] ]</span>
-->

```

for a detailed explanation of the storage allocation.

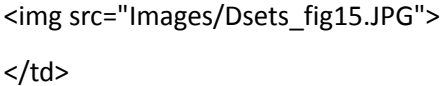
```
<!-- NEW PAGE -->
```

```
<h4><em>Contiguous Storage</em></h4>
```

For a continuous storage option, the data is stored in a single, contiguous block in the file. The data is nominally a fixed-size,

(number of elements x size of element). The figure below shows an example of a two dimensional array stored as a contiguous dataset.

Depending on the fill value properties, the space may be allocated when the dataset is created or when first written (default), and filled with fill values if specified. For parallel I/O, by default the space is allocated when the dataset is created.

<div style="text-align: center;"> <hr style="border: 1px solid green;"/>  </div>	<hr style="border: 1px solid green;"/>
<div> Figure 9. A two dimensional array stored as a contiguous dataset <hr style="border: 1px solid green;"/> </div>	

Chunked

For chunked storage, the data is stored in one or more chunks. Each chunk is a continuous block in the file, but chunks are not necessarily stored contiguously. Each chunk has the same size. The data array has the same nominal size as a contiguous array (number of elements x size of element),

but the storage is allocated in chunks, so the total size in the file can be larger than the nominal size of the array. See the figure below.

If a fill value is defined, each chunk will be filled with the fill value. Chunks must be allocated when data is written, but they may be allocated when the file is created, as the file expands, or when data is written.

For serial I/O, by default chunks are allocated incrementally, as data is written to the chunk. For a sparse dataset, chunks are allocated only for the parts of the dataset that are written. In this case, if the dataset is extended, no storage is allocated.

For parallel I/O, by default chunks are allocated when the dataset is created or extended with fill values written to the chunk.

In either case, the default can be changed using fill value properties. For example, using serial I/O, the properties can select to allocate chunks when the dataset is created.

```
<table width="400" cellpadding="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="center">
```

```
      <hr color="green" size="3"/>
```

```
      
```

```
    </td>
```

```
  </tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
  <td align="left" >
```


Figure 10. A two dimensional array stored in chunks

Changing Dataset Dimensions

`H5Dset_extent` is used to change the current dimensions of the dataset within the limits of the dataspace. Each dimension can be extended up to its maximum or unlimited. Extending the dataspace may or may not allocate space in the file and may or may not write fill values, if they are defined. See the example code below.

The dimensions of the dataset can also be reduced. If the sizes specified are smaller than the dataset's current dimension sizes, `H5Dset_extent` will reduce the dataset's dimension sizes to the specified values. It is the user's responsibility to ensure that valuable data is not lost; `H5Dset_extent` does not check.

<pre> hid_t file_id, dataset_id; Herr_t status; size_t newdims[2]; </pre>

```

/* Open an existing file. */
file_id = H5Fopen("dset.h5", H5F_ACC_RDWR, H5P_DEFAULT);

/* Open an existing dataset. */
dataset_id = H5Dopen(file_id, "/dset", H5P_DEFAULT);

/* Example: dataset is 2 x 3, each dimension is UNLIMITED */
/* extend to 2 x 7 */
newdims[0] = 2;
newdims[1] = 7;

status = H5Dset_extent(dataset_id, newdims);

/* dataset is now 2 x 7 */

```

```
status = H5Dclose(dataset_id);</pre></td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 7. Using <code>H5Dset_extent</code>
```

```
to increase the size of a dataset</b>
```

```
<hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

5.5.1. Storage Allocation in the File: Early, Incremental, Late

The HDF5 Library implements several strategies for when storage is allocated if and when it is filled with fill values for elements not

yet written by the user. Different strategies are recommended for different storage layouts and file drivers. In particular, a parallel program needs storage allocated during a collective call (for example, create or extend) while serial programs may benefit from delaying the allocation until the data is written.

Two file creation properties control when to allocate space, when to write the fill value, and the actual fill value to write.

When to Allocate Space

The table below shows the options for when data is allocated in the file. “Early” allocation is done during the dataset create call. Certain file drivers (especially MPI-I/O and MPI-POSIX) require space to be allocated when a dataset is created, so all processors will have the correct view of the data.

Table 10. File storage allocation options	
Strategy	Description
Early	

```

        <td>Allocate storage for the dataset immediately when the dataset
        is created. </td>
    </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>Late</td>
    <td>Defer allocating space for storing the dataset until the
    dataset is written.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>Incremental</td>
    <td>Defer allocating space for storing each chunk until
    the chunk is written.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>Default</td>
    <td>Use the strategy (Early, Late, or Incremental) for the storage method
    and access method. This is the recommended strategy.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

“Late” allocation is done at the time of the first write to dataset. Space for the whole dataset is allocated at the first write.</p>

“Incremental” allocation (chunks only) is done at the time of the first write to the chunk. Chunks that have never been written are not allocated in the file. In a sparsely populated dataset, this option

allocates chunks only where data is actually written.

The “Default” property selects the option recommended as appropriate for the storage method and access method. The defaults are shown in the table below. Note that “Early” allocation is recommended for all Parallel I/O, while other options are recommended as the default for serial I/O cases.

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="3" align="left" valign="bottom">

Table 11. Default storage options</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td width="33%">

 </td>

<td width="34%">

Serial I/O</td>

<td width="33%">

Parallel I/O</td>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>Contiguous Storage</td>

<td>Late</td>

<td>Early</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

```

        <td>Chunked Storage</td>
        <td>Incremental</td>
        <td>Early</td>
    </tr>
    <tr><td colspan="3"><hr color="green" size="1" /></td></tr>
    <tr valign="top">
        <td>Compact Storage</td>
        <td>Early</td>
        <td>Early</td>
    </tr>
    <tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

<h4>When to Write the Fill Value</h4>

<p>The second property is when to write the fill value. The possible values are “Never” and “Allocation”.

The table below shows these options.</p>

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
    <tr valign="bottom">
        <td colspan="2" align="left" valign="bottom">
            <b>Table 12. When to write fill values</b></td>
        </tr>
    <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
    <tr valign="top">
        <td width="15%"><b>When</b></td>
        <td width="85%"><b>Description</b></td>
    <tr><td colspan="2"><hr color="green" size="1" /></td></tr>

```

```

<tr valign="top">
  <td>Never</td>
  <td>Fill value will never be written.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Allocation</td>
  <td>Fill value is written when space is allocated. (Default for
    chunked and contiguous data storage.)</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

<h4>Fill Values</h4>

<p>The third property is the fill value to write. The table below shows the values. By default, the data is filled with zeroes. The application may choose no fill value (Undefined). In this case, uninitialized data may have random values. The application may define a fill value of an appropriate type. See the chapter “HDF5 Datatypes” for more information regarding fill values.</p>

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="2" align="left" valign="bottom">
      <b>Table 13. Fill values</b></td>
    </tr>
    <tr><td colspan="2"><hr color="green" size="3" /></td></tr>

```

```

<tr valign="top">
  <td><b>What to Write</b></td>
  <td><b>Description</b></td>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Default</td>
  <td>By default, the library fills allocated space with zeroes.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Undefined</td>
  <td>Allocated space is filled with random values.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>User-defined</td>
  <td>The application specifies the fill value.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

Together these three properties control the library's behavior. The table below summarizes the possibilities during the dataset create-write-close cycle.

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="4" align="left" valign="bottom">
      <b>Table 14. Storage allocation and fill summary</b></td>

```



```

</tr>
<tr><td colspan="4"><hr color="green" size="3" /></td></tr>
<tr valign="top">
  <td width="17%">
    <b>When to<br />allocate<br />space</b></td>
    <td width="17%">
      <b>When to<br />write fill<br />value</b></td>
    <td width="17%">
      <b>What fill<br />value to<br />write</b></td>
    <td width="49%">
      <b>Library create-write-close behavior</b></td>
</tr><td colspan="4"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Early</td>
  <td>Never</td>
  <td>-</td>
  <td>Library allocates space when dataset is created, but never
    writes a fill value to dataset. A read of unwritten data returns
    undefined values.</td>
</tr>
<tr><td colspan="4"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Late</td>
  <td>Never</td>
  <td>-</td>
  <td>Library allocates space when dataset is written to, but never
    writes a fill value to the dataset. A read of unwritten data
    returns undefined values.</td>
</tr>
<tr><td colspan="4"><hr color="green" size="1" /></td></tr>
<tr valign="top">

```

Incremental	Library allocates space when a dataset or chunk (whichever is the smallest unit of space) is written to, but it never writes a fill value to a dataset or a chunk. A read of unwritten data returns undefined values.
Never	
<hr/>	
Allocation	Undefined
Error on creating the dataset. The dataset is not created.	
<hr/>	
Early	Allocation
Default or User-defined	Allocate space for the dataset when the dataset is created. Write the fill value (default or user-defined) to the entire dataset when the dataset is created.
<hr/>	
Late	Allocation
Default or User-defined	Allocate space for the dataset when the application first

```

        writes data values to the dataset. Write the fill value to the
        entire dataset before writing application data values.</td>
    </tr>
<tr><td colspan="4"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>Incremental</td>
    <td>Allocation</td>
    <td>Default or User-defined</td>
    <td>Allocate space for the dataset when the application first
        writes data values to the dataset or chunk (whichever is the
        smallest unit of space). Write the fill value to the entire dataset
        or chunk before writing application data values. </td>
</tr>
<tr><td colspan="4"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

```

<p>During the <code>H5Dread</code> function call, the library behavior
depends on whether space has been allocated, whether the fill value has
been written to storage, how the fill value is defined, and when to
write the fill value. The table below summarizes the different behaviors.</p>

```

```

<table width="600" cellpadding="0" cellspacing="0" align="center">
    <tr valign="bottom">
        <td colspan="4" align="left" valign="bottom">

```

```

<b>Table 15. <code>H5Dread</code> summary</b></td>
</tr>
<tr><td colspan="4"><hr color="green" size="3" /></td></tr>
<tr valign="top">
  <td width="17%">
    <b>Is space<br />allocated<br />in the file?</b></td>
    <td width="17%">
      <b>What is the<br />fill value?</b></td>
    <td width="17%">
      <b>When to<br />write the<br />fill value?</b></td>
    <td width="49%">
      <b>Library read behavior</b></td>
</tr>
<tr><td colspan="4"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>No</td>
  <td>Undefined</td>
  <td>&lt;&lt;any&gt;&gt;</td>
  <td><b>Error</b>. Cannot create this dataset.</td>
</tr>
<tr><td colspan="4"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>No</td>
  <td>Default or User-defined</td>
  <td>&lt;&lt;any&gt;&gt;</td>
  <td>Fill the memory buffer with the fill value.</td>
</tr>
<tr><td colspan="4"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Yes</td>
  <td>Undefined</td>
  <td>&lt;&lt;any&gt;&gt;</td>

```

Return data from storage (dataset). Trash is possible if the application has not written data to the portion of the dataset being read.			
<hr/>			
Yes	Default or User-defined	Never	Return data from storage (dataset). Trash is possible if the application has not written data to the portion of the dataset being read.
<hr/>			
Yes	Default or User-defined	Allocation	Return data from storage (dataset).
<hr/>			

There are two cases to consider depending on whether the space in the file has been allocated before the read or not. When space has not yet been allocated and if a fill value is defined, the memory buffer will be filled with the fill values and returned. In other words, no data has been read from the disk. If space has been allocated, the values are returned from the stored data. The unwritten elements will be filled according to the fill value.

5.5.2. Deleting a Dataset from a File and Reclaiming Space

HDF5 does not at this time provide an easy mechanism to remove a dataset from a file or to reclaim the storage space occupied by a deleted object.

Removing a dataset and reclaiming the space it used can be done with the `H5Ldelete` function and the

[h5repack](../RM/Tools.html#Tools-Repack) utility program.

With the `H5Ldelete` function, links to a dataset can be removed from the file structure. After all the links have been removed, the dataset becomes inaccessible to any application and is effectively removed from the file. The way to recover the space occupied by an unlinked dataset is to write all of the objects of the file into a new file. Any unlinked object is inaccessible to the application and will not be included in the new file. Writing objects to a new file can be done with a custom program or with the `h5repack` utility program.

8.11.10, MEE: in the paragraph below, the link should be changed.

Links are now done separately from groups, but there is no HDF5 Links chapter yet. -->

See the chapter "HDF5 Groups" for further discussion of HDF5 file structures and the use of links.

5.5.3. Releasing Memory Resources

The system resources required for HDF5 objects such as datasets, datatypes, and dataspace should be released once access to the object is no longer needed. This is accomplished via the appropriate close function. This is not unique to datasets but a general requirement when working

with the HDF5 Library; failure to close objects will result in resource leaks. </p>

<p>In the case where a dataset is created or data has been transferred,
there are several objects that must be closed. These objects

<!-- editingComment

[[[

(T? above)

originally appeared here. On the full editorial pass,
see if there is any apparent reason for the question.

]]]

-->

include datasets,

datatypes, dataspace, and property lists. </p>

<p>The application program must free any memory variables and buffers it
allocates. When accessing data from the file, the amount of memory required
can be determined by calculating the size of the memory datatype and the
number of elements in the memory selection.</p>

<p>Variable-length data are organized in two or more areas of memory.
See “HDF5 Datatypes” for
more information. When writing data, the application creates an array
of <code>vl_info_t</code> which contains pointers to the elements.
The elements might be, for example, strings. In the file, the
variable-length data is stored in two parts: a heap with the
variable-length values of the data elements and an array of
<code>vlinfo_t</code> elements. When the data is read, the amount of
memory required for the heap can be determined with the
<code>H5Dget_vlen_buf_size</code> call.</p>

The data transfer property may be used to set a custom memory manager for allocating variable-length data for a `H5Dread`. This is set with the `H5Pset_vlen_mem_manager` call.

To free the memory for variable-length data, it is necessary to visit each element, free the variable-length data, and reset the element. The application must free the memory it has allocated. For memory allocated by the HDF5 Library during a read, the `H5Dvlen_reclaim` function can be used to perform this operation.

5.5.4. External Storage Properties

The external storage format allows data to be stored across a set of non-HDF5 files. A set of segments (offsets and sizes) in one or more files is defined as an external file list, or EFL, and the contiguous logical addresses of the data storage are mapped onto these segments. Currently, only the `H5D_CONTIGUOUS` storage format allows external storage. External storage is enabled by a dataset creation property. The table below shows the API.

<!-- NEW PAGE -->

--	--

```

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td width="50%">
    <code>herr_t H5Pset_external (hid_t plist, const char *name,
    off_t offset, hsize_t size)</code></td>
    <td width="50%">This function adds a new segment to the end of
    the external file list of the specified dataset creation property
    list. The segment begins a byte offset of file name and continues
    for size bytes. The space represented by this segment is adjacent
    to the space already represented by the external file list. The
    last segment in a file list may have the size
    <code>H5F_UNLIMITED</code>, in which case the external file may
    be of unlimited size and no more files can be added to the
    external files list.</td>
  </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>int H5Pget_external_count (hid_t plist)</code></td>
  <td>Calling this function returns the number of segments in an
  external file list. If the dataset creation property list has no
  external data, then zero is returned.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>herr_t H5Pget_external (hid_t plist, int idx, size_t
  name_size,<br />char *name, off_t *offset,<br />hsize_t *size)</code></td>
  <td>This is the counterpart for the <code>H5Pset_external()</code>
  function. Given a dataset creation property list and a zero-based
  index into that list, the file name, byte offset, and segment
  size are returned through non-null arguments. At most name_size
  characters are copied into the name argument which is not null

```

terminated if the file name is longer than the supplied name

buffer (this is similar to `strncpy()`). `</td>`

`</tr>`

`<tr><td colspan="2"><hr color="green" size="3" /></td></tr>`

`</table>`

`
`

The figure below shows an example of how a contiguous, one-dimensional dataset is partitioned into three parts and each of those parts is stored in a segment of an external file. The top rectangle represents the logical address space of the dataset while the bottom rectangle represents an external file.

`<table width="500" cellpadding="0" align="center">`

`<tr valign="top">`

`<td align="center">`

`<hr color="green" size="3"/>`

``

`</td>`

`</tr>`

`<tr><td><hr color="green" size="1" /></td></tr>`

`<tr valign="top">`

`<td align="left" >`

`Figure 11. External file storage`

`<hr color="green" size="3"/></td>`

`</tr>`

`</table>`

`
`

The example below shows code that defines the external storage for the example. Note that the segments are defined in order of the logical addresses they represent, not their order within the external file. It would also have been possible to put the segments in separate files. Care should be taken when setting up segments in a single file since the library does not automatically check for segments that overlap.

<div style="border-bottom: 3px solid green; margin-bottom: 10px;"></div> <pre> Plist = H5Pcreate (H5P_DATASET_CREATE); H5Pset_external (plist, "velocity.data", 3000, 1000); H5Pset_external (plist, "velocity.data", 0, 2500); H5Pset_external (plist, "velocity.data", 4500, 1500); </pre>
<div style="border-bottom: 1px solid green; margin-bottom: 10px;"></div> <div> Example 8. External storage </div> <div style="border-bottom: 3px solid green; margin-bottom: 10px;"></div>

The figure below shows an example of how a contiguous, two-dimensional dataset is partitioned into three parts and each of those parts is stored in a separate external file. The top rectangle represents the

logical address space of the dataset while the bottom rectangles represent external files.

```
<table width="500" cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 12. Partitioning a 2-D dataset for external storage</b>
      <hr color="green" size="3"/>
    </td>
  </tr>
</table>
<br />
```

The example below shows code for the partitioning described above.

In this example, the library maps the multi-dimensional array onto a linear address space as defined by the HDF5 format specification, and then maps that address space into the segments defined in the external file list.

```
<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="left">
```

```

    <hr color="green" size="3"/>
    <pre>
Plist = H5Pcreate (H5P_DATASET_CREATE);
H5Pset_external (plist, "scan1.data", 0, 24);
H5Pset_external (plist, "scan2.data", 0, 24);
H5Pset_external (plist, "scan3.data", 0, 16);</pre></td>
    </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 9. Partitioning a 2-D dataset for external storage</b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

The segments of an external file can exist beyond the end of the (external) file. The library reads that part of a segment as zeros. When writing to a segment that exists beyond the end of a file, the external file is automatically extended. Using this feature, one can create a segment (or set of segments) which is larger than the current size of the dataset. This allows the dataset to be extended at a future time (provided the dataspace also allows the extension).

All referenced external data files must exist before performing raw data I/O on the dataset. This is normally not a problem since those files are being managed directly by the application or indirectly through some other library. However, if the file is transferred from its original context, care must be taken to assure that all the external files are accessible in the new location.

<!-- NEW PAGE -->

<h3 class=pagebefore>5.6. Using HDF5 Filters</h3>

<p>This section describes in detail how to use the n-bit and scale-offset filters. </p>

<h3>5.6.1. Using the N-bit Filter</h3>

<p>N-bit data has <i>n</i> significant bits,
where <i>n</i> may not correspond to a precise number of bytes.
On the other hand, computing systems and applications universally,
or nearly so, run most efficiently when manipulating data as
whole bytes or multiple bytes.</p>

<p>Consider the case of 12-bit integer data.
In memory, that data will be handled in at least 2 bytes, or 16 bits,
and on some platforms in 4 or even 8 bytes.
The size of such a dataset can be significantly reduced when written
to disk if the unused bits are stripped out.</p>

<p>The <i>n-bit filter</i> is provided for this purpose,
<i>packing</i> n-bit data on output by stripping off all unused bits
and <i>unpacking</i> on input, restoring the extra bits required
by the computational processor.</p>

<h4>N-bit Datatype</h4>

An *n*-bit datatype is a datatype of *n* significant bits.

Unless it is packed, an *n*-bit datatype is presented as an *n*-bit bitfield within a larger-sized value.

For example, a 12-bit datatype might be presented as a 12-bit field in a 16-bit, or 2-byte, value.

Currently, the datatype classes of *n*-bit datatype or *n*-bit field of a compound datatype or an array datatype are limited to integer or floating-point.

The HDF5 user can create an *n*-bit datatype through a series of function calls.

For example, the following calls create a 16-bit datatype that is stored in a 32-bit value with a 4-bit offset:

```
hid_t nbit_datatype = H5Tcopy(H5T_STD_I32LE);
H5Tset_precision(nbit_datatype, 16);
H5Tset_offset(nbit_datatype, 4);
```

In memory, one value of the above example *n*-bit datatype would be stored on a little-endian machine as follows:

byte 3	byte 2	byte 1
--------	--------	--------

```

        <td width="25%" align="center">byte 0</td>
    </tr>
    <tr>
        <td width="25%" align="center"><code>??????</code></td>
        <td width="25%" align="center"><code>???SPPP</code></td>
        <td width="25%" align="center"><code>PPPPPPPP</code></td>
        <td width="25%" align="center"><code>PPPP????</code></td>
    </tr>
</table>
<table width="80%" border="0" align="center">
    <tr>
        <td colspan="4"><font size="-1">
            Key:
            <code>S</code> - sign bit,
            <code>P</code> - significant bit,
            <code>?</code> - padding bit
        <br>
        Sign bit is included in signed integer datatype precision.
    </font></td>
    </tr>
</table>
<br>

</dt>
</dl>
<br />

<!-- NEW PAGE -->
<h4><em>N-bit Filter</em></h4>

```

<p>When data of an n-bit datatype is stored on disk using the

n-bit filter, the filter *packs* the data by stripping off the padding bits; only the significant bits are retained and stored.

The values on disk will appear as follows:

```
<dl>
  <dt>
    <table border="1" width="80%" align="center">
      <tr>
        <td width="45%" align="center">1st value</td>
        <td width="45%" align="center">2nd value</td>
        <td width="10%" align="center">&nbsp;</td>
      </tr>
      <tr>
        <td width="45%" align="center"><code>SPPPPPPP&nbsp;PPPPPPPP</code></td>
        <td width="45%" align="center"><code>SPPPPPPP&nbsp;PPPPPPPP</code></td>
        <td width="10%" align="center">...</td>
      </tr>
    </table>
    <table width="80%" border="0" align="center">
      <tr>
        <td colspan="4"><font size="-1">
          Key:
            <code>S</code> - sign bit,
            <code>P</code> - significant bit,
            <code>?</code> - padding bit
          <br>
          Sign bit is included in signed integer datatype precision.
        </font></td>
      </tr>
    </table>
  </dt>
```

</dl>

The n-bit filter can be used effectively for compressing data of an n-bit datatype, including arrays and the n-bit fields of compound datatypes. The filter supports complex situations where a compound datatype contains member(s) of a compound datatype or an array datatype has a compound datatype as the base type.

At present, the n-bit filter supports all datatypes.

For datatypes of class time, string, opaque, reference, ENUM, and variable-length, the n-bit filter acts as a no-op which is short for no operation.

For convenience, the rest of this section refers to such datatypes as *no-op datatypes*.

As is the case with all HDF5 filters, an application using the n-bit filter must store data with chunked storage.

How Does the N-bit Filter Work?

The n-bit filter always compresses and decompresses according to dataset properties supplied by the HDF5 Library in the datatype, dataspace, or dataset creation property list.

The dataset datatype refers to how data is stored in an HDF5 file while the memory datatype refers to how data is stored in memory. The HDF5 Library will do datatype conversion when writing data in memory to the dataset or reading data from the dataset to memory if

the memory datatype differs from the dataset datatype.

Datatype conversion is performed by HDF5 Library before n-bit compression and after n-bit decompression.

The following sub-sections examine the common cases:

- N-bit integer conversions

- N-bit floating-point conversions

N-bit Integer Conversions

Integer data with a dataset of integer datatype of less than full precision and a memory datatype of `H5T_NATIVE_INT`, provides the simplest application of the n-bit filter.

The precision of `H5T_NATIVE_INT` is 8 multiplied by `sizeof(int)`.

This value, the size of an `int` in bytes, differs from platform to platform; we assume a value of `4` for the following illustration.

We further assume the memory byte order to be little-endian.

<!-- NEW PAGE -->

In memory, therefore, the precision of `H5T_NATIVE_INT` is 32 and the offset is 0.

One value of `H5T_NATIVE_INT` is laid out in memory as follows:

||
||
||

```

<td>
<pre>
| byte 3 | byte 2 | byte 1 | byte 0 |

| SPPPPPPP | PPPPPPPP | PPPPPPPP | PPPPPPPP |

</pre>
</td>
</tr>
<tr>
<td colspan="4"><font size="-1">
  Key:
  <code>S</code> - sign bit,
  <code>P</code> - significant bit,
  <code>?</code> - padding bit
  <br>
  Sign bit is included in signed integer datatype precision.
  </font></td>
</tr>
</table>

```

Suppose the dataset datatype has a precision of 16 and an offset of 4.

After HDF5 converts values from the memory datatype to the dataset datatype, it passes something like the following to the n-bit filter for compression:

```

<table border="0" width="80%" align="center">
<tr>
<td>
<pre>
| byte 3 | byte 2 | byte 1 | byte 0 |

```

```

      |      |
| ????????|????S|PPP|PPPPPPPP|PPPP|????|
      |_____|
      truncated bits

```

</pre>

</td>

</tr>

<tr>

<td colspan="4">

Key:

<code>S</code> - sign bit,

<code>P</code> - significant bit,

<code>?</code> - padding bit

Sign bit is included in signed integer datatype precision.

</td>

</tr>

</table>

Notice that only the specified 16 bits (15 significant bits and the sign bit) are retained in the conversion. All other significant bits of the memory datatype are discarded because the dataset datatype calls for only 16 bits of precision.

After n-bit compression, none of these discarded bits, known as *padding bits* will be stored on disk.

N-bit Floating-point Conversions

Things get more complicated in the case of a floating-point dataset datatype class. This sub-section provides an example that illustrates the conversion from a memory datatype of

`H5T_NATIVE_FLOAT` to a dataset datatype of class floating-point.

As before, let the `H5T_NATIVE_FLOAT` be 4 bytes long, and let the memory byte order be little-endian.

Per the IEEE standard, one value of `H5T_NATIVE_FLOAT` is laid out in memory as follows:

```
<table border="0" width="80%" align="center">
```

```
<tr>
```

```
<td>
```

```
<pre>
```

```
| byte 3 | byte 2 | byte 1 | byte 0 |
```

```
|SEEEEEEE|EMMMMMMMM|MMMMMMMMMM|MMMMMMMMMM|
```

```
</pre>
```

```
</td></tr>
```

```
<tr>
```

```
<td colspan="4"><font size="-1">
```

```
Key:
```

```
<code>S</code> - sign bit,
```

```
<code>E</code> - exponent bit,
```

```
<code>M</code> - mantissa bit,
```

```
<code>?</code> - padding bit
```

```
<br>
```

```
Sign bit is included in floating-point datatype precision.
```

```
</font></td>
```

```
</tr>
```

```
</table>
```


<p>Suppose the dataset datatype has a precision of 20, offset of 7, mantissa size of 13, mantissa position of 7, exponent size of 6, exponent position of 20, and sign position of 26.

(See “Definition of Datatypes,” section 4.3 of the “Datatypes” chapter in the <cite>HDF5 User’s Guide</cite> for a discussion of creating and modifying datatypes.)</p>

<p>After HDF5 converts values from the memory datatype to the dataset datatype, it passes something like the following to the n-bit filter for compression:</p>

<table border="0" width="80%" align="center">

<tr>

<td>

<pre>

| byte 3 | byte 2 | byte 1 | byte 0 |

| |

|?????SEE|EEEE|MMMM|MMMMMMMM|M|???????|

|_____|

truncated mantissa

</pre>

</td></tr>

<tr>

<td colspan="4">

Key:

<code>S</code> - sign bit,

`E` - exponent bit,

`M` - mantissa bit,

`?` - padding bit

Sign bit is included in floating-point datatype precision.

The sign bit and truncated mantissa bits are not changed during datatype conversion by the HDF5 Library. On the other hand, the conversion of the 8-bit exponent to a 6-bit exponent is a little tricky:

The bias for the new exponent in the n-bit datatype is:

$2^{n-1} - 1$

The following formula is used for this exponent conversion:

$\text{exp8} - (2^{8-1} - 1) =$

$\text{exp6} - (2^{6-1} - 1) =$

actual exponent value

where `exp8` is the stored decimal value

as represented by the 8-bit exponent,

- Compress a data element of a no-op datatype
- Compress a data element of an atomic datatype
- Compress a data element of a compound datatype
- Compress a data element of an array datatype

<p>No-op datatypes:

The n-bit filter does not actually compress no-op datatypes.

Rather, it copies the data buffer of the no-op datatype from the noncompressed buffer to the proper location in the compressed buffer; the compressed buffer has no holes. The term “compress” is used here simply to distinguish this function from the function that performs the reverse operation during decompression.</p>

<p>Atomic datatypes:

The n-bit filter will find the bytes where significant bits are located and try to compress these bytes, one byte at a time, using a loop. At this level, the filter needs the following information:</p>

- The byte offset of the beginning of the current data element with respect to the beginning of the input data buffer
- Datatype size, precision, offset, and byte order

<p>The n-bit filter compresses from the most significant byte containing significant bits to the least significant byte.

For big-endian data, therefore, the loop index progresses from smaller

to larger while for little-endian, the loop index progresses from larger to smaller.

In the extreme case of when the n-bit datatype has full precision, this function copies the content of the entire noncompressed datatype to the compressed output buffer.

Compound datatypes:

The n-bit filter will compress each data member of the compound datatype.

If the member datatype is of an integer or floating-point datatype, the n-bit filter will call the function described above in section 2.1.2.

If the member datatype is of a no-op datatype, the filter will call the function described above in section 2.1.1.

If the member datatype is of a compound datatype, the filter will make a recursive call to itself.

!--

(i.e., to the function described in this section, 2.1.3).

-->

If the member datatype is of an array datatype, the filter will call the function described below in section 2.1.4.

Array datatypes:

The n-bit filter will use a loop to compress each array element in the array. If the base datatype of array element is of an integer or floating-point datatype, the n-bit filter will call the function described above in section 2.1.2.

If the base datatype is of a no-op datatype, the filter will call the function described above in section 2.1.1.

If the base datatype is of a compound datatype, the filter will call the

function described above

The first and second parameters can be obtained using the HDF5 dataspace and datatype interface calls. </p>

<!--

The third parameter is set during the storing process as described in section 3.2.

-->

A compound datatype can have members of array or compound datatype. An array datatype's base datatype can be a complex compound datatype. Recursive calls are required to set parameters for these complex situations.</p>

Before setting the parameters, the number of parameters should be calculated to dynamically allocate the array `cd_values[]`, which will be passed to the HDF5 Library. This also requires recursive calls.</p>

For an atomic datatype (integer or floating-point), parameters that will be stored include the datatype's size, endianness, precision, and offset. </p>

For a no-op datatype, only the size is required.</p>

For a compound datatype, parameters that will be stored include the datatype's total size and number of members. For each member, its member offset needs to be stored. Other parameters for members will depend on the respective datatype class.</p>

For an array datatype, the total size parameter should be stored. Other parameters for the array's base type depend on the base type's datatype class. </p>

Further, to correctly retrieve the parameter for use of n-bit compression or decompression later, parameters for distinguishing between datatype classes should be stored.

[Implementation](#)

Implementation

Three filter callback functions were written for the n-bit filter:

-

- `H5Z_can_apply_nbit`

- `H5Z_set_local_nbit`

- `H5Z_filter_nbit`

These functions are called internally by the HDF5 Library.

A number of utility functions were written for the function

`H5Z_set_local_nbit`. Compression and decompression functions were written and are called by function `H5Z_filter_nbit`.

All these functions are included in the file `H5Znbit.c`.

The public function `H5Pset_nbit` is called by the application to set up the use of the n-bit filter.

This function is included in the file `H5Pdcpl.c`.

The application does not need to supply any parameters.

How N-bit Parameters are Stored

A scheme of storing parameters required by the n-bit filter in the array `cd_values[]` was developed utilizing recursive function calls.

Four private utility functions were written for storing the parameters associated with atomic (integer or floating-point), no-op, array, and compound datatypes:

- `H5Z_set_parms_atomic`

- `H5Z_set_parms_array`

- `H5Z_set_parms_nooptype`

- `H5Z_set_parms_compound`

!-- NEW PAGE -->

The scheme is briefly described below.

<dir>

First, assign a numeric code for datatype class atomic (integer or float), no-op, array, and compound datatype. The code is stored before other datatype related parameters are stored.

<dl>

<dd>

<dt>The first three parameters of `cd_values[]` are reserved for:

- The number of valid entries in the array `cd_values[]`

- A flag indicating whether compression is needed

- The number of elements in the chunk

<dt>Throughout the balance of this explanation,

`i` represents the index of `cd_values[]`.

<dt>In the function <code>H5Z_set_local_nbit</code>:

<dd>

- `i` = 2

- Get the number of elements in the chunk and store in

```
<code>cd_value[i]</code>; increment <code>i</code></li>
```

- Get the class of the datatype:

 For an integer or floating-point datatype, call

```
<code>H5Z_set_parms_atomic</code>
```

 For an array datatype, call

```
<code>H5Z_set_parms_array</code>
```

 For a compound datatype, call

```
<code>H5Z_set_parms_compound</code>
```


 For none of the above, call

```
<code>H5Z_set_parms_noopdatatype</code></li>
```

- Store `i` in `cd_value[0]` and

flag in `cd_values[1]`

</dd>

<dl>

<dt>In the function <code>H5Z_set_parms_atomic</code>:</dt>

<dd>

- Store the assigned numeric code for the atomic datatype in

```
<code>cd_value[i]</code>; increment <code>i</code></li>
```

- Get the size of the atomic datatype and store in

```
<code>cd_value[i]</code>; increment <code>i</code></li>
```


- Get the order of the atomic datatype and store in

- `cd_value[i]`; increment `i`

- Get the precision of the atomic datatype and store in

- `cd_value[i]`; increment `i`

- Get the offset of the atomic datatype and store in

- `cd_value[i]`; increment `i`

- Determine the need to do compression at this point

-

-

-

-

- In the function `H5Z_set_parms_nooptype`:

-

-

- Store the assigned numeric code for the no-op datatype in

- `cd_value[i]`; increment `i`

- Get the size of the no-op datatype and store in

- `cd_value[i]`; increment `i`

-

-

-

-

- In the function `H5Z_set_parms_array`:

-

-

- Store the assigned numeric code for the array datatype in

- `cd_value[i]`; increment `i`

- Get the size of the array datatype and store in

- `cd_value[i]`; increment `i`

Get the class of the array's base datatype.

For an integer or floating-point datatype,

call `H5Z_set_parms_atomic`

For an array datatype, call

`H5Z_set_parms_array`

For a compound datatype, call

`H5Z_set_parms_compound`

If none of the above,

call `H5Z_set_parms_noopdatatype`

In the function `H5Z_set_parms_compound`:

Store the assigned numeric code for the compound datatype in

`cd_value[i]`; increment `i`

Get the size of the compound datatype and store in

`cd_value[i]`; increment `i`

Get the number of members and store in

`cd_values[i]`; increment `i`

For each member

Get the member offset and store in

`cd_values[i]`; increment `i`

Get the class of the member datatype

For an integer or floating-point datatype,

call `H5Z_set_parms_atomic`

For an array datatype,

call `H5Z_set_parms_array`

<h4>Usage Examples</h4>

<p>The following code example illustrates the use of the n-bit filter for writing and reading n-bit integer data.</p>

<table width="600" cellpadding="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

```
#include "hdf5.h"
```

```
#include "stdlib.h"
```

```
#include "math.h"
```

```
#define H5FILE_NAME "nbit_test_int.h5"
```

```
#define DATASET_NAME "nbit_int"
```

```
#define NX 200
```

```
#define NY 300
```

```
#define CH_NX 10
```

```
#define CH_NY 15
```

```
int main(void)
```

```
{
```

```
    hid_t file, dataspace, dataset, datatype, mem_datatype, dset_create_props;
```

```
    hsize_t dims[2], chunk_size[2];
```

```
    int orig_data[NX][NY];
```

```
    int new_data[NX][NY];
```

```
    int i, j;
```

```
    size_t precision, offset;
```

```
/* Define dataset datatype (integer), and set precision, offset */
datatype = H5Tcopy(H5T_NATIVE_INT);
precision = 17; /* precision includes sign bit */
if(H5Tset_precision(datatype,precision)&lt;0) {
    printf("Error: fail to set precision\n");
    return -1;
}
offset = 4;
if(H5Tset_offset(datatype,offset)&lt;0) {
    printf("Error: fail to set offset\n");
    return -1;
}

/* Copy to memory datatype */
mem_datatype = H5Tcopy(datatype);

/* Set order of dataset datatype */
if(H5Tset_order(datatype, H5T_ORDER_BE)&lt;0) {
    printf("Error: fail to set endianness\n");
    return -1;
}

/* Initiliazе data buffer with random data within correct range
 * corresponding to the memory datatype's precision and offset.
 */
for (i=0; i &lt; NX; i++)
```

```
for (j=0; j < NY; j++)
    orig_data[i][j] = rand() % (int)pow(2, precision-1) && offset;

/* Describe the size of the array. */
dims[0] = NX;
dims[1] = NY;
if((dataspace = H5Screate_simple (2, dims, NULL))&lt;0) {
    printf("Error: fail to create dataspace\n");
    return -1;
}

/*
 * Create a new file using read/write access, default file
 * creation properties, and default file access properties.
 */
if((file = H5Fcreate (H5FILE_NAME, H5F_ACC_TRUNC,
                    H5P_DEFAULT, H5P_DEFAULT))&lt;0) {
    printf("Error: fail to create file\n");
    return -1;
}

/*
 * Set the dataset creation property list to specify that
 * the raw data is to be partitioned into 10 x 15 element
 * chunks and that each chunk is to be compressed.
 */
chunk_size[0] = CH_NX;
chunk_size[1] = CH_NY;
```

```

if((dset_create_props = H5Pcreate (H5P_DATASET_CREATE))&lt;0) {
    printf("Error: fail to create dataset property\n");
    return -1;
}
if(H5Pset_chunk (dset_create_props, 2, chunk_size)&lt;0) {
    printf("Error: fail to set chunk\n");
    return -1;
}

```

</pre><!-- NEW PAGE -->

<pre>

/*

* Set parameters for n-bit compression; check the description of
 * the H5Pset_nbit function in the HDF5 Reference Manual for more
 * information.

*/

```

if(H5Pset_nbit (dset_create_props)&lt;0) {
    printf("Error: fail to set nbit filter\n");
    return -1;
}

```

/*

* Create a new dataset within the file. The datatype
 * and dataspace describe the data on disk, which may
 * be different from the format used in the application's
 * memory.

*/

```

if((dataset = H5Dcreate(file, DATASET_NAME, datatype,
    dataspace, H5P_DEFAULT,

```

```
        dset_create_props, H5P_DEFAULT))&lt;0) {
    printf("Error: fail to create dataset\n");
    return -1;
}

/*
 * Write the array to the file. The datatype and dataspace
 * describe the format of the data in the 'orig_data' buffer.
 * The raw data is translated to the format required on disk,
 * as defined above. We use default raw data transfer properties.
 */
if(H5Dwrite (dataset, mem_datatype, H5S_ALL, H5S_ALL,
            H5P_DEFAULT, orig_data)&lt;0) {
    printf("Error: fail to write to dataset\n");
    return -1;
}

H5Dclose (dataset);

if((dataset = H5Dopen(file, DATASET_NAME, H5P_DEFAULT))&lt;0) {
    printf("Error: fail to open dataset\n");
    return -1;
}

/*
 * Read the array. This is similar to writing data,
 * except the data flows in the opposite direction.
```


* Note: Decompression is automatic.

*/

```
if(H5Dread (dataset, mem_datatype, H5S_ALL, H5S_ALL,
          H5P_DEFAULT, new_data)&lt;0) {
    printf("Error: fail to read from dataset\n");
    return -1;
}
```

</pre><!-- NEW PAGE -->

<pre>

```
H5Tclose (datatype);
H5Tclose (mem_datatype);
H5Dclose (dataset);
H5Sclose (dataspace);
H5Pclose (dset_create_props);
H5Fclose (file);
```

```
return 0;
```

}</pre></td>

</tr>

<tr><td><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td align="left">

Example 10. N-bit compression for integer data

Illustrates the use of the n-bit filter for writing and reading
n-bit integer data.

<hr color="green" size="3" /></td>

</tr>

</table>

The following code example illustrates the use of the n-bit filter for writing and reading n-bit floating-point data.

--

--

--

--

```
<pre>
```

```
#include "hdf5.h"
```

```
#define H5FILE_NAME "nbit_test_float.h5"
```

```
#define DATASET_NAME "nbit_float"
```

```
#define NX 2
```

```
#define NY 5
```

```
#define CH_NX 2
```

```
#define CH_NY 5
```

```
int main(void)
```

```
{
```

```
    hid_t file, dataspace, dataset, datatype, dset_create_props;
```

```
    hsize_t dims[2], chunk_size[2];
```

```
    /* orig_data[] are initialized to be within the range that can be
```

```
    * represented by dataset datatype (no precision loss during
```

```
    * datatype conversion)
```

```
    */
```

```
    float orig_data[NX][NY] = {{188384.00, 19.103516, -1.0831790e9,
```

```
-84.242188, 5.2045898}, {-49140.000, 2350.2500, -3.2110596e-1,
6.4998865e-5, -0.0000000}}};
float  new_data[NX][NY];
size_t precision, offset;
```

```
/* Define single-precision floating-point type for dataset
```

```
*-----
```

```
* size=4 byte, precision=20 bits, offset=7 bits,
```

```
* mantissa size=13 bits, mantissa position=7,
```

```
* exponent size=6 bits, exponent position=20,
```

```
* exponent bias=31.
```

```
* It can be illustrated in little-endian order as:
```

```
* (S - sign bit, E - exponent bit, M - mantissa bit,
```

```
* ? - padding bit)
```

```
*
```

```
*      3      2      1      0
```

```
*      ?????SEE EEEEE MMMMMMMMMMMMMMM M???????
```

```
*
```

```
* To create a new floating-point type, the following
```

```
* properties must be set in the order of
```

```
*   set fields -> set offset -> set precision -> set size.
```

```
* All these properties must be set before the type can function.
```

```
* Other properties can be set anytime. Derived type size cannot
```

```
* be expanded bigger than original size but can be decreased.
```

```
* There should be no holes among the significant bits. Exponent
```

```
* bias usually is set  $2^{(n-1)}-1$ , where n is the exponent size.
```

```
*-----*/
```

```
datatype = H5Tcopy(H5T_IEEE_F32BE);
```

```
if(H5Tset_fields(datatype, 26, 20, 6, 7, 13)&lt;0) {
    printf("Error: fail to set fields\n");
    return -1;
}
offset = 7;
if(H5Tset_offset(datatype,offset)&lt;0) {
    printf("Error: fail to set offset\n");
    return -1;
}
precision = 20;
if(H5Tset_precision(datatype,precision)&lt;0) {
    printf("Error: fail to set precision\n");
    return -1;
}
if(H5Tset_size(datatype, 4)&lt;0) {
    printf("Error: fail to set size\n");
    return -1;
}
if(H5Tset_ebias(datatype, 31)&lt;0) {
    printf("Error: fail to set exponent bias\n");
    return -1;
}

/* Describe the size of the array. */
dims[0] = NX;
dims[1] = NY;
if((dataspace = H5Screate_simple (2, dims, NULL))&lt;0) {
    printf("Error: fail to create dataspace\n");
    return -1;
}
```

```
/*
 * Create a new file using read/write access, default file
 * creation properties, and default file access properties.
 */
if((file = H5Fcreate (H5FILE_NAME, H5F_ACC_TRUNC,
                    H5P_DEFAULT, H5P_DEFAULT))&lt;0) {
    printf("Error: fail to create file\n");
    return -1;
}

/*
 * Set the dataset creation property list to specify that
 * the raw data is to be partitioned into 2 x 5 element
 * chunks and that each chunk is to be compressed.
 */
chunk_size[0] = CH_NX;
chunk_size[1] = CH_NY;
if((dset_create_props = H5Pcreate (H5P_DATASET_CREATE))&lt;0) {
    printf("Error: fail to create dataset property\n");
    return -1;
}
if(H5Pset_chunk (dset_create_props, 2, chunk_size)&lt;0) {
    printf("Error: fail to set chunk\n");
    return -1;
}

/*
```

```
* Set parameters for n-bit compression; check the description  
* of the H5Pset_nbit function in the HDF5 Reference Manual  
* for more information.
```

```
*/
```

```
if(H5Pset_nbit (dset_create_props)&lt;0) {  
    printf("Error: fail to set nbit filter\n");  
    return -1;  
}
```

```
/*
```

```
* Create a new dataset within the file. The datatype  
* and dataspace describe the data on disk, which may  
* be different from the format used in the application's  
* memory.
```

```
*/
```

```
if((dataset = H5Dcreate(file, DATASET_NAME, datatype,  
    dataspace, H5P_DEFAULT,  
    dset_creat_plists, H5P_DEFAULT))&lt;0) {  
    printf("Error: fail to create dataset\n");  
    return -1;  
}
```

```
/*
```

```
* Write the array to the file. The datatype and dataspace  
* describe the format of the data in the 'orig_data' buffer.  
* The raw data is translated to the format required on disk,  
* as defined above. We use default raw data transfer properties.
```

```
*/
```

```
if(H5Dwrite (dataset, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL,
```

```
        H5P_DEFAULT, orig_data)&lt;0) {
    printf("Error: fail to write to dataset\n");
    return -1;
}

H5Dclose (dataset);

if((dataset = H5Dopen(file, DATASET_NAME, H5P_DEFAULT))&lt;0) {
    printf("Error: fail to open dataset\n");
    return -1;
}

/*
 * Read the array. This is similar to writing data,
 * except the data flows in the opposite direction.
 * Note: Decompression is automatic.
 */
if(H5Dread (dataset, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL,
           H5P_DEFAULT, new_data)&lt;0) {
    printf("Error: fail to read from dataset\n");
    return -1;
}

H5Tclose (datatype);
H5Dclose (dataset);
H5Sclose (dataspace);
H5Pclose (dset_create_props);
```

```
H5Fclose (file);
```

```
return 0;
```

```
}</pre></td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 11. N-bit compression for floating-point data</b><br />
```

```
Illustrates the use of the n-bit filter for writing and reading
```

```
n-bit floating-point data.
```

```
<hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

```
<a name="limitations">
```

```
<h4><em>Limitations</em></h4>
```

```
</a>
```

Because the array `cd_values[]` has to fit into an object header message of 64K, the n-bit filter has an upper limit on the number of n-bit parameters that can be stored in it. To be conservative, a maximum of 4K is allowed for the number of parameters.

The n-bit filter currently only compresses n-bit datatypes or fields derived from integer or floating-point datatypes. The n-bit filter assumes padding

bits of zero. This may not be true since the HDF5 user can set padding bit to be zero, one, or leave the background alone. However, it is expected the n-bit filter will be modified to adjust to such situations.

The n-bit filter does not have a way to handle the situation where the fill value of a dataset is defined and the fill value is not of an n-bit datatype although the dataset datatype is.

<!-- NEW PAGE -->

<h3>5.6.2. Using the Scale-offset Filter</h3>

Generally speaking, scale-offset compression performs a scale and/or offset operation on each data value and truncates the resulting value to a minimum number of bits (minimum-bits) before storing it.

The current scale-offset filter supports integer and floating-point datatypes only. For the floating-point datatype, float and double are supported, but long double is not supported.

Integer data compression uses a straight-forward algorithm. Floating-point data compression adopts the GRiB data packing mechanism which offers two alternate methods: a fixed minimum-bits method, and a variable minimum-bits method. Currently, only the variable minimum-bits method is implemented. <!-- 9.3.10, MEE: according to Kent, the fixed minimum-bits method has not yet been implemented, and they do not have any plans to implement it. -->

Like other I/O filters supported by the HDF5 Library, applications

using the scale-offset filter must store data with chunked storage.

Integer type:

The minimum-bits of integer data can be determined by the filter.

For example, if the maximum value of data to be compressed is 7065 and the minimum value is 2970. Then the "span" of dataset values is equal to $(\text{max} - \text{min} + 1)$, which is 4676. If no fill value is defined for the dataset, the minimum-bits is:

`ceiling(log2(span)) = 12`. With fill value set, the minimum-bits is: `ceiling(log2(span+1)) = 13`.

HDF5 users can also set the minimum-bits. However, if the user gives a minimum-bits that is less than that calculated by the filter, the compression will be lossy.

Floating-point type:

The basic idea of the scale-offset filter for the floating-point type is to transform the data by some kind of scaling to integer data, and then to follow the procedure of the scale-offset filter for the integer type to do the data compression. Due to the data transformation from floating-point to integer, the scale-offset filter is lossy in nature.

Two methods of scaling the floating-point data are used: the so-called D-scaling and E-scaling. D-scaling is more straightforward and easy to understand. For HDF5 1.8 release, only the D-scaling method has been implemented. 9.3.10, MEE: According to Kent, E-scaling has not yet been implemented, and they have no plans to implement it in the future.

Design

Before the filter does any real work, it needs to gather some information from the HDF5 Library through API calls. The parameters the filter needs are:

- The minimum-bits of the data value

- The number of data elements in the chunk

- The datatype class, size, sign (only for integer type), byte order, and fill value if defined

Size and sign are needed to determine what kind of pointer cast to use when retrieving values from the data buffer.

The pipeline of the filter can be divided into four parts:

(1)pre-compression; (2)compression; (3)decompression;

(4)post-decompression.

Depending on whether a fill value is defined or not, the filter will handle pre-compression and post-decompression differently.

The scale-offset filter only needs the memory byte order, size of datatype, and minimum-bits for compression and decompression.

Since decompression has no access to the original data, the minimum-bits and the minimum value need to be stored with the compressed data for decompression and post-decompression.

Integer Type

Pre-compression:

During pre-compression minimum-bits is calculated if it is not set by the user. For more information on how minimum-bits are calculated, see section 6.1. "The N-bit Filter";

If the fill value is defined, finding the maximum and minimum values should ignore the data element whose value is equal to the fill value.

If no fill value is defined, the value of each data element is subtracted by the minimum value during this stage.

If the fill value is defined, the fill value is assigned to the maximum value. In this way minimum-bits can represent a data element whose value is equal to the fill value and subtracts the minimum value from a data element whose value is not equal to the fill value.

8.19.10, MEE: the paragraph below is Frank's revision of my editing

9.3.10, MEE: Kent reviewed the paragraph below and said it was clear.

The fill value (if defined), the number of elements in a chunk, the class of the datatype, the size of the datatype, the memory order of the datatype, and other similar elements will be stored in the HDF5 object header for the post-decompression usage.

After pre-compression, all values are non-negative and are within the range that can be stored by minimum-bits.

Compression:

All modified data values after pre-compression are packed together into the compressed data buffer. The number of bits for each data value decreases from the number of bits of integer (32 for most platforms) to minimum-bits. The value of minimum-bits and the minimum value are added to

the data buffer and the whole buffer is sent back to the library. In this way, the number of bits for each modified value is no more than the size of minimum-bits.

Decompression:

In this stage, the number of bits for each data value is resumed from minimum-bits to the number of bits of integer.

Post-decompression:

For the post-decompression stage, the filter does the opposite of what it does during pre-compression except that it does not calculate the minimum-bits or the minimum value. These values were saved during compression and can be retrieved through the resumed data buffer. If no fill value is defined, the filter adds the minimum value back to each data element.

If the fill value is defined, the filter assigns the fill value to the data element whose value is equal to the maximum value that minimum-bits can represent and adds the minimum value back to each data element whose value is not equal to the maximum value that minimum-bits can represent.

[SO_FloatingPoint](#)

Floating-point Type

The filter will do data transformation from floating-point type to integer type and then handle the data by using the procedure for handling the integer data inside the filter.

Insignificant bits of floating-point data will be cut off during data transformation, so this filter is a lossy compression method.

There are two scaling methods: D-scaling and E-scaling.

The HDF5 1.8 release only supports D-scaling. D-scaling is short for decimal scaling. E-scaling should be similar conceptually. In order to transform data from floating-point to integer, a scale factor is introduced. The minimum value will be calculated. Each data element value will subtract the minimum value. The modified data will be multiplied by 10 (Decimal) to the power of `scale_factor`, and only the integer part will be kept and manipulated through the routines for the integer type of the filter during pre-compression and compression. Integer data will be divided by 10 to the power of `scale_factor` to transform back to floating-point data during decompression and post-decompression. Each data element value will then add the minimum value, and the floating-point data are resumed. However, the resumed data will lose some insignificant bits compared with the original value.

For example, the following floating-point data are manipulated by the filter, and the D-scaling factor is 2.

`{104.561, 99.459, 100.545, 105.644}`

The minimum value is 99.459, each data element subtracts 99.459, the modified data is

`{5.102, 0, 1.086, 6.185}`

Since the D-scaling factor is 2, all floating-point data will be multiplied by 10^2 with this result:

```
<code>{510.2, 0, 108.6, 618.5}</code>
```

<p>The digit after decimal point will be rounded off, and then the set looks like: </p>

```
<code>{510 , 0, 109, 619}</code>
```

<p>After decompression, each value will be divided by 10^2 and will be added to the offset 99.459.</p>

<p>The floating-point data becomes </p>

```
<code>{104.559, 99.459, 100.549, 105.649}</code>.
```

<p>The relative error for each value should be no more than $5 * (10^{(D\text{-scaling factor} + 1)})$. D-scaling sometimes is also referred as a variable minimum-bits method since for different datasets the minimum-bits to represent the same decimal precision will vary. The data value is modified to 2 to power of `scale_factor` for E-scaling. E-scaling is also called fixed-bits method since for different datasets the minimum-bits will always be fixed to the scale factor of E-scaling. Currently HDF5 ONLY supports D-scaling (variable minimum-bits) method.</p>

<h4>Implementation</h4>

<p>The scale-offset filter implementation was written and included in the file `H5Zscaleoffset.c`. Function `H5Pset_scaleoffset` was written and included in the file `H5Pdcpl.c`. The HDF5 user can supply minimum-bits by calling function

```
<code>H5Pset_scaleoffset</code>.</p>
```

```
<!-- NEW PAGE -->
```

The scale-offset filter was implemented based on the design outlined in this section. However, the following factors need to be considered:

<dl>

<dd>

The filter needs the appropriate cast pointer whenever it needs to retrieve data values.

The HDF5 Library passes to the filter the to-be-compressed data in the format of the dataset datatype, and the filter passes back the decompressed data in the same format. If a fill value is defined, it is also in dataset datatype format.

For example, if the byte order of the dataset datatype is different from that of the memory datatype of the platform, compression or decompression performs an endianness conversion of data buffer. Moreover, it should be aware that memory byte order can be different during compression and decompression.

The difference of endianness and datatype between file and memory should be considered when saving and retrieval of minimum-bits, minimum value, and fill value.

If the user sets the minimum-bits to full precision of the datatype, no operation is needed at the filter side. If the full precision is a result of calculation by the filter, then the minimum-bits needs to be saved for decompression but no compression or decompression is needed (only a copy of the input buffer is needed).

If by calculation of the filter, the minimum-bits is equal to zero, special handling is needed. Since it means all values are the same, no compression or decompression is needed. But the minimum-bits

and minimum value still need to be saved during compression.

For floating-point data, the minimum value of the dataset should be calculated at first. Each data element value will then subtract the minimum value to obtain the "offset" data.

The offset data will then follow the steps outlined above in the discussion of [floating-point types](#SO_FloatingPoint) to do data transformation to integer and rounding.

Usage Examples

The following code example illustrates the use of the scale-offset filter for writing and reading integer data.

--	--

--	--

--	--

--	--

```


```

```
#include "hdf5.h"
```

```
#include "stdlib.h"
```

```
#define H5FILE_NAME "scaleoffset_test_int.h5"
```

```
#define DATASET_NAME "scaleoffset_int"
```

```
#define NX 200
```

```
#define NY 300
```

```
#define CH_NX 10
```

```
#define CH_NY 15
```

```


```

```
<pre>
int main(void)
{
    hid_t file, dataspace, dataset, datatype, dset_create_props;
    hsize_t dims[2], chunk_size[2];
    int orig_data[NX][NY];
    int new_data[NX][NY];
    int i, j, fill_val;

    /* Define dataset datatype */
    datatype = H5Tcopy(H5T_NATIVE_INT);

    /* Initilize data buffer */
    for (i=0; i < NX; i++)
        for (j=0; j < NY; j++)
            orig_data[i][j] = rand() % 10000;

    /* Describe the size of the array. */
    dims[0] = NX;
    dims[1] = NY;
    if((dataspace = H5Screate_simple (2, dims, NULL))&lt;0) {
        printf("Error: fail to create dataspace\n");
        return -1;
    }

    /*
     * Create a new file using read/write access, default file
     * creation properties, and default file access properties.
     */
    if((file = H5Fcreate (H5FILE_NAME, H5F_ACC_TRUNC,
                        H5P_DEFAULT, H5P_DEFAULT))&lt;0) {
```

```
    printf("Error: fail to create file\n");
    return -1;
}

/*
 * Set the dataset creation property list to specify that
 * the raw data is to be partitioned into 10 x 15 element
 * chunks and that each chunk is to be compressed.
 */
chunk_size[0] = CH_NX;
chunk_size[1] = CH_NY;
if((dset_create_props = H5Pcreate (H5P_DATASET_CREATE))&lt;0) {
    printf("Error: fail to create dataset property\n");
    return -1;
}
if(H5Pset_chunk (dset_create_props, 2, chunk_size)&lt;0) {
    printf("Error: fail to set chunk\n");
    return -1;
}

/* Set the fill value of dataset */
fill_val = 10000;
if (H5Pset_fill_value(dset_create_props, H5T_NATIVE_INT,
    &fill_val)&lt;0) {
    printf("Error: can not set fill value for dataset\n");
    return -1;
}

/*
 * Set parameters for scale-offset compression. Check the
 * description of the H5Pset_scaleoffset function in the
```

```
* HDF5 Reference Manual for more information [3].
*/
if(H5Pset_scaleoffset (dset_create_props, H5Z_SO_INT,
                      H5Z_SO_INT_MINIMUMBITS_DEFAULT)&lt;0) {
    printf("Error: fail to set scaleoffset filter\n");
    return -1;
}

/*
 * Create a new dataset within the file. The datatype
 * and dataspace describe the data on disk, which may
 * or may not be different from the format used in the
 * application's memory. The link creation and
 * dataset access property list parameters are passed
 * with default values.
 */
if((dataset = H5Dcreate (file, DATASET_NAME, datatype,
                      dataspace, H5P_DEFAULT,
                      dset_create_props, H5P_DEFAULT))&lt;0) {
    printf("Error: fail to create dataset\n");
    return -1;
}

/*
 * Write the array to the file. The datatype and dataspace
 * describe the format of the data in the 'orig_data' buffer.
 * We use default raw data transfer properties.
 */
if(H5Dwrite (dataset, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
            H5P_DEFAULT, orig_data)&lt;0) {
    printf("Error: fail to write to dataset\n");
```

```

        return -1;
    }

    H5Dclose (dataset);

    if((dataset = H5Dopen(file, DATASET_NAME, H5P_DEFAULT))&lt;0) {
        printf("Error: fail to open dataset\n");
        return -1;
    }

    /*
    * Read the array. This is similar to writing data,
    * except the data flows in the opposite direction.
    * Note: Decompression is automatic.
    */
    if(H5Dread (dataset, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
               H5P_DEFAULT, new_data)&lt;0) {
        printf("Error: fail to read from dataset\n");
        return -1;
    }

    H5Tclose (datatype);
    H5Dclose (dataset);
    H5Sclose (dataspace);
    H5Pclose (dset_create_props);
    H5Fclose (file);

    return 0;
}

```

```

<tr valign="top">
  <td align="left">
    <b>Example 12. Scale-offset compression integer data</b><br />
    Illustrates the use of the scale-offset filter for writing
    and reading integer data.
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

```

<!-- NEW PAGE -->
<p>The following code example illustrates the use of the scale-offset filter
(set for variable minimum-bits method) for writing and reading
floating-point data.</p>

```

```

<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
    </td>
  </tr>
</table>

#include "hdf5.h"
#include "stdlib.h"

#define H5FILE_NAME "scaleoffset_test_float_Dscale.h5"
#define DATASET_NAME "scaleoffset_float_Dscale"
#define NX 200
#define NY 300
#define CH_NX 10
#define CH_NY 15

</pre>

<pre>

```

```
int main(void)
{
    hid_t  file, dataspace, dataset, datatype, dset_create_props;
    hsize_t dims[2], chunk_size[2];
    float  orig_data[NX][NY];
    float  new_data[NX][NY];
    float  fill_val;
    int    i, j;

    /* Define dataset datatype */
    datatype = H5Tcopy(H5T_NATIVE_FLOAT);

    /* Initilize data buffer */
    for (i=0; i < NX; i++)
        for (j=0; j < NY; j++)
            orig_data[i][j] = (rand() % 10000) / 1000.0;

    /* Describe the size of the array. */
    dims[0] = NX;
    dims[1] = NY;
    if((dataspace = H5Screate_simple (2, dims, NULL))&lt;0) {
        printf("Error: fail to create dataspace\n");
        return -1;
    }

    /*
     * Create a new file using read/write access, default file
     * creation properties, and default file access properties.
     */
    if((file = H5Fcreate (H5FILE_NAME, H5F_ACC_TRUNC,
                        H5P_DEFAULT, H5P_DEFAULT))&lt;0) {
```

```
    printf("Error: fail to create file\n");
    return -1;
}

/*
 * Set the dataset creation property list to specify that
 * the raw data is to be partitioned into 10 x 15 element
 * chunks and that each chunk is to be compressed.
 */
chunk_size[0] = CH_NX;
chunk_size[1] = CH_NY;
if((dset_create_props = H5Pcreate (H5P_DATASET_CREATE))&lt;0) {
    printf("Error: fail to create dataset property\n");
    return -1;
}
if(H5Pset_chunk (dset_create_props, 2, chunk_size)&lt;0) {
    printf("Error: fail to set chunk\n");
    return -1;
}

/* Set the fill value of dataset */
fill_val = 10000.0;
if (H5Pset_fill_value(dset_create_props, H5T_NATIVE_FLOAT,
    &fill_val)<0) {
    printf("Error: can not set fill value for dataset\n");
    return -1;
}

/*
 * Set parameters for scale-offset compression; use variable
 * minimum-bits method, set decimal scale factor to 3. Check the
```



```
* description of the H5Pset_scaleoffset function in the HDF5
* Reference Manual for more information [3].
*/
if(H5Pset_scaleoffset (dset_create_props, H5Z_SO_FLOAT_DSCALE, 3)&lt;0) {
    printf("Error: fail to set scaleoffset filter\n");
    return -1;
}

/*
* Create a new dataset within the file. The datatype
* and dataspace describe the data on disk, which may
* or may not be different from the format used in the
* application's memory.
*/
if((dataset = H5Dcreate (file, DATASET_NAME, datatype,
                        dataspace, H5P_DEFAULT,
                        dset_create_props, H5P_DEFAULT))&lt;0) {
    printf("Error: fail to create dataset\n");
    return -1;
}

/*
* Write the array to the file. The datatype and dataspace
* describe the format of the data in the 'orig_data' buffer.
* We use default raw data transfer properties.
*/
if(H5Dwrite (dataset, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL,
            H5P_DEFAULT, orig_data)&lt;0) {
    printf("Error: fail to write to dataset\n");
    return -1;
}
```

```

H5Dclose (dataset);

if((dataset = H5Dopen(file, DATASET_NAME, H5P_DEFAULT))&lt;0) {
    printf("Error: fail to open dataset\n");
    return -1;
}

/*
 * Read the array. This is similar to writing data,
 * except the data flows in the opposite direction.
 * Note: Decompression is automatic.
 */
if(H5Dread (dataset, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL,
           H5P_DEFAULT, new_data)&lt;0) {
    printf("Error: fail to read from dataset\n");
    return -1;
}

H5Tclose (datatype);
H5Dclose (dataset);
H5Sclose (dataspace);
H5Pclose (dset_create_props);
H5Fclose (file);

return 0;
}

```

Example 13. Scale-offset compression floating-point data

Illustrates the use of the scale-offset filter for writing
and reading floating-point data.

Limitations

For floating-point data handling, there are some algorithmic
limitations to the GRiB data packing mechanism:

- Both the E-scaling and D-scaling methods are lossy compression

- For the D-scaling method, since data values have been rounded to integer values (positive) before truncating to the minimum-bits, their range is limited by the maximum value that can be represented by the corresponding unsigned integer type (the same size as that of the floating-point type)

Suggestions

The following are some suggestions for using the filter for floating-point data:

It is better to convert the units of data so that the units are within certain common range (for example, 1200m to 1.2km)

If data values to be compressed are very near to zero, it is strongly recommended that the user sets the fill value away from zero (for example, a large positive number); if the user does

nothing, the HDF5 Library will set the fill value to zero, and this may cause undesirable compression results

Users are not encouraged to use a very large decimal scale factor (e.g. 100) for the D-scaling method; this can cause the filter not to ignore the fill value when finding maximum and minimum values, and they will get a much larger minimum-bits (poor compression)

[Szip](#)

5.6.3. Using the Szip Filter

<p>See The HDF Group website for
further
information regarding the Szip filter.</p>

<p> </p>

<p> </p>

<!-- HEADER RIGHT " " -->

</body>

</html>

I

```

<!doctype HTML public "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
<head>
<title>Chapter 6: HDF5 Datatypes</title>

<!--(Meta)=====-->

<!--(Links)=====-->

<!--( Begin styles definition )=====-->
<link href="ed_styles/NewUglect.css" rel="stylesheet" type="text/css">
<!--( End styles definition )=====-->

</head>

<body>

<!-- #BeginLibraryItem "/ed_libs/Copyright.lbi" -->
<!--
* * * * *
* Copyright by The HDF Group. *
* Copyright by the Board of Trustees of the University of Illinois. *
* All rights reserved. *
* *
* This file is part of HDF5. The full HDF5 copyright notice, including *
* terms governing use, modification, and redistribution, is contained in *
* the files COPYING and Copyright.html. COPYING can be found at the root *
* of the source code distribution tree; Copyright.html can be found at the *
* root level of an installed copy of the electronic HDF5 document set and *
* is linked from the top-level documents page. It can also be found at *
* http://www.hdfgroup.org/HDF5/doc/Copyright.html. If you do not have *

```

```

* access to either file, you may request a copy from help@hdfgroup.org.  *
* * * * *
-->
<!-- #EndLibraryItem --><!-- HEADER LEFT "HDF5 User's Guide" -->
<!-- HEADER RIGHT "HDF5 Datatypes" -->

<!-- ( TOC )=====-->
<!--<SCRIPT language="JavaScript">-->
<!--
document.writeln ('\
<table x-use-null-cells\
    align=right\
width="240"\
cellspacing="0"\
class="tocTable">\
    <tr valign="top"> \
        <td class="tocTableHeaderCell" colspan="2"> \
            <span class="TableHead">Chapter Contents</span></td>\
        </tr>\
-->
<!-- Table Version 3 -->
<!--
\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#Intro">1.</a></td>\
    <td class="tocTableContentCell3">\
        <a href="#Intro">Introduction</a></td> \
    </tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \

```

```

    <a href="#DtypesUsed">2.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#DtypesUsed">How Datatypes Are Used</a></td>\
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#FileFunctSums">3.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#FileFunctSums">Datatype (H5T) Function Summaries</a></td> \
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#Pmodel">4.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#Pmodel">The Programming Model</a></td>\
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#NonNumDtypes">5.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#NonNumDtypes">Other Non-numeric Datatypes</a></td> \
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#Fvalues">6.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#Fvalues">Fill Values</a></td>\
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#CCDtypes">7.</a></td>\

```

```

    <td class="tocTableContentCell3">\
    <a href="#CCDtypes">Complex Combinations of Datatypes</a>\
  </td>\
</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#LCDtypeObj">8.</a></td>\
  <td class="tocTableContentCell3">\
    <a href="#LCDtypeObj">Life Cycle of the Datatype Object</a>\
  </td>\
</tr>\
\
  <tr valign="top"> \
    <td class="tocTableContentCell"> \
-->
<!-- editingComment -- "tocTableContentCell" and "tocTableContentCell4" \
\-->
<!-- are the table-closing cell class.\
  <td class="tocTableContentCell2"> \
\-->
<!--
  <a href="#Dtransfer">9.</a></td>\
  <td class="tocTableContentCell4">\
    <a href="#Dtransfer">Data Transfer: Datatype Conversion and Selection</a>\
  </td></tr>\
</table>\
')
-->
<!-- </SCRIPT> -->
<!--(End TOC)=====-->

```

```
<!-- editingComment
  <span class="editingComment">[ [
    ] ]</span>
-->
```

```
<!-- editingComment
-->
```

```
<div align="center">
<a name="top">
```

7. HDF5 Datatypes

</div>

<dir>

<!-- editingComment

[[[

<h1 class="editingComment align=center">- - - DRAFT - - -</h1>

<p class="editingComment">- - - This is an early draft of the Datatypes chapter of the new HDF5 User's Guide; much of this material will appear in the published version of the new UG, but some will appear in other documents, such as the HDF5 Reference Manual or the HDF5 Tutorial. A PDF version of this draft is being made available to HDF5 users prior to publication of the new UG because it contains a great deal of information that is not otherwise available.

]]

-->

</dir>

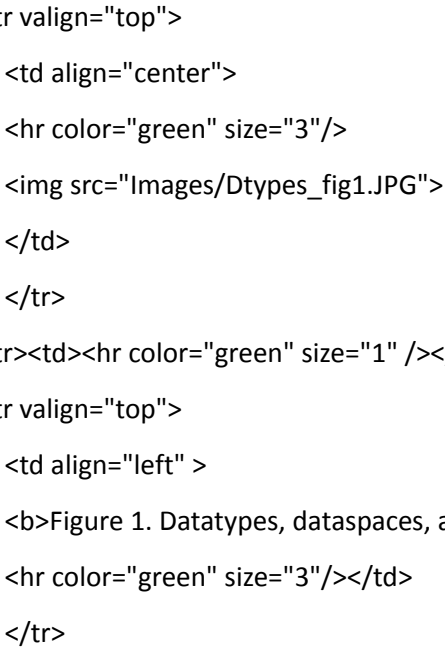
<h3>6.1. Introduction</h3>

<h4>6.1.1. Introduction and Definitions</h4>

<p>An HDF5 dataset is an array of data elements, arranged according to the specifications of the dataspace. In general, a data element is the smallest

addressable unit of storage in the HDF5 file. (Compound datatypes are the exception to this rule.) The HDF5 datatype defines the storage format for a single data element. See the figure below.

The model for HDF5 attributes is extremely similar to datasets: an attribute has a dataspace and a datatype, as shown in the figure below. The information in this chapter applies to both datasets and attributes.


<p>Figure 1. Datatypes, dataspace, and datasets</p>

Abstractly, each data element within the dataset is a sequence of bits, interpreted as a single value from a set of values (e.g., a number or a character). For a given datatype, there is a standard or convention for representing the values as bits, and when the bits are represented in a particular storage the bits are laid out in a specific storage scheme, e.g., as 8-bit bytes, with a specific ordering and alignment

of bytes within the storage array.</p>

<p>HDF5 datatypes implement a flexible, extensible, and portable mechanism for specifying and discovering the storage layout of the data elements, determining how to interpret the elements (e.g., as floating point numbers), and for transferring data from different compatible layouts.</p>

<!-- NEW PAGE -->

<p>An HDF5 datatype describes one specific layout of bits. A dataset has a single datatype which applies to every data element. When a dataset is created, the storage datatype is defined. After the dataset or attribute is created, the datatype cannot be changed.</p>

The datatype describes the storage layout of a single data element

All elements of the dataset must have the same type

The datatype of a dataset is immutable

<p>When data is transferred (e.g., a read or write), each end point of the transfer has a datatype, which describes the correct storage for the elements. The source and destination may have different (but compatible) layouts, in which case the data elements are automatically transformed during the transfer.</p>

<p>HDF5 datatypes describe commonly used binary formats for numbers (integers and floating point) and characters (ASCII). A given computing architecture and programming language supports certain number and character representations. For example, a computer may support 8-, 16-, 32-, and 64-bit signed integers, stored in memory in little-endian byte order. These would presumably correspond to the C programming language types ‘char’, ‘short’, ‘int’, and ‘long’.</p>

When reading and writing from memory, the HDF5 library must know the appropriate datatype that describes the architecture specific layout.

The HDF5 library provides the platform independent `H5T_NATIVE_` types, which are mapped to an appropriate datatype for each platform. So the type `H5T_NATIVE_INT` is an alias for the appropriate descriptor for each platform.

Data in memory has a datatype:

- The storage layout in memory is architecture-specific

- The HDF5 `H5T_NATIVE_` types are predefined aliases for the architecture-specific memory layout

- The memory datatype need not be the same as the stored datatype of the dataset

In addition to numbers and characters, an HDF5 datatype can describe more abstract classes of types, including

date-times,

(TIME REFERENCES COMMENTED OUT 6 FEB 2006,

UNTIL TIME DATATYPE IS PROPERLY SUPPORTED IN THE LIBRARY) -->

enumerations, strings, bit strings, and references (pointers to objects in the HDF5 file). HDF5 supports several classes of composite datatypes which are combinations of one or more other datatypes. In addition to the standard predefined datatypes, users can define new datatypes within the datatype classes.

The HDF5 datatype model is very general and flexible:

For common simple purposes, only predefined types will be needed

Datatypes can be combined to create complex structured datatypes

If needed, users can define custom atomic datatypes

Committed datatypes can be shared by datasets or attributes

<!-- NEW PAGE -->

<h4>6.1.2. HDF5 Datatype Model</h4>

<p>The HDF5 Library implements an object-oriented model of datatypes. HDF5 datatypes are organized as a logical set of base types, or datatype classes. Each datatype class defines a format for representing logical values as a sequence of bits. For example the <code>H5T_INTEGER</code> class is a format for representing twos complement integers of various sizes.</p>

<p>A datatype class is defined as a set of one or more datatype properties. A datatype property is a property of the bit string. The datatype properties are defined by the logical model of the datatype class. For example, the integer class (twos complement integers) has properties such as “signed or unsigned”, “length”, and “byte-order”. The float class (IEEE floating point numbers) has these properties, plus “exponent bits”, “exponent sign”, etc.</p>

<p>A datatype is derived from one datatype class: a given datatype has a specific value for the datatype properties defined by the class. For example, for 32-bit signed integers, stored big-endian, the HDF5 datatype is a sub-type of integer with the properties set to <code>signed=1</code>, <code>size=4</code> (bytes), and

`<code>byte-order=BE</code>.</p>`

<p>The HDF5 datatype API (H5T functions) provides methods to create datatypes of different datatype classes, to set the datatype properties of a new datatype, and to discover the datatype properties of an existing datatype.</p>

<p>The datatype for a dataset is stored in the HDF5 file as part of the metadata for the dataset.</p>

<p>A datatype can be shared by more than one dataset in the file if the datatype is saved to the file with a name. This shareable datatype is known as a committed datatype. In the past, this kind of datatype was called a named datatype. </p>

<p>When transferring data (e.g., a read or write), the data elements of the source and destination storage must have compatible types. As a general rule, data elements with the same datatype class are compatible while elements from different datatype classes are not compatible. When transferring data of one datatype to another compatible datatype, the HDF5 Library uses the datatype properties of the source and destination to automatically transform each data element. For example, when reading from data stored as 32-bit signed integers, big-endian into 32-bit signed integers, little-endian, the HDF5 Library will automatically swap the bytes.</p>

<p>Thus, data transfer operations (<code>H5Dread</code>, <code>H5Dwrite</code>, <code>H5Aread</code>, <code>H5Awrite</code>) require a datatype for both the source and the destination.</p>

<!-- NEW PAGE -->

```

<table width="500" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td></tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 2. The datatype model</b>
      <hr color="green" size="3"/></td></tr>
</table>
<br />

```

The HDF5 Library defines a set of predefined datatypes, corresponding to commonly used storage formats, such as two's complement integers, IEEE Floating point numbers, etc., 4- and 8-byte sizes, big-endian and little-endian byte orders. In addition, a user can derive types with custom values for the properties. For example, a user program may create a datatype to describe a 6-bit integer, or a 600-bit floating point number.

In addition to atomic datatypes, the HDF5 Library supports composite datatypes. A composite datatype is an aggregation of one or more datatypes. Each class of composite datatypes has properties that describe the organization of the composite datatype. See the figure below. Composite datatypes include:

- Compound datatypes: structured records
- Array: a multidimensional array of a datatype

- Variable-length: a one-dimensional array of a datatype

<!-- NEW PAGE -->

--

| |
 |

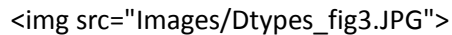
| --- |
| |
 |

Figure 3. Composite datatypes

6.1.2.1. Datatype Classes and Properties

The figure below shows the HDF5 datatype classes. Each class is defined to have a set of properties which describe the layout of the data element and the interpretation of the bits. The table below lists the properties for the datatype classes.

<!-- NEW PAGE -->

--

| |

```

        <td align="center">
        <hr color="green" size="3"/>
        
        </td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left" >
        <b>Figure 4. Datatype classes</b>
        <hr color="green" size="3"/></td></tr>
</table>
<br />
<br />
<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center" cellpadding="0">
    <tr valign="bottom">
        <td colspan="7" align="left" valign="bottom">
            <b>Table 1. Datatype classes and their properties</b></td>
        </tr>
<tr><td colspan="7"><hr color="green" size="3" /></td></tr>
<tr valign="top">
    <td width="20%"><b>Class</b></td>
    <td width="2%">&nbsp;</td>
    <td width="18%"><b>Description</b></td>
    <td width="2%">&nbsp;</td>
    <td width="28%"><b>Properties</b></td>
    <td width="2%">&nbsp;</td>
    <td width="28%"><b>Notes</b></td>
</tr>
<tr><td colspan="7"><hr color="green" size="1" /></td></tr>
<tr valign="top">

```

Integer		Twos complement integers		Size (bytes), precision (bits), offset (bits), pad, byte order, signed/unsigned
<hr/>				
Float		Floating Point numbers		Size (bytes), precision (bits), offset (bits), pad, byte order, sign position, exponent position, exponent size (bits), exponent sign, exponent bias, mantissa position, mantissa (size) bits, mantissa sign, mantissa normalization, internal padding
				See IEEE 754 for a definition of these properties. These properties describe non-IEEE 754 floating point formats as well.
<hr/>				
Character		Array of 1-byte character encoding		Size (characters), Character set, byte order, pad/no pad, pad character

```

<td>&nbsp;</td>
<td>Currently, ASCII and UTF-8 are supported.</td>
</tr>
<tr><td colspan="7"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Bitfield</td>
  <td>&nbsp;</td>
  <td>String of bits</td>
  <td>&nbsp;</td>
  <td>Size (bytes), precision (bits), offset (bits),
  pad, byte order</td>
  <td>&nbsp;</td>
  <td>A sequence of bit values packed into one or more bytes.</td>
</tr>
<tr><td colspan="7"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Opaque</td>
  <td>&nbsp;</td>
  <td>Uninterpreted data</td>
  <td>&nbsp;</td>
  <td>Size (bytes), precision (bits), offset (bits),
  pad, byte order, tag</td>
  <td>&nbsp;</td>
  <td>A sequence of bytes, stored and retrieved as a block. The
  &lsquo;tag&rsquo; is a string that can be used to label
  the value.</td>
</tr>
<tr><td colspan="7"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Enumeration</td>
  <td>&nbsp;</td>

```

<p>A list of discrete values, with symbolic names in the form of strings.</p>
<p>Number of elements, element names, element values</p>
<p>Enumeration is a list of pairs, (name, value). The name is a string, the value is an unsigned integer.</p>

Reference						
Reference to object or region within the HDF5 file						
See the Reference API, H5R						

Array						
Array (1-4 dimensions) of data elements						
Number of dimensions, dimension sizes, base datatype						
The array is accessed atomically: no selection or sub-setting.						

!-- NEW PAGE -->

--	--	--	--	--	--	--

Variable-length	
A variable-length 1-dimensional array of data data elements	
Current size, base type	
<hr/>	
Compound	
A Datatype of a sequence of Datatypes	
Number of members, member names, member types, member offset, member class, member size, byte order	
<hr/>	

6.1.2.2. Predefined Datatypes

The HDF5 library predefines a modest number of commonly used datatypes.

These types have standard symbolic names of the form

`H5T_arch_base` where `arch` is an architecture name and `base` is a programming type name (Table 2). New types can

be derived from the predefined types by copying the predefined type (see `H5Tcopy()`) and then modifying the result.

The base name of most types consists of a letter to indicate the class (Table 3), a precision in bits, and an indication of the byte order (Table 4).

Table 5 shows examples of predefined datatypes.

The full list can be found in the “HDF5 Predefined Datatypes” section of the [HDF5 Reference Manual](http://www.hdfgroup.org/hdf/5.0/RM/RM_H5Front.html).

`<!-- editingComment`

`[[`
 Link to ../PredefDTypes.html
`]] `

`-->`

`
`

`<!-- NEW PAGE -->`

`<table width="600" cellspacing="0" align="center" cellpadding="0">`

`<tr valign="bottom">`

`<td colspan="2" align="left" valign="bottom">`

`Table 2. Architectures used in predefined datatypes</td>`

`</tr>`

`<tr><td colspan="2"><hr color="green" size="3" /></td></tr>`

`<tr valign="top">`

`<td width="20%">`

`Architecture
Name</td>`

`<td width="80%">`

`Description</td>`

`<tr><td colspan="2"><hr color="green" size="1" /></td></tr>`

`<tr valign="top">`

<p><code>IEEE</code></p> <p>IEEE-754 standard floating point types in various byte orders.</p>	
<hr/>	
<p><code>STD</code></p> <p>This is an architecture that contains semi-standard datatypes like signed two's complement integers, unsigned integers, and bitfields in various byte orders.</p>	
<hr/>	
<p><code>C</code> <code>FORTRAN</code></p> <p>Types which are specific to the C or Fortran programming languages are defined in these architectures. For instance, <code>H5T_C_S1</code> defines a base string type with null termination which can be used to derive string types of other lengths.</p>	
<hr/>	
<p><code>NATIVE</code></p> <p>This architecture contains C-like datatypes for the machine on which the library was compiled. The types were actually defined by running the <code>H5detect</code> program when the library was compiled. In order to be portable, applications should almost always use this architecture to describe things in memory.</p>	
<hr/>	

```

<tr valign="top">
  <td><code>CRAY</code> </td>
  <td>Cray architectures. These are
word-addressable, big-endian systems with non-IEEE floating point.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>INTEL</code> </td>
  <td>All Intel and compatible CPU's
including 80286, 80386, 80486, Pentium, Pentium-Pro, and Pentium-II.
These are little-endian systems with IEEE floating-point.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>MIPS</code> </td>
  <td>All MIPS CPU's commonly used in
SGI systems. These are big-endian systems with IEEE floating-point.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>ALPHA</code> </td>
  <td>All DEC Alpha CPU's,
little-endian systems with IEEE floating-point.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<table width="200" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="2" align="left" valign="bottom">
      <b>Table 3. Base types</b></td>
    </tr>
  <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
  <tr valign="top">
    <td width="50">B</td>
    <td width="150">Bitfield</td>
  </tr>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td>F</td>
    <td>Floating point</td>
  </tr>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td>I</td>
    <td>Signed integer</td>
  </tr>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td>R</td>
    <td>References</td>
  </tr>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td>S</td>
    <td>Character string</td>
  </tr>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>

```

```

<tr valign="top">
  <td>U</td>
  <td>Unsigned integer</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

<!-- NEW PAGE -->
<table width="200" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="2" align="left" valign="bottom">
      <b>Table 4. Byte order</b></td>
    </tr>
    <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
    <tr valign="top">
      <td width="50">BE</td>
      <td width="150">Big-endian</td>
    </tr>
    <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
    <tr valign="top">
      <td>LE</td>
      <td>Little-endian</td>
    </tr>
    <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
  </table>
<br />
<br />

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="2" align="left" valign="bottom">
      <b>Table 5. Some predefined datatypes</b></td>
    </tr>
  <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
  <tr valign="top">
    <td width="25%">
      <b>Example</b></td>
    <td width="75%">
      <b>Description</b></td>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td><code>H5T_IEEE_F64LE</code> </td>
    <td>Eight-byte, little-endian, IEEE floating-point</td>
  </tr>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td><code>H5T_IEEE_F32BE</code> </td>
    <td>Four-byte, big-endian, IEEE floating point</td>
  </tr>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td><code>H5T_STD_I32LE</code> </td>
    <td>Four-byte, little-endian, signed two's complement
      integer</td>
  </tr>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">

```

<code>H5T_STD_U16BE</code>	Two-byte, big-endian, unsigned integer
<hr/>	
<code>H5T_C_S1</code>	One-byte, null-terminated string of eight-bit characters
<hr/>	
<code>H5T_INTEL_B64</code>	Eight-byte bit field on an Intel CPU
<hr/>	
<code>H5T_CRAY_F64</code>	Eight-byte Cray floating point
<hr/>	
<code>H5T_STD_ROBJ</code>	Reference to an entire object in a file
<hr/>	

The HDF5 Library predefines a set of `NATIVE` datatypes which are similar to C type names. The native types are set to be an alias for the appropriate HDF5 datatype for each platform. For example, `H5T_NATIVE_INT` corresponds to a C `int` type. On an Intel based PC, this type is the same as `H5T_STD_I32LE`, while on a MIPS system this would be equivalent to `H5T_STD_I32BE`. Table 6 shows examples of `NATIVE` types and corresponding C types for a common 32-bit workstation.

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="2" align="left" valign="bottom">

Table 6. Native and 32-bit C datatypes</td>

</tr>

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td>Example</td>

<td>Corresponding C Type</td>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>H5T_NATIVE_CHAR</code> </td>

<td>char</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>H5T_NATIVE_SCHAR</code> </td>

<td>signed char</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">


```

    <td><code>H5T_NATIVE_UCHAR</code> </td>
    <td>unsigned char</td>
  </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5T_NATIVE_SHORT</code> </td>
  <td>short</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5T_NATIVE_USHORT</code> </td>
  <td>unsigned short</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5T_NATIVE_INT</code> </td>
  <td>int</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5T_NATIVE_UINT</code> </td>
  <td>unsigned</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5T_NATIVE_LONG</code> </td>
  <td>long</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5T_NATIVE_ULONG</code> </td>

```

```

        <td>unsigned long</td>
    </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_LLONG</code> </td>
    <td>long long</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_ULLONG</code> </td>
    <td>unsigned long long</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_FLOAT</code> </td>
    <td>float</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_DOUBLE</code> </td>
    <td>double</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_LDOUBLE</code> </td>
    <td>long double</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_HSIZE</code> </td>
    <td>hsize_t</td>

```

```

    </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_HSSIZE</code> </td>
    <td>hssize_t</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_HERR</code> </td>
    <td>herr_t</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_HBOOL</code> </td>
    <td>hbool_t</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_B8</code> </td>
    <td>8-bit unsigned integer or 8-bit buffer in memory</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_B16</code> </td>
    <td>16-bit unsigned integer or 16-bit buffer in memory</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>H5T_NATIVE_B32</code> </td>
    <td>32-bit unsigned integer or 32-bit buffer in memory</td>
</tr>

```

```

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5T_NATIVE_B64</code> </td>
  <td>64-bit unsigned integer or 64-bit buffer in memory</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="DtypesUsed">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>

```

```

<!-- NEW PAGE -->
<a name="DtypesUsed">
<h3 class=pagebefore>6.2. How Datatypes are Used</h3>
</a>

```

```

<h4>6.2.1. The Datatype Object and the HDF5 Datatype API</h4>

```

```

<p>The HDF5 Library manages datatypes as objects. The HDF5 datatype API

```

manipulates the datatype objects through C function calls. New datatypes can be created from scratch or copied from existing datatypes. When a datatype is no longer needed its resources should be released by calling `H5Tclose()`. </p>

The datatype object is used in several roles in the HDF5 data model and library. Essentially, a datatype is used whenever the format of data elements is needed. There are four major uses of datatypes in the HDF5 Library: at dataset creation, during data transfers, when discovering the contents of a file, and for specifying user-defined datatypes. See the table below.

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="2" align="left" valign="bottom">
      <b>Table 7. Datatype uses</b></td>
    </tr>
    <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
    <tr valign="top">
      <td>
        <b>Use</b></td>
      <td>
        <b>Description</b></td>
    <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
    <tr valign="top">
      <td>Dataset creation</td>
      <td>The datatype of the data elements must be
        declared when the dataset is created.</td>
    </tr>
    <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```

<tr valign="top">
  <td>Data transfer</td>
  <td>The datatype (format) of the data elements
  must be defined for both the source and destination.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Discovery</td>
  <td>The datatype of a dataset can be
  interrogated to retrieve a complete description of the storage layout.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Creating user-defined datatypes</td>
  <td>Users can define their own datatypes by
  creating datatype objects and setting their properties.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

<h4>6.2.2. Dataset Creation</h4>

<p>All the data elements of a dataset have the same datatype. When a dataset is created, the datatype for the data elements must be specified. The datatype of a dataset can never be changed. The example below shows the use of a datatype to create a dataset called “/dset”. In this example, the dataset will be stored as 32-bit signed integers in big-endian order.</p>

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
</td>
<td align="left">
<pre>
hid_t dt;
dt = H5Tcopy(H5T_STD_I32BE);
dataset_id = H5Dcreate(file_id, &ldquo;/dset&rdquo;, dt, dataspace_id,
  H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);</pre>
</td>
</tr>
<tr>
<td align="left">
<hr color="green" size="1" />
</td>
<td align="left">
<b>Example 1. Using a datatype to create a dataset </b>
<hr color="green" size="3"/>
</td>
</tr>
</table>
<br />

```

6.2.3. Data Transfer (Read and Write)

Probably the most common use of datatypes is to write or read data from a dataset or attribute. In these operations, each data element is transferred from the source to the destination (possibly rearranging the order of the elements). Since the source and destination do not need to be identical (i.e., one is disk and the other is memory) the transfer requires both the format of the source element and the destination element. Therefore, data transfers use two datatype objects, for the source and destination.

When data is written, the source is memory and the destination is disk

(file). The memory datatype describes the format of the data element in the machine memory, and the file datatype describes the desired format of the data element on disk. Similarly, when reading, the source datatype describes the format of the data element on disk, and the destination datatype describes the format in memory.

In the most common cases, the file datatype is the datatype specified when the dataset was created, and the memory datatype should be the appropriate NATIVE type.

The examples below show samples of writing data to and reading data from a dataset. The data in memory is declared C type `int`, and the datatype `H5T_NATIVE_INT` corresponds to this type. The datatype of the dataset should be of datatype class `H5T_INTEGER`.

--	--

--	--

--	--

--	--

```


```

```

int dset_data[DATA_SIZE];

```

```

status = H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
    H5P_DEFAULT, dset_data);

```

--	--

--	--

--	--

--	--

--	--


```

        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />
<br />

```

```

<table width="600" cellpadding="0" align="center">
    <tr valign="top">
        <td align="left">
            <hr color="green" size="3"/>
        <pre>
            int dset_data[DATA_SIZE];

            status = H5Dread(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
                H5P_DEFAULT, dset_data);</pre></td>
    </tr>
    <tr><td><hr color="green" size="1" /></td></tr>
    <tr valign="top">
        <td align="left">
            <b>Example 3. Reading from a dataset</b>
            <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

<h4>6.2.4. Discovery of Data Format</h4>

<p>The HDF5 Library enables a program to determine the datatype class and properties for any datatype. In order to discover the storage format of data in a dataset, the datatype is obtained, and the properties are determined

by queries to the datatype object. The example below shows code that analyzes the datatype for an integer and prints out a description of its storage properties (byte order, signed, size.)

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

```
switch (H5Tget_class(type)) {
case H5T_INTEGER:
ord = H5Tget_order(type);
sgn = H5Tget_sign(type);
printf("Integer ByteOrder= ");
switch (ord) {
case H5T_ORDER_LE:
printf("LE");
break;
case H5T_ORDER_BE:
printf("BE");
break;
}
printf(" Sign= ");
switch (sgn) {
case H5T_SGN_NONE:
printf("false");
break;
case H5T_SGN_2:
printf("true");
break;
```

```

}
printf("&ldquo; Size= &rdquo;);
sz = H5Tget_size(type);
printf("&ldquo;%d&rdquo;;, sz);
printf("&ldquo;\n&rdquo;);
break;</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td align="left">
<b>Example 4. Discovering datatype properties</b>
<hr color="green" size="3" /></td>
</tr>
</table>
<br />

```

6.2.5. Creating and Using User-defined Datatypes

Most programs will primarily use the predefined datatypes described above, possibly in composite datatypes such as compound or array datatypes. However, the HDF5 datatype model is extremely general; a user program can define a great variety of atomic datatypes (storage layouts). In particular, the datatype properties can define signed and unsigned integers of any size and byte order, and floating point numbers with different formats, size, and byte order. The HDF5 datatype API provides methods to set these properties.

User-defined types can be used to define the layout of data in memory, e.g., to match some platform specific number format or application defined bit-field. The user-defined type can also describe data in the file, e.g., some application-defined format. The user-defined types can be translated to and from standard types of the same class,

as described above.</p>

<!-- editingComment

[[[

<p>{Simple programming example...}

]]]

-->

<SCRIPT language="JavaScript">

<!--

document.writeln ("

<div align=right>

(Top)

</div>

");

-->

</SCRIPT>

<!-- NEW PAGE -->

<h3 class=pagebefore>6.3. Datatype (H5T) Function Summaries</h3>

<p>Functions that can be used with datatypes (H5T functions) and property

list functions that can be used with datatypes (H5P functions) are listed

below.</p>


```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="3" align="left" valign="bottom">
      <b>Function Listing 1. General datatype operations
    </b></td>
  </tr>
  <tr><td colspan="3"><hr color="green" size="3" /></td></tr>
  <tr valign="top"><td width="25%"><b>C Function<br />Fortran Function</b></td>
    <td width="2%">&nbsp;</td>
    <td width="73%"><b>Purpose</b></td>

  <tr><td colspan="3"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td>
      <code>H5Tcreate<br />h5tcreate_f</code>
    </td><td>&nbsp;</td>
    <td>
      Creates a new datatype.
    </td>
  </tr>
  <tr><td colspan="3"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td>
      <code>H5Topen<br />h5topen_f</code>
    </td><td>&nbsp;</td>
    <td>
      Opens a committed datatype. The C function is a
      macro: see <a href=" ../RM/APICompatMacros.html">&ldquo;API
      Compatibility Macros in HDF5.&rdquo;</a>
    </td>
  </tr>

```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Tcommit<br />h5tcommit_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Commits a transient datatype to a file. The datatype is now a

committed datatype. The C function is a

macro: see ["API](http://www.hdfgroup.org/HDF5/doc/RM/APICompatMacros.html)

Compatibility Macros in HDF5."["API](#)

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Tcommit_anon<br />h5tcommit_anon_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Commits a transient datatype to a file. The datatype is now a

committed datatype, but it is not linked into the file structure.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Tcommitted<br />h5tcommitted_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Determines whether a datatype is a committed or a transient type.

```
</td>
```

```

    </tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Tcopy<br />h5tcopy_f</code>
    </td><td>&nbsp;</td>
    <td>
        Copies an existing datatype.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Tequal<br />h5tequal_f</code>
    </td><td>&nbsp;</td>
    <td>
        Determines whether two datatype identifiers refer to the same datatype.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Tlock<br />(none)</code>
    </td><td>&nbsp;</td>
    <td>
        Locks a datatype.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>

```

```
<code>H5Tget_class<br />h5tget_class_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Returns the datatype class identifier.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Tget_create_plist<br />h5tget_create_plist_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Returns a copy of a datatype creation property list.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Tget_size<br />h5tget_size_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Returns the size of a datatype.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Tget_super<br />h5tget_super_f</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Returns the base datatype from which a datatype is derived.


```

</td>
</tr>
<!-- NEW PAGE -->
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Tget_native_type<br />h5tget_native_type_f</code>
  </td><td>&nbsp;</td>
  <td>
    Returns the native datatype of a specified datatype.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Tdetect_class<br />(none)</code>
  </td><td>&nbsp;</td>
  <td>
    Determines whether a datatype is of the given datatype class.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Tget_order<br />h5tget_order_f</code>
  </td><td>&nbsp;</td>
  <td>
    Returns the byte order of a datatype.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

```

<div data-bbox="240 243 298 268" data-label="Text"><code><td></code></div> <div data-bbox="240 300 837 327" data-label="Text"><code><code>H5Tset_order
h5tset_order_f</code></code></div> <div data-bbox="240 354 513 382" data-label="Text"><code></td><td>&nbsp;</td></code></div> <div data-bbox="240 411 298 436" data-label="Text"><code><td></code></div> <div data-bbox="240 468 675 495" data-label="Text">Sets the byte ordering of a datatype.</div> <div data-bbox="240 522 308 550" data-label="Text"><code></td></code></div> <div data-bbox="240 577 303 604" data-label="Text"><code></tr></code></div>	<hr/>	
<div data-bbox="240 745 298 770" data-label="Text"><code><td></code></div> <div data-bbox="240 802 781 829" data-label="Text"><code><code>H5Tdecode
h5tdecode_f</code></code></div> <div data-bbox="240 856 513 884" data-label="Text"><code></td><td>&nbsp;</td></code></div> <div data-bbox="240 911 298 938" data-label="Text"><code><td></code></div> <div data-bbox="240 970 1008 1052" data-label="Text">Decode a binary object description of datatype and return a new object identifier.</div> <div data-bbox="240 1079 308 1106" data-label="Text"><code></td></code></div> <div data-bbox="240 1134 303 1161" data-label="Text"><code></tr></code></div>	<hr/>	
<div data-bbox="240 1302 298 1327" data-label="Text"><code><td></code></div> <div data-bbox="240 1358 756 1386" data-label="Text"><code><code>H5Tencode
h5tencode</code></code></div> <div data-bbox="240 1413 513 1440" data-label="Text"><code></td><td>&nbsp;</td></code></div> <div data-bbox="240 1467 298 1495" data-label="Text"><code><td></code></div> <div data-bbox="240 1526 919 1554" data-label="Text">Encode a datatype object description into a binary buffer.</div> <div data-bbox="240 1581 308 1608" data-label="Text"><code></td></code></div> <div data-bbox="240 1635 303 1663" data-label="Text"><code></tr></code></div>	<hr/>	
<div data-bbox="240 1803 298 1829" data-label="Text"><code><td></code></div> <div data-bbox="240 1860 727 1887" data-label="Text"><code><code>H5Tclose
h5tclose_f</code></code></div>		

```

</td><td>&nbsp;</td>
<td>
Releases a datatype.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="3" align="left" valign="bottom">
      <b>Function Listing 2. Conversion functions
    </b></td>
  </tr>
  <tr><td colspan="3"><hr color="green" size="3" /></td></tr>
  <tr valign="top">
    <td>
      <b>C Function<br />Fortran Function</b>
    </td><td>&nbsp;</td>
    <td>
      <b>Purpose</b>
    </td>
  </tr>

```

<hr/>		
-------	--	--

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<hr/>		
-------	--	--

<table> <tr> <td> </td> </tr> </table>	

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<hr/>		
-------	--	--

<table> <tr> <td> </td> </tr> </table>	

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<hr/>		
-------	--	--

<table> <tr> <td> </td> </tr> </table>	

<table> <tr> <td> </td> <td> </td> </tr> </table>		

<table> <tr> <td> </td> <td> </td> </tr> </table>		

```

</td><td>&nbsp;</td>
<td>
Registers a conversion function.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5Tunregister<br />(none)</code>
</td><td>&nbsp;</td>
<td>
Removes a conversion function from all conversion paths.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<!-- NEW PAGE -->

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
<tr valign="bottom">
<td colspan="3" align="left" valign="bottom">
<b>Function Listing 3. Atomic datatype properties
</b></td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
<tr valign="top">
<td>

```

C Function Fortran Function	
Purpose	

	<code>H5Tset_size</code> <code>h5tset_size_f</code>	
	Sets the total size for an atomic datatype.	

	<code>H5Tget_precision</code> <code>h5tget_precision_f</code>	
	Returns the precision of an atomic datatype.	

	<code>H5Tset_precision</code> <code>h5tset_precision_f</code>	
	Sets the precision of an atomic datatype.	

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tget_offset
h5tget_offset_f</code>

</td><td> </td>

<td>

Retrieves the bit offset of the first significant bit.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tset_offset
h5tset_offset_f</code>

</td><td> </td>

<td>

Sets the bit offset of the first significant bit.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tget_pad
h5tget_pad_f</code>

</td><td> </td>

<td>

Retrieves the padding type of the least and most-significant bit padding.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

```

<tr valign="top">
  <td>
    <code>H5Tset_pad<br />h5tset_pad_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets the least and most-significant bits padding types.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Tget_sign<br />h5tget_sign_f</code>
  </td><td>&nbsp;</td>
  <td>
    Retrieves the sign type for an integer type.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Tset_sign<br />h5tset_sign_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets the sign property for an integer type.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Tget_fields<br />h5tget_fields_f</code>
  </td><td>&nbsp;</td>

```


<td>

Retrieves floating point datatype bit field information.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tset_fields
h5tset_fields_f</code>

</td><td> </td>

<td>

Sets locations and sizes of floating point bit fields.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tget_ebias
h5tget_ebias_f</code>

</td><td> </td>

<td>

Retrieves the exponent bias of a floating-point type.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tset_ebias
h5tset_ebias_f</code>

</td><td> </td>

<td>

Sets the exponent bias of a floating-point type.

</td>

</tr>

--

--

<pre>H5Tget_norm</pre>

<pre>h5tget_norm_f</pre>

--

Retrieves mantissa normalization of a floating-point datatype.

--

--

--

--

<pre>H5Tset_norm</pre>

<pre>h5tset_norm_f</pre>

--

Sets the mantissa normalization of a floating-point datatype.

--

--

<!-- NEW PAGE -->

--

--

<pre>H5Tget_inpad</pre>

<pre>h5tget_inpad_f</pre>

--

Retrieves the internal padding type for unused bits in floating-point datatypes.

--

--

--

<div data-bbox="240 182 839 327"> <div data-bbox="240 182 300 214"><td></div> <div data-bbox="240 237 839 273"><code><code>H5Tset_inpad
h5tset_inpad_f</code></code></div> <div data-bbox="240 294 513 327"></td><td>&nbsp;</td></div> </div>	<div data-bbox="240 350 703 436"> <div data-bbox="240 350 300 382"><td></div> <div data-bbox="240 405 703 436">Fills unused internal floating point bits.</div> </div>
<div data-bbox="240 459 310 491"></td></div> <div data-bbox="240 514 305 546"></tr></div>	
<div data-bbox="212 569 943 604"><tr><td colspan="3"><hr color="green" size="1" /></td></tr></div>	
<div data-bbox="212 627 420 659"><tr valign="top"></div>	
<div data-bbox="240 682 808 827"> <div data-bbox="240 682 300 714"><td></div> <div data-bbox="240 737 808 772"><code><code>H5Tget_cset
h5tget_cset_f</code></code></div> <div data-bbox="240 793 513 827"></td><td>&nbsp;</td></div> </div>	<div data-bbox="240 850 862 936"> <div data-bbox="240 850 300 882"><td></div> <div data-bbox="240 905 862 936">Retrieves the character set type of a string datatype.</div> </div>
<div data-bbox="240 959 310 991"></td></div> <div data-bbox="240 1014 305 1045"></tr></div>	
<div data-bbox="212 1068 943 1104"><tr><td colspan="3"><hr color="green" size="1" /></td></tr></div>	
<div data-bbox="212 1127 420 1159"><tr valign="top"></div>	
<div data-bbox="240 1182 803 1327"> <div data-bbox="240 1182 300 1213"><td></div> <div data-bbox="240 1236 803 1272"><code><code>H5Tset_cset
h5tset_cset_f</code></code></div> <div data-bbox="240 1293 513 1327"></td><td>&nbsp;</td></div> </div>	<div data-bbox="240 1350 591 1436"> <div data-bbox="240 1350 300 1381"><td></div> <div data-bbox="240 1404 591 1436">Sets character set to be used.</div> </div>
<div data-bbox="240 1459 310 1491"></td></div> <div data-bbox="240 1514 305 1545"></tr></div>	
<div data-bbox="212 1568 943 1604"><tr><td colspan="3"><hr color="green" size="1" /></td></tr></div>	
<div data-bbox="212 1627 420 1659"><tr valign="top"></div>	
<div data-bbox="240 1682 862 1827"> <div data-bbox="240 1682 300 1713"><td></div> <div data-bbox="240 1736 862 1772"><code><code>H5Tget_strpad
h5tget_strpad_f</code></code></div> <div data-bbox="240 1793 513 1827"></td><td>&nbsp;</td></div> </div>	<div data-bbox="240 1850 300 1881"><td></div>

Retrieves the storage mechanism for a string datatype.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tset_strpad
h5tset_strpad_f</code>

</td><td> </td>

<td>

Defines the storage mechanism for character strings.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

</table>

<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="3" align="left" valign="bottom">

Function Listing 4. Enumeration datatypes

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td>

C Function
Fortran Function

</td><td> </td>

<td>

Purpose

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tenum_create
h5tenum_create_f</code>

</td><td> </td>

<td>

Creates a new enumeration datatype.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tenum_insert
h5tenum_insert_f</code>

</td><td> </td>

<td>

Inserts a new enumeration datatype member.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tenum_nameof
h5tenum_nameof_f</code>

</td><td> </td>

<td>

Returns the symbol name corresponding to a specified member of an enumeration datatype.

</td>

<code><td></code> <code><code>H5Tget_member_name
h5tget_member_name_f</code></code> <code></td><td>&nbsp;</td></code> <code><td></code> Retrieves the name of a compound or enumeration datatype member. <code></td></code> <code></tr></code>	<hr style="color: green; width: 100%;"/>		
<code><td></code> <code><code>H5Tget_member_index
(none)</code></code> <code></td><td>&nbsp;</td></code> <code><td></code> Retrieves the index of a compound or enumeration datatype member. <code></td></code> <code></tr></code>	<hr style="color: green; width: 100%;"/>		

Function Listing 5. Compound datatype properties		
---------------------------------------------------------	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

</td><td> </td>

<td>

Retrieves the name of a compound or enumeration datatype member.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tget_member_index
h5tget_member_index_f</code>

</td><td> </td>

<td>

Retrieves the index of a compound or enumeration datatype member.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tget_member_offset
h5tget_member_offset_f</code>

</td><td> </td>

<td>

Retrieves the offset of a field of a compound datatype.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Tget_member_type
h5tget_member_type_f</code>

</td><td> </td>

<td>

Returns the datatype of the specified member.

</td>

```

</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Tinsert<br />h5tinsert_f</code>
  </td><td>&nbsp;</td>
  <td>
    Adds a new member to a compound datatype.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Tpack<br />h5tpack_f</code>
  </td><td>&nbsp;</td>
  <td>
    Recursively removes padding from within a compound datatype.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="3" align="left" valign="bottom">

```

Function Listing 6. Array datatypes

C Function

Purpose

`H5Tarray_create`

Creates an array datatype object. The C function is a

macro: see <http://www.hdfgroup.org/hdf/5.0/compatmacros.html>

Compatibility Macros in HDF5.

`H5Tget_array_ndims`

Returns the rank of an array datatype.

```

    </tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Tget_array_dims<br />h5tget_array_dims_f</code>
    </td><td>&nbsp;</td>
    <td>
        Returns sizes of array dimensions and dimension permutations.
        The C function is a
        macro: see <a href=" ../RM/APICompatMacros.html">&ldquo;API
        Compatibility Macros in HDF5.&rdquo;</a>
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<!-- NEW PAGE -->

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
    <tr valign="bottom">
        <td colspan="3" align="left" valign="bottom">
            <b>Function Listing 7. Variable-length datatypes
        </b></td>
    </tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

```

C Function Fortran Function Purpose	
<hr/>	
<code>H5Tvlen_create</code> Creates a new variable-length datatype.	
<code>H5Tis_variable_str</code> Determines whether datatype is a variable-length string.	
<hr/>	

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
  <tr valign="bottom">
```

```
    <td colspan="3" align="left" valign="bottom">
```

```
      <b>Function Listing 8. Opaque datatypes
```

```
    </b></td>
```

```
  </tr>
```

```
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
  <td>
```

```
    <b>C Function<br />Fortran Function</b>
```

```
  </td><td>&nbsp;</td>
```

```
  <td>
```

```
    <b>Purpose</b>
```

```
  </td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
  <td>
```

```
    <code>H5Tset_tag<br />h5tset_tag_f</code>
```

```
  </td><td>&nbsp;</td>
```

```
  <td>
```

```
    Tags an opaque datatype.
```

```
  </td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
  <td>
```

```

<code>H5Tget_tag<br />h5tget_tag_f</code>
</td><td>&nbsp;</td>
<td>
Gets the tag associated with an opaque datatype.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="3" align="left" valign="bottom">
      <b>Function Listing 9. Conversions between datatype and text
    </b></td>
  </tr>
  <tr><td colspan="3"><hr color="green" size="3" /></td></tr>
  <tr valign="top">
    <td>
      <b>C Function<br />Fortran Function</b>
    </td><td>&nbsp;</td>
    <td>
      <b>Purpose</b>
    </td>
  </tr>
  <tr><td colspan="3"><hr color="green" size="1" /></td></tr>
  <tr valign="top">

```

```

<td>
<code>H5LTtext_to_dtype<br />(none)</code>
</td><td>&nbsp;</td>
<td>
Creates a datatype from a text description.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5LTdtype_to_text<br />(none)</code>
</td><td>&nbsp;</td>
<td>
Generates a text description of a datatype.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<!-- NEW PAGE -->
<table width="600" cellspacing="0" align="center" cellpadding="0">
<tr valign="bottom">
<td colspan="3" align="left" valign="bottom">
<b>Function Listing 10. Datatype creation property list
functions (H5P)</b></td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

```


C Function Fortran Function Purpose	
<hr/>	
H5Pset_char_encoding Sets the character encoding used to encode a string. Use to set ASCII or UTF-8 character encoding for object names.	
H5Pget_char_encoding Retrieves the character encoding used to create a string.	

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
  <tr valign="bottom">
```

```
    <td colspan="3" align="left" valign="bottom">
```

```
      <b>Function Listing 11. Datatype access property list
```

```
      functions (H5P)</b></td>
```

```
    </tr>
```

```
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
  <td>
```

```
    <b>C Function<br />Fortran Function</b>
```

```
  </td><td>&nbsp;</td>
```

```
  <td>
```

```
    <b>Purpose</b>
```

```
  </td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
  <td>
```

```
    <code>H5Pset_type_conv_cb<br />(none)</code>
```

```
  </td><td>&nbsp;</td>
```

```
  <td>
```

```
    Sets user-defined datatype conversion callback function.
```

```
  </td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
  <td>
```

<pre> <code>H5Pget_type_conv_cb
(none)</code> </td><td>&nbsp;</td> <td> Gets user-defined datatype conversion callback function. </td> </tr> <tr><td colspan="3"><hr color="green" size="3" /></td></tr> </table>

 </pre>

```

<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="Pmodel">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>

<!-- NEW PAGE -->
<a name="Pmodel">
<h3 class=pagebefore>6.4. The Programming Model</h3>

```

6.4.1. Introduction

The HDF5 Library implements an object-oriented model of datatypes. HDF5 datatypes are organized as a logical set of base types, or datatype classes. The HDF5 Library manages datatypes as objects. The HDF5 datatype API manipulates the datatype objects through C function calls. The figure below shows the abstract view of the datatype object. The table below shows the methods (C functions) that operate on datatype objects. New datatypes can be created from scratch or copied from existing datatypes.

```
<table width="550" cellspacing="0" align="center">

  <tr valign="top">

    <td align="center">

      <hr color="green" size="3"/>

    </td>

  </tr>

  <tr>

    <td align="center" border="1">

      <tr>

        <td valign="middle" align="center"><code>Datatype</code></td>

      </tr>

      <tr>

        <td valign="middle" align="left">

          <code>&nbsp;size:int?<br />

          &nbsp;byteOrder:BOtype</code></td>

        </tr>

      <tr>

        <td valign="middle" align="left">
```

```

        <code>&nbsp;open(hid_t loc, char *, name):return hid_t<br />
        &nbsp;copy(hid_t tid) return hid_t<br />
        &nbsp;create(hid_class_t&nbsp;cls,&nbsp;size_t&nbsp;size)
        &nbsp;return&nbsp;hid_t&nbsp;</code>

```

```

    </td>

```

```

</tr>

```

```

</table>

```

```

</td></tr>

```

```

<tr><td><hr color="green" size="1" /></td></tr>

```

```

<tr valign="top">

```

```

    <td align="left" >

```

```

        <b>Figure 5. The datatype object</b>

```

```

        <hr color="green" size="3"/></td></tr>

```

```

</table>

```

```

<br />

```

```

<br />

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">

```

```

    <tr valign="bottom">

```

```

        <td colspan="2" align="left" valign="bottom">

```

```

        <b>Table 8. General operations on datatype objects</b></td>

```

```

    </tr>

```

```

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

```

```

<tr valign="top">

```

```

    <td width="50%"><b>API Function</b></td>

```

```

    <td width="50%"><b>Description</b></td>

```

```

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

```

```

<tr valign="top">

```

```

    <td><code>hid_t H5Tcreate (H5T_class_t

```

```

    <i>class</i>, size_t <i>size</i>)</code></td>

```

Create a new datatype object of datatype class `class`. The following datatype classes are supported with this function:

- `H5T_COMPOUND`
- `H5T_OPAQUE`
- `H5T_ENUM`

Other datatypes are created with `H5Tcopy()`.

<tr valign="top">

`hid_t H5Tcopy (hid_t type)`

Obtain a modifiable transient datatype which is a copy of *type*. If *type* is a dataset identifier then the type returned is a modifiable transient copy of the datatype of the specified dataset.

<tr valign="top">

`hid_t H5Topen (hid_t location,
 const char *name, H5P_DEFAULT)`

Open a committed datatype. The committed datatype returned by this function is read-only.

<tr valign="top">

`htri_t H5Tequal (hid_t type1,
 hid_t type2)`

Determines if two types are equal.

</tr>

<!-- NEW PAGE -->

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>herr_t H5Tclose (hid_t <i>type</i>)

</code></td>

<td>Releases resources associated with a

datatype obtained from <code>H5Tcopy</code>, <code>H5Topen</code>, or

<code>H5Tcreate</code>. It is illegal to close an

immutable transient datatype (e.g., predefined types).</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>herr_t H5Tcommit (hid_t

<i>location</i>, const char *<i>name</i>, hid_t <i>type</i>,

H5P_DEFAULT, H5P_DEFAULT,
H5P_DEFAULT)</code></td>

<td>Commit a transient datatype (not immutable)

to a file to become a committed datatype. Committed datatypes can be shared.</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>htri_t H5Tcommitted (hid_t

<i>type</i>)</code></td>

<td>Test whether the datatype is

transient or committed (named).</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>herr_t H5Tlock (hid_t

<i>type</i>)</code></td>

<td>Make a transient datatype immutable

(read-only and not closable). Predefined types are locked.

In order to use a datatype, the object must be created

(`H5Tcreate`), or a reference obtained by cloning from an

existing type (`H5Tcopy`), or opened (`H5Topen`).

In addition, a reference to the datatype of a dataset or attribute

can be obtained with `H5Dget_type` or

`H5Aget_type`. For composite datatypes a reference

to the datatype for members or base types can be obtained

(`H5Tget_member_type`, `H5Tget_super`).

When the datatype object is no longer needed, the reference is

discarded with `H5Tclose`.

Two datatype objects can be tested to see if they are the same with

`H5Tequal`. This function returns true if the two datatype

references refer to the same datatype object. However, if two datatype

objects define equivalent datatypes (the same datatype class and

datatype properties), they will not be considered 'equal'.

A datatype can be written to the file as a first class object

(`H5Tcommit`). This is a committed datatype and can be used

in the same way as any other datatype.

6.4.2. Discovery of Datatype Properties

Any HDF5 datatype object can be queried to discover all of its

datatype properties. For each datatype class, there are a set of API functions to retrieve the datatype properties for this class. </p>

6.4.2.1. Properties of Atomic Datatypes

Table 9 lists the functions to discover the properties of atomic datatypes. Table 10 lists the queries relevant to specific numeric types. Table 11 gives the properties for atomic string datatype, and Table 12 gives the property of the opaque datatype.</p>

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="2" align="left" valign="bottom">

Table 9. Functions to discover properties of atomic datatypes</td>

</tr>

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td width="50%">Functions</td>

<td width="50%">Description</td>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>H5T_class_t H5Tget_class (hid_t

<i>type</i></code></td>

<td>The datatype class: <code>H5T_INTEGER,

H5T_FLOAT, H5T_STRING, or H5T_BITFIELD, H5T_OPAQUE,

H5T_COMPOUND, H5T_REFERENCE, H5T_ENUM, H5T_VLEN, H5T_ARRAY</code></td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>size_t H5Tget_size

```

(hid_t <i>type</i></code></td>
<td>The total size of the element in bytes, including padding
which may appear on either side of the actual value.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td><code>H5T_order_t H5Tget_order
(hid_t <i>type</i></code></td>
<td>The byte order describes how the bytes of the datatype are
laid out in memory. If the lowest memory address contains the
least significant byte of the datum then it is
said to be <i>little-endian</i> or <code>H5T_ORDER_LE</code>. If
the bytes are in the opposite order then they are said to be
<i>big-endian</i> or <code>H5T_ORDER_BE.</code></td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td><code>size_t H5Tget_precision
(hid_t <i>type</i></code></td>
<td>The <code>precision</code> property identifies the number
of significant bits of a datatype and the
<code>offset</code> property (defined below) identifies its location.
Some datatypes occupy more bytes than what is needed to store the
value. For instance, a <code>short</code> on a Cray is 32 significant
bits in an eight-byte field.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td><code>int H5Tget_offset (hid_t <i>type</i></code></td>
<td>The <code>offset</code> property defines the bit location
of the least significant bit of a bit

```

field whose length is `precision`.

`herr_t H5Tget_pad`

(`hid_t` *type*, `H5T_pad_t` **lsb*, `H5T_pad_t`

**msb*)

Padding is the bits of a data element

which are not significant as defined by the `precision`

and `offset` properties. Padding in the low-numbered

bits is *lsb* padding and padding in the high-numbered

bits is *msb* padding. Padding bits can be set to zero

(`H5T_PAD_ZERO`) or one (`H5T_PAD_ONE`).

```

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5T_sign_t H5Tget_sign
  (hid_t <i>type</i></code></td>
  <td><b>(INTEGER)</b> Integer data can be signed two's
  complement (<code>H5T_SGN_2</code>)
  or unsigned (<code>H5T_SGN_NONE</code>).</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>herr_t H5Tget_fields
  (hid_t <i>type</i>, size_t *<i>spos</i>, size_t *<i>epos</i>,
  size_t *<i>esize</i>, size_t *<i>mpos</i>,
  size_t *<i>msize</i>)</code> </td>
  <td><b>(FLOAT)</b> A floating-point
  data element has bit fields which are the exponent and mantissa
  as well as a mantissa sign bit. These properties define the
  location (bit position of least significant bit of the field)
  and size (in bits) of each field. The sign bit is always of
  length one and none of the fields are allowed to overlap.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>size_t H5Tget_ebias
  (hid_t <i>type</i>)</code></td>
  <td><b>(FLOAT)</b> The exponent is stored as a non-negative
  value which is <code>ebias</code> larger than the true exponent. </td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5T_norm_t H5Tget_norm

```

(<code>hid_t</code> <i>type</i>)	
(FLOAT) This property describes the normalization method of the mantissa.	
<ul style="list-style-type: none"> <code>H5T_NORM_MSBSET</code>: the mantissa is shifted left (if non-zero) until the first bit after the radix point is set and the exponent is adjusted accordingly. All bits of the mantissa after the radix point are stored. <code>H5T_NORM IMPLIED</code>: the mantissa is shifted left \ (if non-zero) until the first bit after the radix point is set and the exponent is adjusted accordingly. The first bit after the radix point is not stored since it's always set. <code>H5T_NORM_NONE</code>: the fractional part of the mantissa is stored without normalizing it. 	
<hr/>	
<code>H5T_pad_t</code> <code>H5Tget_inpad</code> (<code>hid_t</code> <i>type</i>)	
(FLOAT) If any internal bits (that is, bits between the sign bit, the mantissa field, and the exponent field but within the precision field) are unused, then they will be filled according to the value of this property. The padding can be: <code>H5T_PAD_NONE</code> , <code>H5T_PAD_ZERO</code> or <code>H5T_PAD_ONE</code> .	
<hr/>	

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="2" align="left" valign="bottom">

Table 11. Functions to discover properties of atomic
string datatypes</td>

</tr>

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td width="50%">Functions</td>

<td width="50%">Description</td>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>H5T_cset_t H5Tget_cset

(hid_t type</code></td>

<td>Two character sets are currently

supported: ASCII (<code>H5T_CSET_ASCII</code>) and UTF-8

(<code>H5T_CSET_UTF8</code>).</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>H5T_str_t H5Tget_strpad

(hid_t type</code></td>

<td>The string datatype has a fixed

length, but the string may be shorter than the length. This

property defines the storage mechanism for the left over bytes.

The options are: <code>H5T_STR_NULLTERM</code>,</td>

<code>H5T_STR_NULLPAD</code>, or <code>H5T_STR_SPACEPAD</code>.</td>

</tr>

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

</table>

<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="2" align="left" valign="bottom">

Table 12. Functions to discover properties of atomic opaque
datatypes</td>

</tr>

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td width="50%">Functions</td>

<td width="50%">Description</td>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>char *H5Tget_tag(hid_t type_id)</code></td>

<td>A user-defined string.</td>

</tr>

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

</table>

<h4>6.4.2.2. Properties of Composite Datatypes</h4>

<p>The composite datatype classes can also be analyzed to discover their datatype properties and the datatypes that are members or base types of the composite datatype. The member or base type can, in turn, be analyzed. The table below lists the functions that can access the datatype properties of the different composite datatypes.</p>

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="2" align="left" valign="bottom">

Table 13. Functions to discover properties of composite datatypes

</td>

</tr>

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td width="50%">Functions</td>

<td width="50%">Description</td>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>int H5Tget_nmembers(hid_t type_id)</code></td>

<td>(COMPOUND)The number of fields in the compound datatype.</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>H5T_class_t H5Tget_member_class

(hid_t cdtype_id, unsigned member_no)</code></td>

<td>(COMPOUND) The datatype class of compound datatype member <code>member_no</code>.</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>char * H5Tget_member_name

(hid_t type_id, unsigned field_idx)</code></td>

<td>(COMPOUND) The name of field <code>field_idx</code> of a compound datatype.</td>


```

    </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>size_t H5Tget_member_offset
(hid_t type_id, unsigned memb_no)</code></td>
    <td><b>(COMPOUND)</b> The byte offset
of the beginning of a field within a compound datatype.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>hid_t H5Tget_member_type
(hid_t type_id, unsigned field_idx)</code></td>
    <td><b>(COMPOUND)</b> The datatype of the specified member.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>int H5Tget_array_ndims
(hid_t adtype_id)</code></td>
    <td><b>(ARRAY)</b> The number of dimensions (rank) of the array
datatype object.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>int H5Tget_array_dims
(hid_t adtype_id, hsize_t *dims[])</code></td>
    <td><b>(ARRAY)</b> The sizes of the dimensions and the dimension
permutations of the array datatype object.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>hid_t H5Tget_super(hid_t type)

```

```

</code></td>
<td><b>(ARRAY, VL, ENUM)</b>The base datatype from which the
datatype type is derived.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td><code>herr_t H5Tenum_nameof(hid_t type <br />
void *value, char *name, size_t size)</code></td>
<td><b>(ENUM)</b> The symbol name
that corresponds to the specified value of the enumeration datatype</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td><code>herr_t H5Tenum_valueof(hid_t type <br />
char *name, void *value)</code></td>
<td><b>(ENUM)</b> The value that corresponds to the specified
name of the enumeration datatype</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td><code>herr_t H5Tget_member_value<br />
(hid_t type unsigned memb_no, <br />void *value)</code></td>
<td><b>(ENUM)</b> The value of the
enumeration datatype member <code>memb_no</code></td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

6.4.3. Definition of Datatypes

The HDF5 Library enables user programs to create and modify datatypes. The essential steps are:

- a) Create a new datatype object of a specific composite datatype class,

or

- b) Copy an existing atomic datatype object

- Set properties of the datatype object

- Use the datatype object

- Close the datatype object

To create a user-defined atomic datatype, the procedure is to clone a predefined datatype of the appropriate datatype class (`H5Tcopy`), and then set the datatype properties appropriate to the datatype class. The table below shows how to create a datatype to describe a 1024-bit unsigned integer.

<pre> hid_t new_type = H5Tcopy (H5T_NATIVE_INT); H5Tset_precision(new_type, 1024); H5Tset_sign(new_type, H5T_SGN_NONE); </pre>

```

<tr valign="top">
  <td align="left">
    <b>Example 5. Create a new datatype</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

Composite datatypes are created with a specific API call for each datatype class. The table below shows the creation method for each datatype class. A newly created datatype cannot be used until the datatype properties are set. For example, a newly created compound datatype has no members and cannot be used.

```

<table width="400" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="2" align="left" valign="bottom">
      <b>Table 14. Functions to create each datatype class</b></td>
    </tr>
  <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
  <tr valign="top">
    <td width="50%"><b>Datatype Class</b></td>
    <td width="50%"><b>Function to Create</b></td>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td>COMPOUND</td>
    <td><code>H5Tcreate</code></td>
  </tr>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>

```

```

<tr valign="top">
  <td>OPAQUE</td>
  <td><code>H5Tcreate</code></td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>ENUM</td>
  <td><code>H5Tenum_create</code></td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>ARRAY</td>
  <td><code>H5Tarray_create</code></td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>VL</td>
  <td><code>H5Tvlen_create</code></td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

Once the datatype is created and the datatype properties set, the datatype object can be used.

Predefined datatypes are defined by the library during initialization using the same mechanisms as described here. Each predefined datatype is locked (`H5Tlock`), so that it cannot be changed or destroyed. User-defined datatypes may also be locked using `H5Tlock`.

<!-- NEW PAGE -->

6.4.3.1. User-defined Atomic Datatypes

Table 15 summarizes the API methods that set properties of atomic types. Table 16 shows properties specific to numeric types, Table 17 shows properties specific to the string datatype class. Note that offset, pad, etc. do not apply to strings. Table 18 shows the specific property of the OPAQUE datatype class.

<p>Table 15. API methods that set properties of atomic datatypes</p>	
<p>Functions</p>	<p>Description</p>
<p><code>herr_t H5Tset_size (hid_t <i>type</i>, size_t <i>size</i>)</code></p>	<p>Set the total size of the element in bytes. This includes padding which may appear on either side of the actual value. If this property is reset to a smaller value which would cause the significant part of the data to extend beyond the edge of the datatype, then the offset property is decremented a bit at a time. If the offset reaches zero and the significant part of the data still extends beyond the edge of the datatype then the precision property is decremented a bit at a time.</p>

Decreasing the size of a datatype may fail if the

`H5T_FLOAT` bit fields would extend beyond the significant part of the type. </td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>herr_t H5Tset_order

(hid_t <i>type</i>, H5T_order_t <i>order</i></code></td>

<td>Set the byte order to little-endian

(<code>H5T_ORDER_LE</code>) or big-endian (<code>H5T_ORDER_BE</code>).</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>herr_t H5Tset_precision

(hid_t <i>type</i>, size_t <i>precision</i></code></td>

<td>Set the number of significant bits

of a datatype. The <code>offset</code> property (defined below)

identifies its location. The size property defined above represents

the entire size (in bytes) of the datatype. If the precision is

decreased then padding bits are inserted on the MSB side of the

significant bits (this will fail for <code>H5T_FLOAT</code> types

if it results in the sign, mantissa, or exponent bit field extending

beyond the edge of the significant bit field). On the other hand,

if the precision is increased so that it “hangs over”

the edge of the total size then the offset property is decremented

a bit at a time. If the offset reaches zero and the significant

bits still hang over the edge, then the total size is increased

a byte at a time. </td>

</tr>

<!-- NEW PAGE -->

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<pre><tr valign="top"> <td><code>herr_t H5Tset_offset (hid_t <i>type</i>, size_t <i>offset</i>)</code></td> <td>Set the bit location of the least significant bit of a bit field whose length is <code>precision</code>. The bits of the entire data are numbered beginning at zero at the least significant bit of the least significant byte (the byte at the lowest memory address for a little-endian type or the byte at the highest address for a big-endian type). The offset property defines the bit location of the least significant bit of a bit field whose length is precision. If the offset is increased so the significant bits &ldquo;hang over&rdquo; the edge of the datum, then the size property is automatically incremented.</td> </tr> <tr><td colspan="2"><hr color="green" size="1" /></td></tr> <tr valign="top"> <td><code>herr_t H5Tset_pad (hid_t <i>type</i>, H5T_pad_t <i>lsb</i>, H5T_pad_t <i>msb</i>)</code></td> <td>Set the padding to zeros (<code>H5T_PAD_ZERO</code>) or ones (<code>H5T_PAD_ONE</code>). Padding is the bits of a data element which are not significant as defined by the <code>precision</code> and <code>offset</code> properties. Padding in the low-numbered bits is <code><i>lsb</i></code> padding and padding in the high-numbered bits is <code><i>msb</i></code> padding. </td> </tr> <tr><td colspan="2"><hr color="green" size="3" /></td></tr> </table>

 </pre>	
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

<!-- NEW PAGE -->

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="2" align="left" valign="bottom">
      <b>Table 16. API methods that set properties of numeric datatypes</b></td>
    </tr>
  <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
  <tr valign="top">
    <td width="50%"><b>Functions</b></td>
    <td width="50%"><b>Description</b></td>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td><code>herr_t H5Tset_sign
(hid_t <i>type</i>, H5T_sign_t <i>sign</i>)</code></td>
    <td><b>(INTEGER)</b> Integer
data can be signed two's complement (<code>H5T_SGN_2</code>)
or unsigned (<code>H5T_SGN_NONE</code>).</td>
  </tr>
  <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td><code>herr_t H5Tset_fields
(hid_t <i>type</i>, size_t <i>spos</i>, size_t <i>epos</i>,
size_t <i>esize</i>, size_t <i>mpos</i>, size_t <i>msize</i>)
</code></td>
    <td><b>(FLOAT)</b> Set the
properties define the location (bit position of least significant
bit of the field) and size (in bits) of each field. The sign bit
is always of length one and none of the fields are allowed to overlap.</td>
```

```

    </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>herr_t H5Tset_ebias (hid_t <i>type</i>,
    size_t <i>ebias</i>)</code></td>
    <td><b>(FLOAT)</b> The exponent
    is stored as a non-negative value which is <code>ebias</code> larger
    than the true exponent.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>herr_t H5Tset_norm
    (hid_t <i>type</i>, H5T_norm_t <i>norm</i>)</code></td>
    <td><b>(FLOAT)</b> This
    property describes the normalization method of the mantissa.
    <ul>
        <li><code>H5T_NORM_MSBSET</code>: the mantissa is shifted left
        (if non-zero) until the first bit after the radix point is set and
        the exponent is adjusted accordingly. All bits of the mantissa
        after the radix point are stored. </li>
        <li><code>H5T_NORM IMPLIED</code>: the mantissa is shifted left
        (if non-zero) until the first bit after the radix point is set and
        the exponent is adjusted accordingly. The first bit after the
        radix point is not stored since it is always set. </li>
        <li><code>H5T_NORM_NONE</code>: the fractional part of the
        mantissa is stored without normalizing it. </li></ul></td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td><code>herr_t H5Tset_inpad
    (hid_t <i>type</i>, H5T_pad_t <i>inpad</i>)</code></td>

```

<td>(FLOAT)</td> If any

internal bits (that is, bits between the sign bit, the mantissa field, and the exponent field but within the precision field) are unused, then they will be filled according to the value of this property.

The padding can be: <code>H5T_PAD_NONE</code>, <code>H5T_PAD_ZERO</code> or <code>H5T_PAD_ONE</code>.</td>

</tr>

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

</table>

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="2" align="left" valign="bottom">

Table 17. API methods that set properties of string datatypes</td>

</tr>

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td width="50%">Functions</td>

<td width="50%">Description</td>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td><code>herr_t H5Tset_size (hid_t <i>type</i>,</td>

<code>size_t <i>size</i></code></td>

<td>Set the length of the string, in bytes.

The precision is automatically set to 8*<code>size</code>.</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

```

<tr valign="top">
  <td><code>herr_t H5Tset_precision
  (hid_t <i>type</i>, size_t <i>precision</i>)</code></td>
  <td>The precision must be a multiple of 8.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>herr_t H5Tset_cset
  (hid_t type_id, H5T_cset_t cset )</code></td>
  <td>Two character sets are currently
  supported: ASCII (<code>H5T_CSET_ASCII</code>) and UTF-8
  (<code>H5T_CSET_UTF8</code>).</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>herr_t H5Tset_strpad
  (hid_t type_id, H5T_str_t strpad )</code></td>
  <td>The string datatype has a fixed
  length, but the string may be shorter than the length. This property
  defines the storage mechanism for the left over bytes. The method
  used to store character strings differs with the programming language:
  <ul>
    <li>C usually null terminates strings </li>
    <li>Fortran left-justifies and space-pads strings</li>
  </ul>
  <p>Valid string padding values, as passed in the parameter strpad,
  are as follows: </p>
  <dl>
    <dt><code>H5T_STR_NULLTERM</code> (0)
    <dd>Null terminate (as C does)
    <dt><code>H5T_STR_NULLPAD</code> (1)

```

```

        <dd>Pad with zeros
        <dt><code>H5T_STR_SPACEPAD</code> (2)
        <dd>Pad with spaces (as FORTRAN does).
    </dl></td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

<table width="600" cellspacing="0" align="center" cellpadding="0">
    <tr valign="bottom">
        <td colspan="2" align="left" valign="bottom">
            <b>Table 18. API methods that set properties of opaque datatypes</b></td>
        </tr>
        <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
        <tr valign="top">
            <td width="50%"><b>Functions</b></td>
            <td width="50%"><b>Description</b></td>
        <tr><td colspan="2"><hr color="green" size="1" /></td></tr>
        <tr valign="top">
            <td><code>herr_t H5Tset_tag (hid_t type_id
            <br />const char *tag )</code></td>
            <td>Tags the opaque datatype type_id
            with an ASCII identifier tag.</td>
        </tr>
        <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

<!-- NEW PAGE -->

<h4>Examples</h4>

The example below shows how to create a 128-bit little-endian signed integer type. Increasing the precision of a type automatically increases the total size. Note that the proper procedure is to begin from a type of the intended datatype class which in this case is a

```
<code>NATIVE_INT</code>.
```

<pre> hid_t new_type = H5Tcopy (H5T_NATIVE_INT); H5Tset_precision (new_type, 128); H5Tset_order (new_type, H5T_ORDER_LE); </pre>
<hr style="border: 1px solid green;"/>
<pre> Example 6. Create a new 128-bit little-endian signed integer datatype </pre> <hr style="border: 1px solid green;"/>

The figure below shows the storage layout as the type is defined. The

`H5Tcopy` creates a datatype that is the same as `H5T_NATIVE_INT`. In this example, suppose this is a 32-bit big-endian number (Figure a). The precision is set to 128 bits, which automatically extends the size to 8 bytes (Figure b). Finally, the byte order is set to little-endian (Figure c).

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <table align="center" border="0" width="100%">
        <tr><td>

          <table border="1" align="left">
            <tr>
              <td valign="middle" align="center"><code>Byte 0</code></td>
              <td valign="middle" align="center"><code>Byte 1</code></td>
              <td valign="middle" align="center"><code>Byte 2</code></td>
              <td valign="middle" align="center"><code>Byte 3</code></td>
            </tr>
            <tr>
              <td valign="middle" align="center"><code>01234567</code></td>
              <td valign="middle" align="center"><code>89012345</code></td>
              <td valign="middle" align="center"><code>67890123</code></td>
              <td valign="middle" align="center"><code>45678901</code></td>
            </tr>
          </table>

        </td></tr>
      </table>
    </td>
  </tr>
</table>
```

```
<table border="0" align="left">
  <tr>
    <td>a) The <code>H5T_NATIVE_INT</code> datatype<br />&nbsp;  </td></tr>
</table>
```

```
</td></tr>
```

```
<tr><td>
```

```
<table border="1" align="left">
  <tr>
    <td valign="middle" align="center"><code>Byte 0</code></td>
    <td valign="middle" align="center"><code>Byte 1</code></td>
    <td valign="middle" align="center"><code>Byte 2</code></td>
    <td valign="middle" align="center"><code>Byte 3</code></td>
    <td valign="middle" align="center"><code>Byte 4</code></td>
    <td valign="middle" align="center"><code>Byte 5</code></td>
    <td valign="middle" align="center"><code>Byte 6</code></td>
    <td valign="middle" align="center"><code>Byte 7</code></td>
  </tr>
  <tr>
    <td valign="middle" align="center"><code>01234567</code></td>
    <td valign="middle" align="center"><code>89012345</code></td>
    <td valign="middle" align="center"><code>67890123</code></td>
    <td valign="middle" align="center"><code>45678901</code></td>
    <td valign="middle" align="center"><code>23456789</code></td>
    <td valign="middle" align="center"><code>01234567</code></td>
    <td valign="middle" align="center"><code>89012345</code></td>
    <td valign="middle" align="center"><code>67890123</code></td>
  </tr>
</table>
```



```
</td></tr>
```

```
<tr><td>
```

```
<table border="0" align="left">
```

```
<tr>
```

```
<td>b) Precision is extended to 128-bits, and the size is
```

```
automatically adjusted.<br />&nbsp;  </td></tr>
```

```
</table>
```

```
</td></tr>
```

```
<tr><td>
```

```
<table border="1" align="left">
```

```
<tr>
```

```
<td valign="middle" align="center"><code>Byte 0</code></td>
```

```
<td valign="middle" align="center"><code>Byte 1</code></td>
```

```
<td valign="middle" align="center"><code>Byte 2</code></td>
```

```
<td valign="middle" align="center"><code>Byte 3</code></td>
```

```
<td valign="middle" align="center"><code>Byte 4</code></td>
```

```
<td valign="middle" align="center"><code>Byte 5</code></td>
```

```
<td valign="middle" align="center"><code>Byte 6</code></td>
```

```
<td valign="middle" align="center"><code>Byte 7</code></td>
```

```
</tr>
```

```
<tr>
```

```
<td valign="middle" align="center"><code>01234567</code></td>
```

```
<td valign="middle" align="center"><code>89012345</code></td>
```

```
<td valign="middle" align="center"><code>67890123</code></td>
```

```
<td valign="middle" align="center"><code>45678901</code></td>
```

```
<td valign="middle" align="center"><code>23456789</code></td>
```

```
<td valign="middle" align="center"><code>01234567</code></td>
```

```

        <td valign="middle" align="center"><code>89012345</code></td>
        <td valign="middle" align="center"><code>67890123</code></td>
    </tr>
</table>

</td></tr>
<tr><td>

<table border="0" align="left">
    <tr><td>c) The byte order is switched.</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left" >
        <b>Figure 6. The storage layout for a new 128-bit little-endian
        signed integer datatype</b>
        <hr color="green" size="3" /></td>
    </tr>
</table>
<br />

```

The significant bits of a data element can be offset from the beginning of the memory for that element by an amount of padding. The `offset` property specifies the number of bits of padding that appear to the “right of” the value. The table and figure below show how a 32-bit unsigned integer with 16-bits of precision having the value

`<code>0x1122</code>` will be laid out in memory.</p>

<!-- NEW PAGE -->

<table width="600" cellpadding="0" align="center" cellspacing="0">

<tr valign="bottom">

<td colspan="5" align="left" valign="bottom">

Table 19. Memory Layout for a 32-bit unsigned integer</td>

</tr>

<tr><td colspan="5"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td>Byte Position</td>

<td>Big-Endian
Offset=0</td>

<td>Big-Endian
Offset=16</td>

<td>Little-Endian
Offset=0</td>

<td>Little-Endian
Offset=16</td>

<tr><td colspan="5"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>0:</td>

<td>[pad]</td>

<td>[0x11]</td>

<td>[0x22]</td>

<td>[pad]</td>

</tr>

<tr><td colspan="5"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>1:</td>

<td>[pad]</td>

<td>[0x22]</td>

<td>[0x11]</td>

<td>[pad]</td>

</tr>

```

<tr><td colspan="5"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>2:</td>
  <td>[0x11]</td>
  <td>[pad]</td>
  <td>[pad]</td>
  <td>[0x22]</td>
</tr>
<tr><td colspan="5"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>3:</td>
  <td>[0x22]</td>
  <td>[pad]</td>
  <td>[pad]</td>
  <td>[0x11]</td>
</tr>
<tr><td colspan="5"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<table width="400" cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>

  <tr>
    <td align="center" valign="middle">Big-Endian: Offset = 0</td>

```

```
</tr>
```

```
<tr><td>
```

```
<table border="1" align="center">
```

```
<tr>
```

```
<td valign="middle" align="center"><code>Byte 0</code></td>
```

```
<td valign="middle" align="center"><code>Byte 1</code></td>
```

```
<td valign="middle" align="center"><code>Byte 2</code></td>
```

```
<td valign="middle" align="center"><code>Byte 3</code></td>
```

```
</tr>
```

```
<tr>
```

```
<td valign="middle" align="center"><code>01234567</code></td>
```

```
<td valign="middle" align="center"><code>89012345</code></td>
```

```
<td valign="middle" align="center"><code>67890123</code></td>
```

```
<td valign="middle" align="center"><code>45678901</code></td>
```

```
</tr>
```

```
<tr>
```

```
<td valign="middle" align="center"><code><em>PPPPPPPP</em></code></td>
```

```
<td valign="middle" align="center"><code><em>PPPPPPPP</em></code></td>
```

```
<td valign="middle" align="center"><code>00010001</code></td>
```

```
<td valign="middle" align="center"><code>00100010</code></td>
```

```
</tr>
```

```
</table>
```

```
</td></tr>
```

```
<tr>
```

```
<td valign="middle" align="center">&nbsp;<br />Big-Endian: Offset = 16</td>
```

```
</tr>
```

```
<tr>
```

```
<td>
```

```

<table border="1" align="center">
  <tr>
    <td valign="middle" align="center"><code>Byte 0</code></td>
    <td valign="middle" align="center"><code>Byte 1</code></td>
    <td valign="middle" align="center"><code>Byte 2</code></td>
    <td valign="middle" align="center"><code>Byte 3</code></td>
  </tr>
  <tr>
    <td valign="middle" align="center"><code>01234567</code></td>
    <td valign="middle" align="center"><code>89012345</code></td>
    <td valign="middle" align="center"><code>67890123</code></td>
    <td valign="middle" align="center"><code>45678901</code></td>
  </tr>
  <tr>
    <td valign="middle" align="center"><code>00010001</code></td>
    <td valign="middle" align="center"><code>00100010</code></td>
    <td valign="middle" align="center"><code><em>PPPPPPPP</em></code></td>
    <td valign="middle" align="center"><code><em>PPPPPPPP</em></code></td>
  </tr>
</table>

</td></tr>
<tr>
  <td valign="middle" align="center">&nbsp;<br />Little-Endian:
  Offset = 0</td>
</tr>

<tr>
  <td>

<table border="1" align="center">

```

```

<tr>
  <td valign="middle" align="center"><code>Byte 0</code></td>
  <td valign="middle" align="center"><code>Byte 1</code></td>
  <td valign="middle" align="center"><code>Byte 2</code></td>
  <td valign="middle" align="center"><code>Byte 3</code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code>01234567</code></td>
  <td valign="middle" align="center"><code>89012345</code></td>
  <td valign="middle" align="center"><code>67890123</code></td>
  <td valign="middle" align="center"><code>45678901</code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code>00010001</code></td>
  <td valign="middle" align="center"><code>00100010</code></td>
  <td valign="middle" align="center"><code><em>PPPPPPPP</em></code></td>
  <td valign="middle" align="center"><code><em>PPPPPPPP</em></code></td>
</tr>
</table>

</td></tr>
<tr>
  <td valign="middle" align="center">&nbsp;<br />Little-Endian:
  Offset = 16</td>
</tr>
<tr>
  <td>
<table border="1" align="center">
  <tr>
    <td valign="middle" align="center"><code>Byte 0</code></td>
    <td valign="middle" align="center"><code>Byte 1</code></td>

```

```

<td valign="middle" align="center"><code>Byte 2</code></td>
<td valign="middle" align="center"><code>Byte 3</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code>01234567</code></td>
<td valign="middle" align="center"><code>89012345</code></td>
<td valign="middle" align="center"><code>67890123</code></td>
<td valign="middle" align="center"><code>45678901</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code><em>PPPPPPPP</em></code></td>
<td valign="middle" align="center"><code><em>PPPPPPPP</em></code></td>
<td valign="middle" align="center"><code>00010001</code></td>
<td valign="middle" align="center"><code>00100010</code></td>
</tr>
</table>

</td></tr>
</table>&nbsp;

</td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td align="left" >
<b>Figure 7. Memory Layout for a 32-bit unsigned integer</b>
<hr color="green" size="3" /></td>
</tr>
</table>
<br />

```


If the offset is incremented then the total size is incremented also if necessary to prevent significant bits of the value from hanging over the edge of the datatype.

The bits of the entire data are numbered beginning at zero at the least significant bit of the least significant byte (the byte at the lowest memory address for a little-endian type or the byte at the highest address for a big-endian type). The `offset` property defines the bit location of the least significant bit of a bit field whose length is `precision`. If the offset is increased so the significant bits “hang over” the edge of the datum, then the `size` property is automatically incremented.

To illustrate the properties of the integer datatype class, the example below shows how to create a user-defined datatype that describes a 24-bit signed integer that starts on the third bit of a 32-bit word. The datatype is specialized from a 32-bit integer, the `precision` is set to 24 bits, and the `offset` is set to 3.

```
<table width="600" cellspacing="0" align="center">
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<hr color="green" size="3"/>
```

```
<pre>
```

```
hid_t dt;
```

```
dt = H5Tcopy(H5T_SDT_I32LE);
```

```

H5Tset_precision(dt, 24);
H5Tset_offset(dt,3);
H5Tset_pad(dt, H5T_PAD_ZERO, H5T_PAD_ONE);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 7. A user-defined datatype with a 24-bit signed integer</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

The figure below shows the storage layout for a data element. Note that the unused bits in the offset will be set to zero and the unused bits at the end will be set to one, as specified in the `H5Tset_pad` call.

```

<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <br />
      <table border="1" align="center" width="67%">
        <tr>
          <td align="center"><code>Byte 0</code></td>
          <td align="center"><code>Byte 1</code></td>
          <td align="center"><code>Byte 2</code></td>

```

```

        <td valign="middle" align="center"><code>Byte 3</code></td>
    </tr>
<tr>
    <td valign="middle" align="center"><code>01234567</code></td>
    <td valign="middle" align="center"><code>89012345</code></td>
    <td valign="middle" align="center"><code>67890123</code></td>
    <td valign="middle" align="center"><code>45678901</code></td>
</tr>
<tr>
    <td valign="middle" align="center"><code><strong><em>ooo</em></strong>00000</code></td>
    <td valign="middle" align="center"><code>00000000</code></td>
    <td valign="middle" align="center"><code>00000000</code></td>
    <td valign="middle" align="center"><code>00s<strong><em>ppppp</em></strong></code></td>
</tr>
<tr>
    <td valign="middle" align="center" colspan="4">
        
    </td>
</tr>
</table>
<br />
<tr><td>
</td>
</tr>

<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left" >
        <b>Figure 8. A user-defined integer datatype a range of -1,048,583
        to 1,048,584</b>
        <hr color="green" size="3"/></td>

```

```
</tr>
</table>
<br />
```

To illustrate a user-defined floating point number, the example below shows how to create a 24-bit floating point number that starts 5 bits into a 4 byte word. The floating point number is defined to have a mantissa of 19 bits (bits 5-23), an exponent of 3 bits (25-27), and the sign bit is bit 28. (Note that this is an illustration of what can be done and is not necessarily a floating point format that a user would require.)

```
<!-- NEW PAGE -->
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
hid_t dt;

dt = H5Tcopy(H5T_IEEE_F32LE);

H5Tset_precision(dt, 24);
H5Tset_fields (dt, 28, 25, 3, 5, 19);
H5Tset_pad(dt, H5T_PAD_ZERO, H5T_PAD_ONE);
H5Tset_inpad(dt, H5T_PAD_ZERO);</pre> </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
</table>
```

```

<td align="left">
  <b>Example 8. A user-defined 24-bit floating point datatype</b>
  <hr color="green" size="3"/></td>
</tr>
</table>
<br />
<br />

```

```

<table width="500" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>

      <table border="1" align="center" width="67%">
        <tr>
          <td valign="middle" align="center"><code>Byte 0</code></td>
          <td valign="middle" align="center"><code>Byte 1</code></td>
          <td valign="middle" align="center"><code>Byte 2</code></td>
          <td valign="middle" align="center"><code>Byte 3</code></td>
        </tr>
        <tr>
          <td valign="middle" align="center"><code>01234567</code></td>
          <td valign="middle" align="center"><code>89012345</code></td>
          <td valign="middle" align="center"><code>67890123</code></td>
          <td valign="middle" align="center"><code>45678901</code></td>
        </tr>
        <tr>
          <td valign="middle" align="center"><code><strong><em>ooooo</em></strong>mmm</code></td>
        </tr>
      </table>
    </td>
  </tr>
</table>

```

```

<td valign="middle" align="center"><code>mmmmmmmm</code></td>
<td valign="middle" align="center"><code>mmmmmmmm</code></td>
<td valign="middle" align="center"><code><strong>i</strong>eees<strong>
  <em>ppp</em></strong></code></td>
</tr>
<tr>
<td valign="middle" align="center" colspan="4">

</td>
</tr>
</table>

</td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 9. A user-defined floating point datatype</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

The figure above shows the storage layout of a data element for this datatype. Note that there is an unused bit (24) between the mantissa and the exponent. This bit is filled with the `inpad` value which in this case is 0.

The sign bit is always of length one and none of the fields are allowed to overlap. When expanding a floating-point type one should set the precision first; when decreasing the size one should set the field positions and sizes first.

6.4.3.2. Composite Datatypes

All composite datatypes must be user-defined; there are no predefined composite datatypes.

6.4.3.2.1. Compound Datatypes

The subsections below describe how to create a compound datatype and how to write and read data of a compound datatype.

6.4.3.2.1.1. Defining Compound Datatypes

Compound datatypes are conceptually similar to a C struct or Fortran derived types. The compound datatype defines a contiguous sequence of bytes, which are formatted using one up to 2^{16} datatypes (members). A compound datatype may have any number of members, in any order, and the members may have any datatype, including compound. Thus, complex nested compound datatypes can be created. The total size of the compound datatype is greater than or equal to the sum of the size of its members, up to a maximum of 2^{32} bytes. HDF5 does not support datatypes with distinguished records or the equivalent of C unions or Fortran EQUIVALENCE statements.

Usually a C struct or Fortran derived type will be defined to hold a data point in memory, and the offsets of the members in memory will be the offsets of the struct members from the beginning of an instance

of the struct. The HDF5 C library provides a macro `HOFFSET(s,m)` to calculate the member's offset. The HDF5 Fortran applications have to calculate offsets by using sizes of members datatypes and by taking in consideration the order of members in the Fortran derived type.

<dl>

<dt><code>HOFFSET(s,m)</code>

<dd>This macro computes the offset of member *m* within a struct *s*</dd>

<dt><code>offsetof(s,m)</code>

<dd>This macro defined in `stddef.h` does exactly the same thing as the `HOFFSET()` macro.</dd>

</dl>

Note for Fortran users: Offsets of Fortran structure members correspond to the offsets within a packed datatype (see explanation below) stored in an HDF5 file.

Each member of a compound datatype must have a descriptive name which is the key used to uniquely identify the member within the compound datatype. A member name in an HDF5 datatype does not necessarily have to be the same as the name of the member in the C struct or Fortran derived type, although this is often the case. Nor does one need to define all members of the C struct or Fortran derived type in the HDF5 compound datatype (or vice versa).

Unlike atomic datatypes which are derived from other atomic datatypes, compound datatypes are created from scratch. First, one creates an empty compound datatype and specifies its total size. Then members are added to the compound datatype in any order. Each member type is inserted at a

designated offset. Each member has a name which is the key used to uniquely identify the member within the compound datatype.

The example below shows a way of creating an HDF5 C compound datatype to describe a complex number. This is a structure with two components, "real" and "imaginary", and each component is a double. An equivalent C struct whose type is defined by the `complex_t` struct is shown.

<pre> typedef struct { double re; /*real part*/ double im; /*imaginary part*/ } complex_t; hid_t complex_id = H5Tcreate (H5T_COMPOUND, sizeof (complex_t)); H5Tinsert (complex_id, "real", HOFFSET(complex_t,re), H5T_NATIVE_DOUBLE); H5Tinsert (complex_id, "imaginary", HOFFSET(complex_t,im), H5T_NATIVE_DOUBLE); </pre>	<hr style="border: 1px solid green; width: 300px;"/> <p>Example 9. A compound datatype for complex numbers in C</p> <hr style="border: 1px solid green; width: 300px;"/>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```

    </tr>
</table>
<br />

```

The example below shows a way of creating an HDF5 Fortran compound datatype to describe a complex number. This is a Fortran derived type with two components, `“real”` and `“imaginary”`, and each component is DOUBLE PRECISION. An equivalent Fortran TYPE whose type is defined by the TYPE `complex_t` is shown.

```

<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
TYPE complex_t
  DOUBLE PRECISION re  ! real part
  DOUBLE PRECISION im; ! imaginary part
END TYPE complex_t

CALL h5tget_size_f(H5T_NATIVE_DOUBLE, re_size, error)
CALL h5tget_size_f(H5T_NATIVE_DOUBLE, im_size, error)
complex_t_size = re_size + im_size
CALL h5tcreate_f(H5T_COMPOUND_F, complex_t_size, type_id)
offset = 0
CALL h5tinsert_f(type_id, &ldquo;real&rdquo;, offset, H5T_NATIVE_DOUBLE, error)
offset = offset + re_size
CALL h5tinsert_f(type_id, &ldquo;imaginary&rdquo;, offset, H5T_NATIVE_DOUBLE, error)</pre>
    </td>
  </tr>

```

```

<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 10. A compound datatype for complex numbers in Fortran</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

Important Note: The compound datatype is created with a size sufficient to hold all its members. In the C example above, the size of the C struct and the `HOFFSET` macro are used as a convenient mechanism to determine the appropriate size and offset. Alternatively, the size and offset could be manually determined: the size can be set to 16 with `"real"` at offset 0 and `"imaginary"` at offset 8. However, different platforms and compilers have different sizes for `"double"` and may have alignment restrictions which require additional padding within the structure. It is much more portable to use the `HOFFSET` macro which assures that the values will be correct for any platform.

The figure below shows how the compound datatype would be laid out assuming that `NATIVE_DOUBLE` are 64-bit numbers and that there are no alignment requirements. The total size of the compound datatype will be 16 bytes, the `"real"` component will start at byte 0, and `"imaginary"` will start at byte 8.

```

<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">

```

```
<hr color="green" size="3"/>
```

```
<table border="1" align="center" width="550">
<tr>
<td valign="top" align="right" rowspan="4" width="150">
  
</td>
  <td valign="middle" align="center" width="100"><code>Byte 0</code></td>
  <td valign="middle" align="center" width="100"><code>Byte 1</code></td>
  <td valign="middle" align="center" width="100"><code>Byte 2</code></td>
  <td valign="middle" align="center" width="100"><code>Byte 3</code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
  <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
  <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
  <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code>Byte 4</code></td>
  <td valign="middle" align="center"><code>Byte 5</code></td>
  <td valign="middle" align="center"><code>Byte 6</code></td>
  <td valign="middle" align="center"><code>Byte 7</code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
  <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
  <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
  <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
</tr>
<tr>
```

```

<td valign="top" align="right" rowspan="4" width="150">
  
</td>
  <td valign="middle" align="center"><code>Byte 8</code></td>
<td valign="middle" align="center"><code>Byte 9</code></td>
<td valign="middle" align="center"><code>Byte 10</code></td>
<td valign="middle" align="center"><code>Byte 11</code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code>Byte 12</code></td>
  <td valign="middle" align="center"><code>Byte 13</code></td>
  <td valign="middle" align="center"><code>Byte 14</code></td>
  <td valign="middle" align="center"><code>Byte 15</code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
</tr>
</table>

<table align="center" border="0" width="550">
  <tr>
    <td valign="top" align="right" width="150">&nbsp;</td>

```

```

<td valign="top" align="left" colspan="4">Total size of
    compound datatype is 16 bytes</td>
</tr>
</table>

```

```

</td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left" >
        <b>Figure 10. Layout of a compound datatype</b>
        <hr color="green" size="3" /></td>
</tr>
</table>
<br />

```

The members of a compound datatype may be any HDF5 datatype including the compound, array, and variable-length (VL) types. The figure and example below show the memory layout and code which creates a compound datatype composed of two complex values, and each complex value is also a compound datatype as in the figure above.

```

<table width="600" cellpadding="0" align="center">
    <tr valign="top">
        <td align="center">
            <hr color="green" size="3" />

```

```

<table border="1" align="center" width="550">
  <tr>
    <td valign="top" align="right" rowspan="4" width="150">
      
    </td>
    <td valign="middle" align="center" width="100"><code>Byte 0</code></td>
    <td valign="middle" align="center" width="100"><code>Byte 1</code></td>
    <td valign="middle" align="center" width="100"><code>Byte 2</code></td>
    <td valign="middle" align="center" width="100"><code>Byte 3</code></td>
  </tr>
  <tr>
    <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
    <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
    <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
    <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
  </tr>
  <tr>
    <td valign="middle" align="center"><code>Byte 4</code></td>
    <td valign="middle" align="center"><code>Byte 5</code></td>
    <td valign="middle" align="center"><code>Byte 6</code></td>
    <td valign="middle" align="center"><code>Byte 7</code></td>
  </tr>
  <tr>
    <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
    <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
    <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
    <td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
  </tr>
  <tr>
    <td valign="top" align="right" rowspan="4" width="150">

```

```


</td>
    <td valign="middle" align="center"><code>Byte 8</code></td>
<td valign="middle" align="center"><code>Byte 9</code></td>
<td valign="middle" align="center"><code>Byte 10</code></td>
<td valign="middle" align="center"><code>Byte 11</code></td>
</tr>
<tr>
    <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
    <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
    <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
    <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
</tr>
<tr>
    <td valign="middle" align="center"><code>Byte 12</code></td>
    <td valign="middle" align="center"><code>Byte 13</code></td>
    <td valign="middle" align="center"><code>Byte 14</code></td>
    <td valign="middle" align="center"><code>Byte 15</code></td>
</tr>
<tr>
    <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
    <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
    <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
    <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
</tr>
<tr>
    <td valign="top" align="right" rowspan="4">
</td>
    <td valign="middle" align="center" width="100"><code>Byte 16</code></td>
    <td valign="middle" align="center" width="100"><code>Byte 17</code></td>
    <td valign="middle" align="center" width="100"><code>Byte 18</code></td>

```



```

<td valign="middle" align="center" width="100"><code>Byte 19</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
<td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
<td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
<td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
</tr>
<tr>
<td valign="middle" align="center"><code>Byte 20</code></td>
<td valign="middle" align="center"><code>Byte 21</code></td>
<td valign="middle" align="center"><code>Byte 22</code></td>
<td valign="middle" align="center"><code>Byte 23</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
<td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
<td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
<td valign="middle" align="center"><code><strong>rrrrrrrr</strong></code></td>
</tr>
<tr>
<td valign="top" align="right" rowspan="4">
</td>
<td valign="middle" align="center"><code>Byte 24</code></td>
<td valign="middle" align="center"><code>Byte 25</code></td>
<td valign="middle" align="center"><code>Byte 26</code></td>
<td valign="middle" align="center"><code>Byte 27</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
<td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>

```

```

<td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
<td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code>Byte 28</code></td>
  <td valign="middle" align="center"><code>Byte 29</code></td>
  <td valign="middle" align="center"><code>Byte 30</code></td>
  <td valign="middle" align="center"><code>Byte 31</code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
  <td valign="middle" align="center"><code><strong>iiiiiii</strong></code></td>
</tr>
</table>

<table align="center" width="550">
  <tr>
    <td width="150">&nbsp;</td>
    <td>Total size of compound datatype is 32 bytes.</td>
  </tr>
</table>
</td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 11. Layout of a compound datatype nested within a compound
    datatype</b>
    <hr color="green" size="3"/></td>

```

```

</tr>
</table>
<br />
<br />

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
typedef struct {
  complex_t x;
  complex_t y;
} surf_t;

hid_t complex_id, surf_id; /*hdf5 datatypes*/

complex_id = H5Tcreate (H5T_COMPOUND, sizeof(complex_t));
H5Tinsert (complex_id, &ldquo;re&rdquo;, HOFFSET(complex_t,re),
           H5T_NATIVE_DOUBLE);
H5Tinsert (complex_id, &ldquo;im&rdquo;, HOFFSET(complex_t,im),
           H5T_NATIVE_DOUBLE);

surf_id = H5Tcreate (H5T_COMPOUND, sizeof(surf_t));
H5Tinsert (surf_id, &ldquo;x&rdquo;, HOFFSET(surf_t,x), complex_id);
H5Tinsert (surf_id, &ldquo;y&rdquo;, HOFFSET(surf_t,y), complex_id);</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">

```

Example 11. Code for a compound datatype nested within a compound datatype

Note that a similar result could be accomplished by creating a compound datatype and inserting four fields. See the figure below. This results in the same layout as the figure above. The difference would be how the fields are addressed. In the first case, the real part of `y` is called `y.re`; in the second case it is `y-re`.

<table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top;"> <hr/> <pre> typedef struct { complex_t x; complex_t y; } surf_t; hid_t surf_id = H5Tcreate (H5T_COMPOUND, sizeof(surf_t)); H5Tinsert (surf_id, "x-re", HOFFSET(surf_t,x.re), H5T_NATIVE_DOUBLE); </pre> </td> </tr> </table>	<hr/> <pre> typedef struct { complex_t x; complex_t y; } surf_t; hid_t surf_id = H5Tcreate (H5T_COMPOUND, sizeof(surf_t)); H5Tinsert (surf_id, "x-re", HOFFSET(surf_t,x.re), H5T_NATIVE_DOUBLE); </pre>
<hr/> <pre> typedef struct { complex_t x; complex_t y; } surf_t; hid_t surf_id = H5Tcreate (H5T_COMPOUND, sizeof(surf_t)); H5Tinsert (surf_id, "x-re", HOFFSET(surf_t,x.re), H5T_NATIVE_DOUBLE); </pre>	

```

H5Tinsert (surf_id, &ldquo;x-im&rdquo;, HOFFSET(surf_t,x.im),
          H5T_NATIVE_DOUBLE);
H5Tinsert (surf_id, &ldquo;y-re&rdquo;, HOFFSET(surf_t,y.re),
          H5T_NATIVE_DOUBLE);
H5Tinsert (surf_id, &ldquo;y-im&rdquo;, HOFFSET(surf_t,y.im),
          H5T_NATIVE_DOUBLE);    </pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 12. Another compound datatype nested within a
    compound datatype </b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

The members of a compound datatype do not always fill all the bytes. The `HOFFSET` macro assures that the members will be laid out according to the requirements of the platform and language. The example below shows an example of a C struct which requires extra bytes of padding on many platforms. The second element, `b`, is a 1-byte character followed by an 8 byte double, `c`. On many systems, the 8-byte value must be stored on a 4- or 8-byte boundary. This requires the struct to be larger than the sum of the size of its elements.

In the example below, `sizeof` and `HOFFSET` are used to assure that the members are inserted at the correct offset to match the memory conventions of the platform. The figure below shows how this data element would be stored in memory, assuming the double must start on a 4-byte boundary. Notice the extra bytes between `b` and `c`.

<pre> typedef struct s1_t { int a; char b; double c; } s1_t; s1_tid = H5Tcreate (H5T_COMPOUND, sizeof(s1_t)); H5Tinsert(s1_tid, &ldquo;a_name&rdquo;, HOFFSET(s1_t, a), H5T_NATIVE_INT); H5Tinsert(s1_tid, &ldquo;b_name&rdquo;, HOFFSET(s1_t, b), H5T_NATIVE_CHAR); H5Tinsert(s1_tid, &ldquo;c_name&rdquo;, HOFFSET(s1_t, c), H5T_NATIVE_DOUBLE); </pre>
<p>Example 13. A compound datatype that requires padding</p>

```
</table>
```

```
<br />
```

```
<br />
```

```
<!-- NEW PAGE -->
```

```
<table width="600" cellspacing="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="center">
```

```
      <hr color="green" size="3"/>
```

```
      
```

```
    </td>
```

```
  </tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
  <td align="left" >
```

```
    <b>Figure 12. Memory layout of a compound datatype that requires  
padding </b>
```

```
    <hr color="green" size="3"/></td>
```

```
  </tr>
```

```
</table>
```

```
<br />
```

However, data stored on disk does not require alignment, so unaligned versions of compound data structures can be created to improve space efficiency on disk. These unaligned compound datatypes can be created by computing offsets by hand to eliminate inter-member padding, or the members can be packed by calling `H5Tpack` (which modifies a datatype directly, so it is usually preceded by a call to

`H5Tcopy`).

The example below shows how to create a disk version of the compound datatype from the figure above in order to store data on disk in as compact a form as possible.

Packed compound datatypes should generally not be used to describe memory as they may violate alignment constraints for the architecture being used. Note also that using a packed datatype for disk storage may involve a higher data conversion cost.

<div style="border: 1px solid green; padding: 5px; margin-bottom: 10px;"> <pre> hid_t s2_tid = H5Tcopy (s1_tid); H5Tpack (s2_tid); </pre> </div>	<div style="border: 1px solid green; padding: 5px; margin-bottom: 10px;"> <p>Example 14. Create a packed compound datatype in C</p> </div>
<div style="border: 1px solid green; padding: 5px;"> </div>	

The example below shows the sequence of Fortran calls to create a packed compound datatype. An HDF5 Fortran compound datatype never describes a compound datatype in memory and compound data is *ALWAYS* written by fields as described in the next section. Therefore packing is not needed unless the offset of each consecutive member is not equal to the sum of the sizes of the previous members.

<div style="text-align: left;"> <hr style="border: 0; border-top: 1px solid green; margin: 10px 0;"/> <pre>CALL h5tcopy_f(s1_id, s2_id, error) CALL h5tpack_f(s2_id, error)</pre> </div>	<div style="text-align: left;"> <hr style="border: 0; border-top: 1px solid green; margin: 10px 0;"/> </div>
<div style="text-align: left;"> Example 15. Create a packed compound datatype in Fortran <hr style="border: 0; border-top: 1px solid green; margin: 10px 0;"/> </div>	<div style="text-align: left;"> <hr style="border: 0; border-top: 1px solid green; margin: 10px 0;"/> </div>

6.4.3.2.1.2. Creating and Writing Datasets with Compound Datatypes

Creating datasets with compound datatypes is similar to creating datasets with any other HDF5 datatypes. But writing and reading may be different since datasets that have compound datatypes can be written or read by a field (member) or subsets of fields (members). The compound datatype is the only composite datatype that supports “sub-setting” by the elements the datatype is built from.

The example below shows a C example of creating and writing a dataset with a compound datatype.

--

--	--

--	--	--

--	--	--	--

--	--	--	--	--

```
typedef struct s1_t {
```

```
    int  a;
```

```
    float b;
```

```
    double c;
```

```
} s1_t;
```

```
s1_t data[LENGTH];
```

```
/* Initialize data */
```

```
for (i = 0; i < LENGTH; i++) {
```

```
    data[i].a = i;
```

```
    data[i].b = i*i;
```

```

    data[i].c = 1./(i+1);
...
s1_tid = H5Tcreate (H5T_COMPOUND, sizeof(s1_t));
H5Tinsert(s1_tid, &ldquo;a_name&rdquo;, HOFFSET(s1_t, a), H5T_NATIVE_INT);
H5Tinsert(s1_tid, &ldquo;b_name&rdquo;, HOFFSET(s1_t, b), H5T_NATIVE_FLOAT);
H5Tinsert(s1_tid, &ldquo;c_name&rdquo;, HOFFSET(s1_t, c), H5T_NATIVE_DOUBLE);
...
dataset_id = H5Dcreate(file_id, &ldquo;SDScompound.h5&rdquo;, s1_t, space_id,
    H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
H5Dwrite (dataset_id, s1_tid, H5S_ALL, H5S_ALL, H5P_DEFAULT, data);</pre> </td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 16. Create and write a dataset with a compound datatype in C</b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

<!-- NEW PAGE -->

<p>The example below shows the content of the file written on
a little-endian machine.</p>

<table width="600" cellpadding="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

HDF5 “SDScompound.h5” {

```

GROUP "ArrayOfStructures" {
  DATASET "ArrayOfStructures" {
    DATATYPE H5T_COMPOUND {
      H5T_STD_I32LE "a_name";
      H5T_IEEE_F32LE "b_name";
      H5T_IEEE_F64LE "c_name";
    }
    DATASPACE SIMPLE { ( 3 ) / ( 3 ) }
    DATA {
      (0): {
        0,
        0,
        1
      },
      (1): {
        1,
        1,
        0.5
      },
      (2): {
        2,
        4,
        0.333333
      }
    }
  }
}

```

```

}

```

```


```

```


```

```


```

| |
|-------------------------------------------------------------------------------------------------------------------------------|
| <p>Example 17. Create and write a little-endian dataset with a compound datatype in C</p> <hr color="green" size="3"/> |
|-------------------------------------------------------------------------------------------------------------------------------|

<!-- NEW PAGE -->

It is not necessary to write the whole data at once. Datasets with compound datatypes can be written by field or by subsets of fields. In order to do this one has to remember to set the transfer property of the dataset using the `H5Pset_preserve` call and to define the memory datatype that corresponds to a field. The example below shows how float and double fields are written to the dataset.

| |
|----------------------------------------------------------------------------------------------|
| <pre> typedef struct sb_t { float b; double c; } sb_t; typedef struct sc_t { </pre> |
|----------------------------------------------------------------------------------------------|

```

    float b;
    double c;
} sc_t;
sb_t data1[LENGTH];
sc_t data2[LENGTH];

/* Initialize data */
for (i = 0; i < LENGTH; i++) {
    data1.b = i*i;
    data2.c = 1./(i+1);
}
...
/* Create dataset as in example 15 */
...
/* Create memory datatypes corresponding to float and
double datatype files */

sb_tid = H5Tcreate (H5T_COMPOUND, sizeof(sb_t));
H5Tinsert(sb_tid, "b_name", HOFFSET(sb_t, b), H5T_NATIVE_FLOAT);
sc_tid = H5Tcreate (H5T_COMPOUND, sizeof(sc_t));
H5Tinsert(sc_tid, "c_name", HOFFSET(sc_t, c), H5T_NATIVE_DOUBLE);
...
/* Set transfer property */
xfer_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_preserve(xfer_id, 1);
H5Dwrite (dataset_id, sb_tid, H5S_ALL, H5S_ALL, xfer_id, data1);
H5Dwrite (dataset_id, sc_tid, H5S_ALL, H5S_ALL, xfer_id, data2);

```

Example 18. Writing floats and doubles to a dataset

<!-- NEW PAGE -->

The figure below shows the content of the file written on a little-endian machine. Only float and double fields are written. The default fill value is used to initialize the unwritten integer field.

--

| |
 |

```


```

```

HDF5 &ldquo;SDScompound.h5&rdquo; {

```

```

GROUP &ldquo;/&rdquo; {

```

```

    DATASET &ldquo;ArrayOfStructures&rdquo; {

```

```

        DATATYPE H5T_COMPOUND {

```

```

            H5T_STD_I32LE &ldquo;a_name&rdquo;;

```

```

            H5T_IEEE_F32LE &ldquo;b_name&rdquo;;

```

```

            H5T_IEEE_F64LE &ldquo;c_name&rdquo;;

```

```

        }

```

```

        DATASPACE SIMPLE { ( 3 ) / ( 3 ) }

```

```

        DATA {

```

```

            (0): {

```

```
0,
0,
1
},
(1): {
0,
1,
0.5
},
(2): {
0,
4,
0.333333
}
}
}
}
}</pre> </td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td align="left">
<b>Example 19. Writing floats and doubles to a dataset on a little-endian
system</b>
<hr color="green" size="3"/></td>
</tr>
</table>
<br />
```


<!-- NEW PAGE -->

<p>The example below contains a Fortran example that creates and writes a dataset with a compound datatype. As this example illustrates, writing and reading compound datatypes in Fortran is *always* done by fields. The content of the written file is the same as shown in the example above.</p>

<table width="600" cellpadding="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

! One cannot write an array of a derived datatype in Fortran.

TYPE s1_t

INTEGER a

REAL b

DOUBLE PRECISION c

END TYPE s1_t

TYPE(s1_t) d(LENGTH)

! Therefore, the following code initializes an array corresponding

! to each field in the derived datatype and writes those arrays

! to the dataset

INTEGER, DIMENSION(LENGTH) :: a

REAL, DIMENSION(LENGTH) :: b

DOUBLE PRECISION, DIMENSION(LENGTH) :: c

! Initialize data

do i = 1, LENGTH

a(i) = i-1

```
b(i) = (i-1) * (i-1)
c(i) = 1./i
enddo

...

! Set dataset transfer property to preserve partially initialized fields
! during write/read to/from dataset with compound datatype.
!
CALL h5pcreate_f(H5P_DATASET_XFER_F, plist_id, error)
CALL h5pset_preserve_f(plist_id, .TRUE., error)
...
!
! Create compound datatype.
!
! First calculate total size by calculating sizes of each member
!
CALL h5tget_size_f(H5T_NATIVE_INTEGER, type_sizei, error)
CALL h5tget_size_f(H5T_NATIVE_REAL, type_sizer, error)
CALL h5tget_size_f(H5T_NATIVE_DOUBLE, type_sized, error)
type_size = type_sizei + type_sizer + type_sized
CALL h5tcreate_f(H5T_COMPOUND_F, type_size, dtype_id, error)
!
! Insert members
!
!
! INTEGER member
!
offset = 0
CALL h5tinsert_f(dtype_id, 'a_name', offset, H5T_NATIVE_INTEGER, error)
!
```

```

! REAL member
!
offset = offset + type_sizei
CALL h5tinsert_f(dtype_id, &ldquo;b_name&rdquo;, offset, H5T_NATIVE_REAL, error)
!
! DOUBLE PRECISION member
!
offset = offset + type_sizer
CALL h5tinsert_f(dtype_id, &ldquo;c_name&rdquo;, offset, H5T_NATIVE_DOUBLE, error)

!
! Create the dataset with compound datatype.
!
CALL h5dcreate_f(file_id, dsetname, dtype_id, dspace_id, &amp;
                dset_id, error, H5P_DEFAULT_F, H5P_DEFAULT_F, H5P_DEFAULT_F)
!
...
! Create memory types. We have to create a compound datatype
! for each member we want to write.
!
!
CALL h5tcreate_f(H5T_COMPOUND_F, type_sizei, dt1_id, error)
offset = 0
CALL h5tinsert_f(dt1_id, &ldquo;a_name&rdquo;, offset, H5T_NATIVE_INTEGER, error)
!
CALL h5tcreate_f(H5T_COMPOUND_F, type_sizer, dt2_id, error)
offset = 0
CALL h5tinsert_f(dt2_id, &ldquo;b_name&rdquo;, offset, H5T_NATIVE_REAL, error)
!
CALL h5tcreate_f(H5T_COMPOUND_F, type_sized, dt3_id, error)
offset = 0

```

```
CALL h5tinsert_f(dt3_id, &ldquo;c_name&rdquo;, offset, H5T_NATIVE_DOUBLE, error)
```

```
!
```

```
! Write data by fields in the datatype. Fields order is not important.
```

```
!
```

```
CALL h5dwrite_f(dset_id, dt3_id, c, data_dims, error, xfer_prp = plist_id)
```

```
CALL h5dwrite_f(dset_id, dt2_id, b, data_dims, error, xfer_prp = plist_id)
```

```
CALL h5dwrite_f(dset_id, dt1_id, a, data_dims, error, xfer_prp = plist_id)</pre></td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 20. Create and write a dataset with a compound datatype in
```

```
Fortran</b>
```

```
<hr color="green" size="3" /></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

```
<!-- NEW PAGE -->
```

```
<h4>6.4.3.2.1.3. Reading Datasets with Compound Datatypes</h4>
```

Reading datasets with compound datatypes may be a

challenge. For general applications there is no way to

know *a priori* the corresponding C structure.

Also, C structures cannot be allocated on the fly during discovery

of the dataset's datatype. For general C, C++, Fortran

and Java application the following steps will be required

to read and to interpret data from the dataset with

compound datatype:

```
<dl>
<dt>
<ol>
  <li>Get the identifier of the compound datatype in the file
    with the H5Dget_type call</li>
  <li>Find the number of the compound datatype members
    with the H5Tget_nmembers call</li>
  <li>Iterate through compound datatype members</li>
</ol>
<dd>
  <ul>
    <li>Get member class with the
      H5Tget_member_class call</li>
    <li>Get member name with the
      H5Tget_member_name call</li>
    <li>Check class type against predefined classes</li>

  <ul>
    <li><code>H5T_INTEGER</code></li>
    <li><code>H5T_FLOAT</code></li>
    <li><code>H5T_STRING</code></li>
    <li><code>H5T_BITFIELD</code></li>
    <li><code>H5T_OPAQUE</code></li>
    <li><code>H5T_COMPOUND</code></li>
    <li><code>H5T_REFERENCE</code></li>
    <li><code>H5T_ENUM</code></li>
    <li><code>H5T_VLEN</code></li>
    <li><code>H5T_ARRAY</code></li>
  </ul>
</ul>
```

```

<li>If class is <code>H5T_COMPOUND</code>,
  then go to step 2 and repeat all steps under
  step 3. If class is not <code>H5T_COMPOUND</code>,
  then a member is of an atomic class and can be
  read to a corresponding buffer after discovering
  all necessary information specific to each atomic
  type (e.g. size of the integer or floats, super
  class for enumerated and array datatype,
  and it sizes, etc.)</li>
</ul>
</dd>
</dl>

```

The examples below show how to read a dataset with a known compound datatype.

The first example below shows the steps needed to read data of a known structure. First, build a memory datatype the same way it was built when the dataset was created, and then second use the datatype in a `H5Dread` call.

```
<!-- NEW PAGE -->
```

```
<table width="600" cellpadding="0" align="center">
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<hr color="green" size="3"/>
```

```
<pre>
```

```
typedef struct s1_t {
```

```
    int  a;
```

```

float b;

double c;
} s1_t;

s1_t *data;

...

s1_tid = H5Tcreate(H5T_COMPOUND, sizeof(s1_t));
H5Tinsert(s1_tid, &ldquo;a_name&rdquo;, HOFFSET(s1_t, a), H5T_NATIVE_INT);
H5Tinsert(s1_tid, &ldquo;b_name&rdquo;, HOFFSET(s1_t, b), H5T_NATIVE_FLOAT);
H5Tinsert(s1_tid, &ldquo;c_name&rdquo;, HOFFSET(s1_t, c), H5T_NATIVE_DOUBLE);
...

dataset_id = H5Dopen(file_id, &ldquo;SDScompound.h5&rdquo;, H5P_DEFAULT);
...

data = (s1_t *) malloc (sizeof(s1_t)*LENGTH);
H5Dread(dataset_id, s1_tid, H5S_ALL, H5S_ALL, H5P_DEFAULT, data);</pre>

```

</tr>

<tr><td><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td align="left">

Example 21. Read a dataset using a memory datatype

<!-- used to be Figure 25f -->

<hr color="green" size="3"/></td>

</tr>

</table>

Instead of building a memory datatype, the application could use the

`H5Tget_native_type` function. See the example below.</p>

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
typedef struct s1_t {
    int  a;
    float b;
    double c;
} s1_t;

s1_t *data;
hid_t file_s1_t, mem_s1_t;
...
dataset_id = H5Dopen(file_id, &ldquo;SDScompound.h5&rdquo;, H5P_DEFAULT);
/* Discover datatype in the file */
file_s1_t = H5Dget_type(dataset_id);
/* Find corresponding memory datatype */
mem_s1_t = H5Tget_native_type(file_s1_t, H5T_DIR_DEFAULT);

...
data = (s1_t *) malloc (sizeof(s1_t)*LENGTH);
H5Dread (dataset_id, mem_s1_t, H5S_ALL, H5S_ALL, H5P_DEFAULT, data);</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      <b>Example 22. Read a dataset using <code>H5Tget_native_type</code>
      <!-- used to be Figure 25g --></b>

```



```

        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

<p>The example below shows how to read just one float member of a compound datatype.</p>

```

<!-- used to be Example 25h -->

```

```

<table width="600" cellpadding="0" align="center">
    <tr valign="top">
        <td align="left">
            <hr color="green" size="3"/>
            <pre>
typedef struct s1_t {
    float b;
} sf_t;

sf_t *data;

...

sf_tid = H5Tcreate(H5T_COMPOUND, sizeof(sf_t));
H5Tinsert(sf_tid, &ldquo;b_name&rdquo;, HOFFSET(sf_t, b), H5T_NATIVE_FLOAT);
...

dataset_id = H5Dopen(file_id, &ldquo;SDScompound.h5&rdquo;, H5P_DEFAULT);

```

```

...
data = (sf_t *) malloc (sizeof(sf_t)*LENGTH);
H5Dread(dataset_id, sf_tid, H5S_ALL, H5S_ALL, H5P_DEFAULT, data);</pre> </td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 23. Read one floating point member of a compound datatype
    <!-- used to be Figure 25h --></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

The example below <!-- used to be Figure 25i --> shows how to read float and double members of a compound datatype into a structure that has those fields in a different order. Please notice that <code>H5Tinsert</code> calls can be used in an order different from the order of the structure's members.</p>

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>

```

```

typedef struct s1_t {
    double c;
    float b;
} sdf_t;

sdf_t *data;

...

sdf_tid = H5Tcreate(H5T_COMPOUND, sizeof(sdf_t));
H5Tinsert(sdf_tid, &ldquo;b_name&rdquo;, HOFFSET(sdf_t, b), H5T_NATIVE_FLOAT);
H5Tinsert(sdf_tid, &ldquo;c_name&rdquo;, HOFFSET(sdf_t, c), H5T_NATIVE_DOUBLE);
...

dataset_id = H5Dopen(file_id, &ldquo;SDScompound.h5&rdquo;, H5P_DEFAULT);
...

data = (sdf_t *) malloc (sizeof(sdf_t)*LENGTH);
H5Dread(dataset_id, sdf_tid, H5S_ALL, H5S_ALL, H5P_DEFAULT, data);</pre></td>

```

</tr>

<tr><td><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td align="left">

Example 24. Read float and double members of a compound datatype

<!-- used to be Figure 25i -->

<hr color="green" size="3"/></td>

</tr>

</table>

<!-- NEW PAGE -->

<h4>6.4.3.2.2. Array</h4>

Many scientific datasets have multiple measurements for each point in a space. There are several natural ways to represent this data, depending on the variables and how they are used in computation. See the table and the figure below.

Table 20. Representing data with multiple measurements		
<hr/>		
Storage Strategy		
Stored as		
Remarks		
<hr/>		
Multitple planes		
Several datasets with identical dataspace		
This is optimal when variables are accessed individually, or when often uses only selected variables.		
<hr/>		
Additional dimension		

<p>One dataset, the last "dimension";</p> <p>is a vector of variables</p> <p> </p> <p>This can give good performance, although</p> <p>selecting only a few variables may be slow. This may not reflect the</p> <p>science.</p>	<hr/>
<p>Record with multiple values</p> <p> </p> <p>One dataset with compound datatype</p> <p> </p> <p>This enables the variables to be read all</p> <p>together or selected. Also handles "vectors" of</p> <p>heterogenous data.</p>	<hr/>
<p>Vector or Tensor value</p> <p> </p> <p>One dataset, each data element is a small</p> <p>array of values.</p> <p> </p> <p>This uses the same amount of space as</p> <p>the previous two, and may represent the science model better.</p>	<hr/>

```

<!-- NEW PAGE -->
<table width="400" cellspacing="0" align="center">
  <tr><td colspan="3"><hr color="green" size="3" /></td></tr>
  <tr valign="top">
    <td align="left">
      </td>
      <td align="center">
        </td>
      </tr>
  <tr><td colspan="3"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      </td>
      <td align="center">
        </td>
      </tr>
  <tr><td colspan="3"><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" colspan="3">
      <b>Figure 13. Representing data with multiple measurements</b>
    </td>
  </tr>
  <tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

The HDF5 `H5T_ARRAY` datatype defines the data element to be a homogeneous, multi-dimensional array. See Figure 13d above. The elements of the array can be any HDF5 datatype (including compound and array), and the size of the datatype is the total size of the array.

A dataset of array datatype cannot be subdivided for I/O within the data element: the entire array of the data element must be transferred. If the data elements need to be accessed separately, e.g., by plane, then the array datatype should not be used. The table below formerly Table 22 shows advantages and disadvantages of various storage methods.

NEW PAGE

Table 21. Storage method advantages and disadvantages				
<hr/>				
Method		Advantages		Disadvantages
a) Multiple Datasets				

<p>Easy to access each plane, can select any plane(s)</p> <p> </p> <p>Less efficient to access a 'column' through the planes</p>

<hr style="border: 1px solid green;"/>				
<p>b) N+1 Dimension</p> <p> </p> <p>All access patterns supported</p> <p> </p> <p>Must be homogeneous datatype</p> <p>The added dimension may not make sense in the scientific model</p>				

<hr style="border: 1px solid green;"/>				
<p>c) Compound Datatype</p> <p> </p> <p>Can be heterogenous datatype</p> <p> </p> <p>Planes must be named, selection is by plane</p> <p>Not a natural representation for a matrix</p>				

<hr style="border: 1px solid green;"/>				
<p>d) Array</p> <p> </p> <p>A natural representation for vector or tensor data</p> <p> </p> <p>Cannot access elements separately (no access by plane)</p>				


```

</tr>
<tr><td colspan="5"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

An array datatype may be multi-dimensional with 1 to `H5S_MAX_RANK` (the maximum rank of a dataset is currently 32) dimensions. The dimensions can be any size greater than 0, but unlimited dimensions are not supported (although the datatype can be a variable-length datatype).

An array datatype is created with the `H5Tarray_create` call, which specifies the number of dimensions, the size of each dimension, and the base type of the array. The array datatype can then be used in any way that any datatype object is used. The example below formerly Figure 27 shows the creation of a datatype that is a two-dimensional array of native integers, and this is then used to create a dataset. Note that the dataset can be a dataspace that is any number and size of dimensions. The figure below formerly Figure 28 shows the layout in memory assuming that the native integers are 4 bytes. Each data element has 6 elements, for a total of 24 bytes.

```

<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>

```

```

    <pre>
hid_t    file, dataset;
hid_t    datatype, dataspace;
hsize_t  adims[] = {3, 2};

datatype = H5Tarray_create(H5T_NATIVE_INT, 2, adims, NULL);

dataset = H5Dcreate(file, datasetname, datatype, dataspace,
    H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 25. Create a two-dimensional array datatype
        <!-- formerly Figure 27 --></b>
        <hr color="green" size="3" /></td>
    </tr>
</table>
<br />
<br />

```

```

<table width="600" cellpadding="0" align="center">
    <tr valign="top">
        <td align="center">
            <hr color="green" size="3" />
            
        </td>
    </tr>
    <tr><td><hr color="green" size="1" /></td></tr>

```

```

<tr valign="top">
  <td align="left" >
    <b>Figure 14. Memory layout of a two-dimensional array datatype
    <!-- formerly Figure 28 --></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

```

<!-- NEW PAGE -->
<a name="VariableLengthDatatypes">
<h4>6.4.3.2.3. Variable-length Datatypes</h4></a>

```

A variable-length (VL) datatype is a one-dimensional sequence of a datatype which are not fixed in length from one dataset location to another, i.e., each data element may have a different number of members. Variable-length datatypes cannot be divided, the entire data element must be transferred.

VL datatypes are useful to the scientific community in many different ways, possibly including:

- Ragged arrays**: Multi-dimensional ragged arrays can be implemented with the last (fastest changing) dimension being ragged by using a VL datatype as the type of the element stored.

- Fractal arrays**: A nested VL datatype can be used to implement ragged arrays of ragged arrays, to whatever nesting depth is required for the user.

- Polygon lists**: A common storage requirement is to

- efficiently store arrays of polygons with different numbers of vertices. A VL datatypes can be used to efficiently and succinctly describe an array of polygons with different numbers of vertices.
- Character strings: Perhaps the most common use of VL datatypes will be to store C-like VL character strings in dataset elements or as attributes of objects.
- Indices, e.g. of objects within the file: An array of VL object references could be used as an index to all the objects in a file which contain a particular sequence of dataset values.
- Object Tracking: An array of VL dataset region references can be used as a method of tracking objects or features appearing in a sequence of datasets.

A VL datatype is created by calling `H5Tvlen_create` which specifies the base datatype. The first example below <!-- formerly Figure 29 --> shows an example of code that creates a VL datatype of unsigned integers. Each data element is a one-dimensional array of zero or more members and is stored in the `hvl_t` structure. See the second example below. <!-- formerly Figure 30 --></p>

```
<table width="600" cellpadding="0" align="center">
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<hr color="green" size="3"/>
```

```
<pre>
```

```
tid1 = H5Tvlen_create (H5T_NATIVE_UINT);
```

```
dataset=H5Dcreate(fid1, "Dataset1", tid1, sid1, H5P_DEFAULT,
```

```

        H5P_DEFAULT, H5P_DEFAULT);</pre> </td>
    </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 26. Create a variable-length datatype of unsigned integers
        <!-- formerly Figure 29 --></b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />
<br />

```

```

<table width="600" cellpadding="0" align="center">
    <tr valign="top">
        <td align="left">
            <hr color="green" size="3"/>
            <pre>
typedef struct {
    size_t len; /* Length of VL data (in base type units) */
    void *p; /* Pointer to VL data */
} hvl_t;</pre></td>
    </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 27. Data element storage for members of the VL datatype
        <!-- formerly Figure 30 --></b>
        <hr color="green" size="3"/></td>
    </tr>

```

</tr>
</table>

<!-- NEW PAGE -->

<p>The first example below <!-- formerly Figure 31 --> shows how the VL data is written. For each of the 10 data elements, a length and data buffer must be allocated. Below the two examples is a figure <!-- formerly Figure 33 --> that shows how the data is laid out in memory. </p>

<p>An analogous procedure must be used to read the data. See the second example below.

An appropriate array of <code>vl_t</code> must be allocated, and the data read. It is then traversed one data element at a time. The <code>H5Dvlen_reclaim</code> call frees the data buffer for the buffer. With each element possibly being of different sequence lengths for a dataset with a VL datatype, the memory for the VL datatype must be dynamically allocated. Currently there are two methods of managing the memory for VL datatypes: the standard C malloc/free memory allocation routines or a method of calling user-defined memory management routines to allocate or free memory (set with <code>H5Pset_vlen_mem_manager</code>). Since the memory allocated when reading (or writing) may be complicated to release, the <code>H5Dvlen_reclaim</code> function is provided to traverse a memory buffer and free the VL datatype information without leaking memory.</p>

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
hvl_t wdata[10];      /* Information to write */

/* Allocate and initialize VL data to write */
for(i=0; i < 10; i++) {
  wdata[i].p = malloc((i+1)*sizeof(unsigned int));
  wdata[i].len = i+1;
  for(j=0; j<(i+1); j++)
    ((unsigned int *)wdata[i].p)[j]=i*10+j;
}

ret=H5Dwrite(dataset, tid1, H5S_ALL, H5S_ALL, H5P_DEFAULT, wdata);</pre> </td>
    </tr>
    <tr><td><hr color="green" size="1" /></td></tr>
    <tr valign="top">
      <td align="left">
        <b>Example 28. Write VL data</b>
        <!-- formerly Figure 31 --></b>
        <hr color="green" size="3"/></td>
      </tr>
</table>
<br />
<br />

```

```

<table width="600" cellspacing="0" align="center">

```

```

<tr valign="top">
  <td align="left">
    <hr color="green" size="3"/>
    <pre>
hvl_t rdata[SPACE1_DIM1];
ret=H5Dread(dataset, tid1, H5S_ALL, H5S_ALL, xfer_pid, rdata);

for(i=0; i<SPACE1_DIM1; i++) {
  printf("&ldquo;%d: len %d &rdquo;;",rdata[i].len);
  for(j=0; j<rdata[i].len; j++) {
    printf("&ldquo; value: %u\n&rdquo;;",((unsigned int *)rdata[i].p)[j]);
  }
}
ret=H5Dvlen_reclaim(tid1, sid1, xfer_pid, rdata);</pre> </td>
  </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 29. Read VL data
    <!-- formerly Figure 32 --></b>
    <hr color="green" size="3"/></td>
</tr>
</table>
<br />
<br />

```

<!-- NEW PAGE -->

```

<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">

```

```

<hr color="green" size="3"/>

</td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 15. Memory layout of a VL datatype
    <!-- formerly Figure 33 --></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

The user program must carefully manage these relatively complex data structures.

The `H5Dvlen_reclaim` function performs a standard traversal, freeing all the data. This function analyzes the datatype and dataspace objects, and visits each VL data element, recursing through nested types. By default, the system `free` is called for the pointer in each `vl_t`. Obviously, this call assumes that all of this memory was allocated with the system `malloc`.

The user program may specify custom memory manager routines, one for allocating and one for freeing. These may be set with the `H5Pvlen_mem_manager`, and must have the following prototypes:

```

<ul>
  <li><code>typedef void *(*H5MM_allocate_t)(size_t size, void *info);</code> </li>

```

```
<li><code>typedef void (*H5MM_free_t)(void *mem, void *free_info);</code> </li>
</ul>
```

The utility function `H5Dget_vlen_buf_size` checks the number of bytes required to store the VL data from the dataset. This function analyzes the datatype and dataspace object to visit all the VL data elements, to determine the number of bytes required to store the data for the in the destination storage (memory). The `size` value is adjusted for data conversion and alignment in the destination.

```
<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="NonNumDtypes">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>
```

```
<a name="NonNumDtypes">
<h3 class=pagebefore>6.5. Other Non-numeric Datatypes</h3>
</a>
```

Several datatype classes define special types of objects.

6.5.1. Strings

Text data is represented by arrays of characters called strings. Because C and Fortran terminate strings differently, the library can read and write strings in several ways. The important questions that an application needs to answer to read strings correctly are the following:

- What is the size of the string?
- How is the string terminated?

See the entry for `H5Tset_strpad` in the *HDF5 Reference Manual* for more information on how to specify a particular style of string padding and terminating.

In the rest of this section, we will look at some of the common ways strings might be stored.

The figures below show different ways that the strings "Four score" and "seven years ago" might be stored in a C environment. By C environment, we mean that the disk and memory versions of the dataset terminate the strings with a NULL, \0. This also means that the length of a string is one less than the size of the string: the size counts the NULL. Note that the single quotation marks have been added for clarity in the examples in this section and would not be stored in an actual dataset. See ["Strings in Mixed Environments"](#stringsInMixedEnvironments) at the end of this section for more information.

The figure below shows a string stored in a dataset in a one-dimensional array using 27 elements. The `H5T_NATIVE_CHAR` datatype is used. Each character of the string is stored in an element of the dataset. The result is a block of text data that gives little indication of any structure in the text.

```
<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
```

```
<table align="center" width="100%">
  <tr>
    <td colspan="2">
      <table align="center" border="1" width="95%">
        <tr>
          <td width="12.5%" align="center">0</td>
          <td width="12.5%" align="center">1</td>
          <td width="12.5%" align="center">2</td>
          <td width="12.5%" align="center">3</td>
          <td width="12.5%" align="center">4</td>
          <td width="12.5%" align="center">5</td>
          <td width="12.5%" align="center">6</td>
          <td width="12.5%" align="center">7</td>
        </tr>
        <tr>
          <td align="center">&lsquo;f&rsquo;</td>
          <td align="center">&lsquo;o&rsquo;</td>
          <td align="center">&lsquo;u&rsquo;</td>
```

```

<td align="center">&lsquo;r&rsquo;</td>
<td align="center">&lsquo;&nbsp;&rsquo;</td>
<td align="center">&lsquo;s&rsquo;</td>
<td align="center">&lsquo;c&rsquo;</td>
<td align="center">&lsquo;o&rsquo;</td>
</tr>
</table>
</td>
</tr>
<tr><td colspan="3">&nbsp;</td></tr>
</table>

```

```

<table align="center" width="100%">
<tr>
<td colspan="2">
<table align="center" border="1" width="95%">
<tr>
<td width="12.5%" align="center">8</td>
<td width="12.5%" align="center">9</td>
<td width="12.5%" align="center">10</td>
<td width="12.5%" align="center">11</td>
<td width="12.5%" align="center">12</td>
<td width="12.5%" align="center">13</td>
<td width="12.5%" align="center">14</td>
<td width="12.5%" align="center">15</td>
</tr>
<tr>
<td align="center">&lsquo;r&rsquo;</td>
<td align="center">&lsquo;e&rsquo;</td>
<td align="center">&lsquo;\0&rsquo;</td>
<td align="center">&lsquo;s&rsquo;</td>

```

```

<td align="center">&lsquo;e&rsquo;</td>
<td align="center">&lsquo;v&rsquo;</td>
<td align="center">&lsquo;e&rsquo;</td>
<td align="center">&lsquo;n&rsquo;</td>
</tr>
</table>
</td>
</tr>
<tr><td colspan="3">&nbsp;</td></tr>
</table>

<table align="center" width="100%">
<tr>
<td colspan="2">
<table align="center" border="1" width="95%">
<tr>
<td width="12.5%" align="center">16</td>
<td width="12.5%" align="center">17</td>
<td width="12.5%" align="center">18</td>
<td width="12.5%" align="center">19</td>
<td width="12.5%" align="center">20</td>
<td width="12.5%" align="center">21</td>
<td width="12.5%" align="center">22</td>
<td width="12.5%" align="center">23</td>
</tr>
<tr>
<td align="center">&lsquo;&nbsp;&rsquo;</td>
<td align="center">&lsquo;y&rsquo;</td>
<td align="center">&lsquo;e&rsquo;</td>
<td align="center">&lsquo;a&rsquo;</td>
<td align="center">&lsquo;r&rsquo;</td>

```

```

<td align="center">&lsquo;s&rsquo;</td>
<td align="center">&lsquo;&nbsp;&rsquo;</td>
<td align="center">&lsquo;a&rsquo;</td>
</tr>
</table>
</td>
</tr>
<tr><td colspan="3">&nbsp;</td></tr>
</table>

```

```

<table align="center" width="100%">
<tr>
<td colspan="2">
<table align="center" border="1" width="95%">
<tr>
<td width="12.5%" align="center">24</td>
<td width="12.5%" align="center">25</td>
<td width="12.5%" align="center">26</td>
<td width="12.5%" align="center">27</td>
<td width="12.5%" align="center">28</td>
<td width="12.5%" align="center">29</td>
<td width="12.5%" align="center">30</td>
<td width="12.5%" align="center">31</td>
</tr>
<tr>
<td align="center">&lsquo;g&rsquo;</td>
<td align="center">&lsquo;o&rsquo;</td>
<td align="center">&lsquo;\0&rsquo;</td>
<td align="center">&nbsp;</td>
<td align="center">&nbsp;</td>
<td align="center">&nbsp;</td>

```

```

<td align="center">&nbsp;</td>
<td align="center">&nbsp;</td>
</tr>
</table>
</td>
</tr>
</table>

```

```

<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 16. A string stored as one-character elements in a
    one-dimensional array
    <!-- formerly part of Figure 34 --></b><hr color="green" size="3" /></td>
  </tr>
</table>
<br />

```

The figure below shows how these strings might be stored using a **fixed-length** datatype. This one-dimensional array uses the `H5T_STRING` datatype. The dataset reserves space for a specified number of characters in each string although some strings may be shorter. In the figure below, the size is set to 20. This approach is simple and usually fast to access, but this approach can waste storage space if the lengths of the strings vary. The single quotation marks are used to show the 20 characters included in each dataset element.

```

<table width="600" cellpadding="0" align="center">
  <tr valign="top">

```


The figure below illustrates how these strings might be stored using a **variable-length datatype**. This can be done using the mechanisms described in the ["Variable-length Datatypes"](#VariableLengthDatatypes) section above.

The program would use `vl_t` structures to write and read the data. The dataset is a one-dimensional array with two elements, and each element is a variable-length string.

This is the same result as the strings stored in fixed-length elements in the figure above except that the first element of the array will need only 11 bytes for storage instead of 20, and the second element will need only 16 bytes instead of 20. Note that the single quotation marks are used to show the characters in each dataset element.

```
<table width="600" cellpadding="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="center">
```

```
      <hr color="green" size="3"/>
```

```
<table align="center" width="100%">
```

```
<tr>
```

```
<td colspan="2">
```

```
  <table align="center" border="1" width="70%">
```

```
    <tr>
```

```
      <td width="50%" align="center">0</td>
```

```
      <td width="50%" align="center">1</td>
```

```
    </tr>
```

```
  </tr>
```

```

<td align="center">&lsquo;Four&nbsp;score\0&rsquo;</td>
<td align="center">&lsquo;seven&nbsp;years&nbsp;ago\0&rsquo;</td>
</tr>
</table>
</td>
</tr>
</table>

<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 18. Strings stored as variable-length dataset elements
    <!-- formerly part of Figure 34 --></b><hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

An alternate way to set up variable-length dataset elements is to set the size of the string datatype class to `H5T_VARIABLE`. The example below <!-- formerly Figure 35 --> shows a declaration of a datatype of type `H5T_C_S1` which is set to `H5T_VARIABLE`. The HDF5 Library automatically translates between this and the `vl_t` structure. Note that the `H5T_VARIABLE` size can only be used with string datatypes.

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>

```

```

    <pre>
tid1 = H5Tcopy (H5T_C_S1);

ret = H5Tset_size (tid1, H5T_VARIABLE);</pre></td>
    </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 30. Set the string datatype size to <code>H5T_VARIABLE</code>
        <!-- formerly Figure 35 --></b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

Variable-length strings can be read into C strings
(in other words, pointers to zero
terminated arrays of `char`). See the example below.

```

<table width="650" cellspacing="0" align="center">
    <tr valign="top">
        <td align="left">
            <hr color="green" size="3"/>
            <pre>
char *rdata[SPACE1_DIM1];

ret=H5Dread(dataset, tid1, H5S_ALL, H5S_ALL, xfer_pid, rdata);

```

```
for(i=0; i<SPACE1_DIM1; i++) {
    printf("&ldquo;%d: len: %d, str is: %s\n&rdquo;;", i, strlen(rdata[i]),rdata[i]);
}
```

```
ret=H5Dvlen_reclaim(tid1, sid1, xfer_pid, rdata);</pre></td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 31. Read variable-length strings into C strings
```

```
<!-- formerly Figure 36 --></b>
```

```
<hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

```
<!--
```

3.23.2012. I commented out the "Strings in Mixed Environments" section below.

I have spent too many GMQS hours and have to stop charging GMQS for awhile. MEE.

```
<a name="stringsInMixedEnvironments">
```

```
<b>Strings in Mixed Environments</b></a>
```

```
<p>In the figures above, the strings are terminated with NULLs.
```

```
Suppose in another scenario that the strings were stored on disk and
were not terminated with NULLs, and suppose that the users of the data
would be using applications that expected strings to be terminated with
NULLs? What APIs might an application use to properly handle the strings?
```

```
</p>
```

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
```

The HDF Group

```
<td align="left" >
```

```
<b>Figure 17???. Strings stored as fixed-length dataset elements
and not terminated with NULLs
```

```
</b><hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

```
<p>
```

```
<p>??????? rewrite for this new example
```

The figure below shows how these strings might be stored using a **fixed-length** datatype. This one-dimensional array uses the `H5T_STRING` datatype. The dataset reserves space for a specified number of characters in each string although some strings may be shorter. In the example below, the size is set to 20. This approach is simple and usually fast to access, but this approach can waste storage space if the lengths of the strings vary. The single quotation marks are used to show the 20 characters included in each dataset element.

```
<table width="600" cellpadding="0" align="center">
```

```
<tr valign="top">
```

```
<td align="center">
```

```
<hr color="green" size="3"/>
```

```
<table align="center" width="100%">
```

```
<tr>
```

```
<td colspan="2">
```

```
<table align="center" border="1" width="70%">
```


-->

<h4>6.5.2. Reference</h4>

<p>In HDF5, objects (i.e. groups, datasets, and committed datatypes) are usually accessed by name. There is another way to access stored objects - by reference. There are two reference datatypes: object reference and region reference. Object reference objects are created with `H5Rcreate` and other calls (cross reference). These objects can be stored and retrieved in a dataset as elements with reference datatype. The first example below <!-- formerly Figure 37 --> shows an example of code that creates references to four objects, and then writes the array of object references to a dataset. The second example below <!-- formerly Figure 38 --> shows a dataset of datatype reference being read and one of the reference objects being dereferenced to obtain an object pointer.</p>

<p>In order to store references to regions of a dataset, the datatype should be `H5T_REGION_OBJ`. Note that a data element must be either an object reference or a region reference: these are different types and cannot be mixed within a single array.</p>

<p>A reference datatype cannot be divided for I/O: an element is read or written completely.</p>

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
dataset=H5Dcreate(fid1, &ldquo;Dataset3&rdquo;, H5T_STD_REF_OBJ, sid1,
    H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

/* Create reference to dataset */
ret = H5Rcreate(&amp;wbuf[0], fid1,&ldquo;/Group1/Dataset1&rdquo;, H5R_OBJECT, -1);

/* Create reference to dataset */
ret = H5Rcreate(&amp;wbuf[1], fid1, &ldquo;/Group1/Dataset2&rdquo;, H5R_OBJECT, -1);

/* Create reference to group */
ret = H5Rcreate(&amp;wbuf[2], fid1, &ldquo;/Group1&rdquo;, H5R_OBJECT, -1);

/* Create reference to committed datatype */
ret = H5Rcreate(&amp;wbuf[3], fid1, &ldquo;/Group1/Datatype1&rdquo;, H5R_OBJECT, -1);

/* Write selection to disk */

ret=H5Dwrite(dataset, H5T_STD_REF_OBJ, H5S_ALL, H5S_ALL, H5P_DEFAULT, wbuf);</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      <b>Example 32. Create object references and write to a dataset
      <!-- formerly Figure 37 --></b>

```

```

    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
<br />

```

```

<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
rbuf = malloc(sizeof(hobj_ref_t)*SPACE1_DIM1);

/* Read selection from disk */
ret=H5Dread(dataset, H5T_STD_REF_OBJ, H5S_ALL, H5S_ALL, H5P_DEFAULT, rbuf);

/* Open dataset object */
dset2 = H5Rdereference(dataset, H5R_OBJECT, &rbuf[0]);</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      <b>Example 33. Read a dataset with a reference datatype
      <!-- formerly Figure 38 --></b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

<!-- NEW PAGE -->

<h4>6.5.3. ENUM</h4>

<p>The enum datatype implements a set of (name, value) pairs, similar to C/C++ enum. The values are currently limited to native integer datatypes. Each name can be the name of only one value, and each value can have only one name. </p>

<p>The data elements of the ENUMERATION are stored according to the datatype, e.g., as an array of integers. The example below <!-- formerly Figure 39 --> shows an example of how to create an enumeration with five elements. The elements map symbolic names to 2-byte integers. See the table below.</p>

```
<table width="600" cellpadding="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="left">
```

```
      <hr color="green" size="3"/>
```

```
      <pre>
```

```
hid_t hdf_en_colors = H5Tcreate(H5T_ENUM, sizeof(short));
```

```
short val;
```

```
  H5Tenum_insert(hdf_en_colors, &ldquo;RED&rdquo;, (val=0,&amp;val));
```

```
  H5Tenum_insert(hdf_en_colors, &ldquo;GREEN&rdquo;, (val=1,&amp;val));
```

```
  H5Tenum_insert(hdf_en_colors, &ldquo;BLUE&rdquo;, (val=2,&amp;val));
```

```
  H5Tenum_insert(hdf_en_colors, &ldquo;WHITE&rdquo;, (val=3,&amp;val));
```

```
  H5Tenum_insert(hdf_en_colors, &ldquo;BLACK&rdquo;, (val=4,&amp;val));
```

```
  H5Dcreate(fileid, datasetname, hdf_en_colors, spaceid, H5P_DEFAULT,
```

```
    H5P_DEFAULT, H5P_DEFAULT);</pre></td>
```

```

    </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 34. Create an enumeration with five elements
        <!-- formerly Figure 39 --></b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />
<br />

<table width="200" cellspacing="0" align="center" cellpadding="0">
    <tr valign="bottom">
        <td colspan="2" align="left" valign="bottom">
            <b>Table 22. An enumeration<br />with five elements</b>
            <!-- formerly Table 23 --></td>
        </tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
<tr valign="top">
    <td width="50%"><b>Name</b></td>
    <td width="50%"><b>Value</b></td>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>RED</td>
    <td>0</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>GREEN</td>

```

```

    <td>1</td>
  </tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>BLUE</td>
  <td>2</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>WHITE</td>
  <td>3</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>BLACK</td>
  <td>4</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

```

<p>The figure below <!-- formerly Figure 40 -->shows how an array of eight
values might be stored. Conceptually,
the array is an array of symbolic names [BLACK, RED, WHITE, BLUE, ...]. See
item a in the figure below. <!-- formerly Figure 40a -->

```

```

These are stored as the values and are short integers. So, the first 2 bytes
are the value associated with &ldquo;BLACK&rdquo;, which is the number 4,
and so on. See item b in the figure below. <!-- formerly Figure 40b --></p>

```

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <table align="center" width="100%">
        <tr>
          <td align="center">a) Logical data to be written -
            eight elements</td>
        </tr>
      </table>
    </td>
  </tr>
  <tr>
    <td align="center">
      <table>
        <tr>
          <td width="50" align="center">Index</td>
          <td width="135" align="center">Name</td>
        </tr>
      </table>
      <table border="1">
        <tr>
          <td width="30" align="center">0</td>
          <td width="130" align="left">:BLACK</td>
        </tr>
        <tr>
          <td width="30" align="center">1</td>
          <td width="130" align="left">RED</td>
        </tr>
        <tr>
          <td width="30" align="center">2</td>
          <td width="130" align="left">WHITE</td>
        </tr>
      </table>
    </td>
  </tr>
</table>

```

```

</tr>
<tr>
  <td width="30" align="center">3</td>
<td width="130" align="left">BLUE</td>
</tr>
<tr>
  <td width="30" align="center">4</td>
<td width="130" align="left">RED</td>
</tr>
<tr>
  <td width="30" align="center">5</td>
<td width="130" align="left">WHITE</td>
</tr>
<tr>
  <td width="30" align="center">6</td>
<td width="130" align="left">BLUE</td>
</tr>
<tr>
  <td width="30" align="center">7</td>
<td width="130" align="left">GREEN</td>
</tr>
</table>
</td>
</tr>
<tr><td>&nbsp;</td></tr>
<tr>
  <td align="center"></td>
</tr>
<tr>
  <td align="center">b) The storage layout. Total size of the
    array is 16 bytes, 2 bytes per element.

```



```

</td>
</tr>

</table>
</td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 19. Storing an enum array
    <!-- formerly Figure 40 --></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

The order that members are inserted into an enumeration type is unimportant; the important part is the associations between the symbol names and the values. Thus, two enumeration datatypes will be considered equal if and only if both types have the same symbol/value associations and both have equal underlying integer datatypes. Type equality is tested with the `H5Tequal` function.

If a particular architecture type is required, a little-endian or big-endian datatype for example, use a native integer datatype as the ENUM base datatype and use `H5Tconvert` on values as they are read from or written to a dataset.

<!-- NEW PAGE -->

6.5.4. Opaque

In some cases, a user may have data objects that should be stored and retrieved as blobs with no attempt to interpret them. For example, an application might wish to store an array of encrypted certificates which are 100 bytes long.

While an arbitrary block of data may always be stored as bytes, characters, integers, or whatever, this might mislead programs about the meaning of the data. The opaque datatype defines data elements which are uninterpreted by HDF5. The opaque data may be labeled with `H5Tset_tag` with a string that might be used by an application. For example, the encrypted certificates might have a tag to indicate the encryption and the certificate standard.

6.5.5. Bitfield

Some data is represented as bits, where the number of bits is not an integral byte and the bits are not necessarily interpreted as a standard type. Some examples might include readings from machine registers (e.g., switch positions), a cloud mask, or data structures with several small integers that should be store in a single byte.

This data could be stored as integers, strings, or enumerations. However, these storage methods would likely result in considerable wasted space. For example, storing a cloud mask with one byte per value would use up to eight times the space of a packed array of bits.

The HDF5 bitfield datatype class defines a data element that is a contiguous sequence of bits, which are stored on disk in a packed array. The programming model is the same as for unsigned integers: the datatype

object is created by copying a predefined datatype, and then the precision, offset, and padding are set.

While the use of the bitfield datatype will reduce storage space substantially, there will still be wasted space if the bitfield as a whole does not match the 1-, 2-, 4-, or 8-byte unit in which it is written. The remaining unused space can be removed by applying the [N-bit filter](10_Datasets.html#N-Bit) to the dataset containing the bitfield data.

<!--

5.6. Time

The HDF5 time datatype defines storage layout for various date and time standards. Currently, only Unix "time" and "timeval" structs are supported. The H5T_UNIX_D32BE (LE) defines storage for 4 bytes (sufficient for the time struct), H5T_UNIX_D64BE (LE) is sufficient for timeval. The data is treated as a single opaque value.

-->

<SCRIPT language="JavaScript">

<!--

document.writeln ("

<div align=right>

(Top)

</div>

");

-->

</SCRIPT>

<h3 class=pagebefore>6.6. Fill Values</h3>

The “fill value” for a dataset is the specification of the default value assigned to data elements that have not yet been written. In the case of a dataset with an atomic datatype, the fill value is a single value of the appropriate datatype, such as ‘0’ or ‘-1.0’. In the case of a dataset with a composite datatype, the fill value is a single data element of the appropriate type. For example, for an array or compound datatype, the fill value is a single data element with values for all the component elements of the array or compound datatype.</p>

The fill value is set (permanently) when the dataset is created.

The fill value is set in the dataset creation properties

<!-- editingComment

[[[

(see chapter ??)

]]]

-->

in the `H5Dcreate` call. Note that the `H5Dcreate` call must also include the datatype of the dataset, and the value provided for the fill value will be interpreted as a single element of this datatype.

The example below <!-- formerly Figure 41 -->shows code which creates a dataset of integers with fill

value -1. Any unwritten data elements will be set to -1.</p>

<!-- NEW PAGE -->

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
hid_t    plist_id;
int filler;

filler = -1;
plist_id = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_fill_value(plist_id, H5T_NATIVE_INT, &filler);

/* Create the dataset with fill value -1's. */
dataset_id = H5Dcreate(file_id, &ldquo;/dset&rdquo;, H5T_STD_I32BE,
  dataspace_id, H5P_DEFAULT, plist_id, H5P_DEFAULT);</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      <b>Example 35. Create a dataset with a fill value of -1
      <!-- formerly Figure 41 --></b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />
<br />

```

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">

```

```

    <hr color="green" size="3"/>
    <pre>
typedef struct s1_t {
    int  a;
    char b;
    double c;
} s1_t;
s1_t    filler;

s1_tid = H5Tcreate (H5T_COMPOUND, sizeof(s1_t));
H5Tinsert(s1_tid, &ldquo;a_name&rdquo;, HOFFSET(s1_t, a), H5T_NATIVE_INT);
H5Tinsert(s1_tid, &ldquo;b_name&rdquo;, HOFFSET(s1_t, b), H5T_NATIVE_CHAR);
H5Tinsert(s1_tid, &ldquo;c_name&rdquo;, HOFFSET(s1_t, c), H5T_NATIVE_DOUBLE);

filler.a = -1;
filler.b = &lsquo;*&rsquo;;
filler.c = -2.0;

plist_id = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_fill_value(plist_id, s1_tid, &amp;filler);

/* Create the dataset with fill value (-1, &lsquo;*&rsquo;, -2.0). */
dataset = H5Dcreate(file, datasetname, s1_tid, space, H5P_DEFAULT,
    plist_id, H5P_DEFAULT);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 36. Create a fill value for a compound datatype
        <!-- formerly Figure 42 --></b>
        <hr color="green" size="3"/></td>

```

```

    </tr>
</table>
<br />

```

The figure above <!-- formerly Figure 42 --> shows how to create a fill value for a compound datatype. The procedure is the same as the previous example except the filler must be a structure with the correct fields. Each field is initialized to the desired fill value.</p>

The fill value for a dataset can be retrieved by reading the dataset creation properties of the dataset and then by reading the fill value with `H5Pget_fill_value`. The data will be read into memory using the storage layout specified by the datatype. This transfer will convert data in the same way as `H5Dread`.

The figure below <!-- formerly Figure 43 --> shows how to get the fill value from the dataset created in Example 33 above.</p>

```

<!-- NEW PAGE -->
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
hid_t plist2;
int filler;

dataset_id = H5Dopen(file_id, &ldquo;/dset&rdquo;, H5P_DEFAULT);

```

```
plist2 = H5Dget_create_plist(dataset_id);
```

```
H5Pget_fill_value(plist2, H5T_NATIVE_INT, &filler);
```

```
/* filler has the fill value, &lsquo;-1&rsquo; */</pre></td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 37. Retrieve a fill value
```

```
<!-- formerly Figure 43 --></b>
```

```
<hr color="green" size="3" /></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

A similar procedure is followed for any datatype. The example below

shows how to

read the fill value for the compound datatype created in an example above

. Note that the program must pass an

element large enough to hold a fill value of the datatype indicated by the

argument to `H5Pget_fill_value`. Also, the program must

understand the datatype in order to interpret its components. This may

be difficult to determine without knowledge of the application that

created the dataset.

```
<table width="600" cellspacing="0" align="center">
```



```

<tr valign="top">
  <td align="left">
    <hr color="green" size="3"/>
    <pre>
char *    fillbuf;
int sz;
dataset = H5Dopen( file, DATASETNAME, H5P_DEFAULT);

s1_tid = H5Dget_type(dataset);

sz = H5Tget_size(s1_tid);

fillbuf = (char *)malloc(sz);

plist_id = H5Dget_create_plist(dataset);

H5Pget_fill_value(plist_id, s1_tid, fillbuf);

printf(&ldquo;filler.a: %d\n&rdquo;,((s1_t *) fillbuf)-&gt;a);
printf(&ldquo;filler.b: %c\n&rdquo;,((s1_t *) fillbuf)-&gt;b);
printf(&ldquo;filler.c: %f\n&rdquo;,((s1_t *) fillbuf)-&gt;c);</pre></td>
  </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 38. Read the fill value for a compound datatype
    <!-- formerly Figure 44 --></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

```
<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="CCDtypes">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>

<!-- NEW PAGE -->
<a name="CCDtypes">
<h3 class=pagebefore>6.7. Complex Combinations of Datatypes</h3>
</a>
```

<p>Several composite datatype classes define collections of other datatypes, including other composite datatypes. In general, a datatype can be nested to any depth, with any combination of datatypes.</p>

<p>For example, a compound datatype can have members that are other compound datatypes, arrays, VL datatypes. An array can be an array of array, an array of compound, or an array of VL. And a VL datatype can be a variable-length array of compound, array, or VL datatypes.</p>


<p>These complicated combinations of datatypes form a logical tree,

with a single root datatype, and leaves which must be atomic datatypes (predefined or user-defined). The figure below <!-- formerly Figure 45 --> shows an example of a logical tree describing a compound datatype constructed from different datatypes.

Recall that the datatype is a description of the layout of storage. The complicated compound datatype is constructed from component datatypes, each of which describe the layout of part of the storage. Any datatype can be used as a component of a compound datatype, with the following restrictions:

- No byte can be part of more than one component datatype (i.e., the fields cannot overlap within the compound datatype)
- The total size of the components must be less than or equal to the total size of the compound datatype

These restrictions are essentially the rules for C structures and similar record types familiar from programming languages. Multiple typing, such as a C union, is not allowed in HDF5 datatypes.

<hr/>  <hr/>
<hr/>

```

<tr valign="top">
  <td align="left" >
    <b>Figure 20. A compound datatype built with
    different datatypes<!-- formerly Figure 45 --></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

6.7.1. Creating a Complicated Compound Datatype

To construct a complicated compound datatype, each component is constructed, and then added to the enclosing datatype description. The example below <!-- formerly Figure 46 --> shows how to create a compound datatype with four members:

```

<ul>
  <li>&ldquo;T1&rdquo;, a compound datatype with three members</li>
  <li>&ldquo;T2&rdquo;, a compound datatype with two members</li>
  <li>&ldquo;T3&rdquo;, a one-dimensional array of integers</li>
  <li>&ldquo;T4&rdquo;, a string</li>
</ul>

```

Below the example code is a figure that shows this datatype as a logical tree. <!-- formerly Figure 47 --> The output of the `h5dump` utility is shown in the example below the figure. <!-- the example was formerly called Figure 48.-->

Each datatype is created as a separate datatype object. Figure 20 below
formerly Figure 49 --> shows
the storage layout for the four individual datatypes. Then the datatypes are
inserted into the outer datatype at an appropriate offset. Figure 21 below
formerly Figure 50 --> shows
the resulting storage layout. The combined record is 89 bytes long.

The Dataset is created using the combined compound datatype. The dataset
is declared to be a 4 by 3 array of compound data. Each data element is an
instance of the 89-byte compound datatype. Figure 22 below
formerly Figure 51 --> shows the layout of
the dataset, and expands one of the elements to show the relative position
of the component data elements.

Each data element is a compound datatype, which can be written or read
as a record, or each field may be read or written individually. The first
field ("T1") is itself a compound datatype with three fields
("T1.a", "T1.b", and "T1.c").
"T1" can be read or written as a record, or individual
fields can be accessed. Similarly, the second field is a compound datatype
with two fields ("T2.f1", "T2.f2").

The third field ("T3") is an array datatype. Thus,
"T3" should be accessed as an array of 40 integers. Array
data can only be read or written as a single element, so all 40
integers must be read or written to the third field. The fourth
field ("T4") is a single string of length 25.

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center">

```
<tr valign="top">
```

```
  <td align="left">
```

```
  <hr color="green" size="3"/>
```

```
  <pre>
```

```
typedef struct s1_t {
```

```
int  a;
```

```
char b;
```

```
double c;
```

```
} s1_t;
```

```
typedef struct s2_t {
```

```
float f1;
```

```
float f2;
```

```
} s2_t;
```

```
hid_t  s1_tid, s2_tid, s3_tid, s4_tid, s5_tid;
```

```
/* Create a datatype for s1 */
```

```
s1_tid = H5Tcreate (H5T_COMPOUND, sizeof(s1_t));
```

```
H5Tinsert(s1_tid, &ldquo;a_name&rdquo;, HOFFSET(s1_t, a), H5T_NATIVE_INT);
```

```
H5Tinsert(s1_tid, &ldquo;b_name&rdquo;, HOFFSET(s1_t, b), H5T_NATIVE_CHAR);
```

```
H5Tinsert(s1_tid, &ldquo;c_name&rdquo;, HOFFSET(s1_t, c), H5T_NATIVE_DOUBLE);
```

```
/* Create a datatype for s2. */
```

```
s2_tid = H5Tcreate (H5T_COMPOUND, sizeof(s2_t));
```

```
H5Tinsert(s2_tid, &ldquo;f1&rdquo;, HOFFSET(s2_t, f1), H5T_NATIVE_FLOAT);
```

```
H5Tinsert(s2_tid, &ldquo;f2&rdquo;, HOFFSET(s2_t, f2), H5T_NATIVE_FLOAT);
```

```
/* Create a datatype for an Array of integers */
```

```
s3_tid = H5Tarray_create(H5T_NATIVE_INT, RANK, dim);
```

```
/* Create a datatype for a String of 25 characters */
```

```

s4_tid = H5Tcopy(H5T_C_S1);
H5Tset_size(s4_tid, 25);

/*
 * Create a compound datatype composed of one of each of these
 * types.
 * The total size is the sum of the size of each.
 */

sz = H5Tget_size(s1_tid) + H5Tget_size(s2_tid) + H5Tget_size(s3_tid)
    + H5Tget_size(s4_tid);

s5_tid = H5Tcreate (H5T_COMPOUND, sz);

/* insert the component types at the appropriate offsets */

H5Tinsert(s5_tid, &ldquo;T1&rdquo;, 0, s1_tid);
H5Tinsert(s5_tid, &ldquo;T2&rdquo;, sizeof(s1_t), s2_tid);
H5Tinsert(s5_tid, &ldquo;T3&rdquo;, sizeof(s1_t)+sizeof(s2_t), s3_tid);
H5Tinsert(s5_tid, &ldquo;T4&rdquo;, (sizeof(s1_t) +sizeof(s2_t)+
    H5Tget_size(s3_tid)), s4_tid);

/*
 * Create the dataset with this datatype.
 */
dataset = H5Dcreate(file, DATASETNAME, s5_tid, space, H5P_DEFAULT,
    H5P_DEFAULT, H5P_DEFAULT);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">

```

Example 39. Create a compound datatype with four members

!-- formerly Figure 46 -->

Figure 21. Logical tree for the compound

datatype with four members!-- formerly Figure 47 -->

```

<tr valign="top">
  <td align="left">
    <hr color="green" size="3"/>
    <pre>
DATATYPE H5T_COMPOUND {
H5T_COMPOUND {
  H5T_STD_I32LE &ldquo;a_name&rdquo;;
  H5T_STD_I8LE &ldquo;b_name&rdquo;;
  H5T_IEEE_F64LE &ldquo;c_name&rdquo;;
} &ldquo;T1&rdquo;;
H5T_COMPOUND {
  H5T_IEEE_F32LE &ldquo;f1&rdquo;;
  H5T_IEEE_F32LE &ldquo;f2&rdquo;;
} &ldquo;T2&rdquo;;
H5T_ARRAY { [10] H5T_STD_I32LE } &ldquo;T3&rdquo;;
H5T_STRING {
  STRSIZE 25;
  STRPAD H5T_STR_NULLTERM;
  CSET H5T_CSET_ASCII;
  CTYPE H5T_C_S1;
} &ldquo;T4&rdquo;;
}</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 40. Output from h5dump for the compound datatype
    <!-- formerly Figure 48 --></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>

```



```
<table width="600" cellspacing="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="center">
```

```
      <hr color="green" size="3"/>
```

```
    <table align="center" border="0" width="100%">
```

```
      <tr>
```

```
        <td valign="middle" align="left">a) Compound type 's1_t', size 16 bytes.</td>
```

```
      </tr>
```

```
    </tr></td>
```

```
  <table border="1" align="center" width="100%">
```

```
<tr>
```

```
  <td valign="middle" align="center" width="25%"><code>Byte 0</code></td>
```

```
  <td valign="middle" align="center" width="25%"><code>Byte 1</code></td>
```

```
  <td valign="middle" align="center" width="25%"><code>Byte 2</code></td>
```

```
  <td valign="middle" align="center" width="25%"><code>Byte 3</code></td>
```

```
</tr>
```

```
<tr>
```

```
  <td valign="middle" align="center"><code>aaaaaaaa</code></td>
```

```
  <td valign="middle" align="center"><code>aaaaaaaa</code></td>
```

```
  <td valign="middle" align="center"><code>aaaaaaaa</code></td>
```

```
  <td valign="middle" align="center"><code>aaaaaaaa</code></td>
```

```
</tr>
```

```
<tr>
```

```
  <td valign="middle" align="center"><code>Byte 4</code></td>
```

```

<td valign="middle" align="center"><code>Byte 5</code></td>
<td valign="middle" align="center"><code>Byte 6</code></td>
<td valign="middle" align="center"><code>Byte 7</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code>bbbbbbbbb</code></td>
<td valign="middle" align="center"><code>&nbsp;</code></td>
<td valign="middle" align="center"><code>&nbsp;</code></td>
<td valign="middle" align="center"><code>&nbsp;</code></td>
</tr>
</table>

<table border="1" align="center" width="100%">
<tr>
<td valign="middle" align="center" width="25%"><code>Byte 8</code></td>
<td valign="middle" align="center" width="25%"><code>Byte 9</code></td>
<td valign="middle" align="center" width="25%"><code>Byte 10</code></td>
<td valign="middle" align="center" width="25%"><code>Byte 11</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code>cccccccc</code></td>
<td valign="middle" align="center"><code>cccccccc</code></td>
<td valign="middle" align="center"><code>cccccccc</code></td>
<td valign="middle" align="center"><code>cccccccc</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code>Byte 12</code></td>
<td valign="middle" align="center"><code>Byte 13</code></td>
<td valign="middle" align="center"><code>Byte 14</code></td>
<td valign="middle" align="center"><code>Byte 15</code></td>
</tr>

```

```

<tr>
  <td valign="middle" align="center"><code>cccccccc</code></td>
  <td valign="middle" align="center"><code>cccccccc</code></td>
  <td valign="middle" align="center"><code>cccccccc</code></td>
  <td valign="middle" align="center"><code>cccccccc</code></td>
</tr>
</table>

```

```

</td></tr>
<tr>
  <td valign="middle" align="left">&nbsp;<br />b) Compound type 's2_t', size 8 bytes.</td>
</tr>

```

```

<tr><td>

<table border="1" align="center" width="100%">
  <tr>
    <td valign="middle" align="center" width="25%"><code>Byte 0</code></td>
    <td valign="middle" align="center" width="25%"><code>Byte 1</code></td>
    <td valign="middle" align="center" width="25%"><code>Byte 2</code></td>
    <td valign="middle" align="center" width="25%"><code>Byte 3</code></td>
  </tr>
<tr>
  <td valign="middle" align="center"><code>ffffff</code></td>
  <td valign="middle" align="center"><code>ffffff</code></td>
  <td valign="middle" align="center"><code>ffffff</code></td>
  <td valign="middle" align="center"><code>ffffff</code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code>Byte 4</code></td>

```

```

<td valign="middle" align="center"><code>Byte 5</code></td>
<td valign="middle" align="center"><code>Byte 6</code></td>
<td valign="middle" align="center"><code>Byte 7</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code>gggggggg</code></td>
<td valign="middle" align="center"><code>gggggggg</code></td>
<td valign="middle" align="center"><code>gggggggg</code></td>
<td valign="middle" align="center"><code>gggggggg</code></td>
</tr>
</table>

</td></tr>
<tr>
<td valign="middle" align="left">&nbsp;<br />c) Array type 's3_tid', 40 integers, total size
40 bytes.</td>
</tr>

<tr><td>

<table border="1" align="center" width="100%">
<tr>
<td valign="middle" align="center" width="25%"><code>Byte 0</code></td>
<td valign="middle" align="center" width="25%"><code>Byte 1</code></td>
<td valign="middle" align="center" width="25%"><code>Byte 2</code></td>
<td valign="middle" align="center" width="25%"><code>Byte 3</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code>00000000</code></td>
<td valign="middle" align="center"><code>00000000</code></td>
<td valign="middle" align="center"><code>00000000</code></td>

```



```

<td valign="middle" align="center"><code>00000000</code></td>
<td valign="middle" align="center"><code>00000000</code></td>
<td valign="middle" align="center"><code>00001010</code></td>
</tr>
</table>

```

```

</td></tr>
<tr>
<td valign="middle" align="left">&nbsp;<br />d) String type &lsquo;s4_tid&rsquo;; size 25 bytes.</td>
</tr>
<tr><td>

```

```

<table border="1" align="center" width="100%">
<tr>
<td valign="middle" align="center" width="25%"><code>Byte 0</code></td>
<td valign="middle" align="center" width="25%"><code>Byte 1</code></td>
<td valign="middle" align="center" width="25%"><code>Byte 2</code></td>
<td valign="middle" align="center" width="25%"><code>Byte 3</code></td>
</tr>
<tr>
<td valign="middle" align="center"><code>&lsquo;a&rsquo;</code></td>
<td valign="middle" align="center"><code>&lsquo;b&rsquo;</code></td>
<td valign="middle" align="center"><code>&lsquo;c&rsquo;</code></td>
<td valign="middle" align="center"><code>&lsquo;d&rsquo;</code></td>
</tr>
</table>

```

```

<table align="center" width="100%">
<tr>
<td align="center" colspan="4">&nbsp;&nbsp;&nbsp;&nbsp;<br />&nbsp;&nbsp;&nbsp;</td>
</tr>

```

```

</table>

<table border="1" align="center" width="100%">
<tr>
  <td valign="middle" align="center" width="25%"><code>Byte 24</code></td>
  <td valign="middle" align="center" width="25%"><code>Byte 25</code></td>
  <td valign="middle" align="center" width="25%"><code>Byte 26</code></td>
  <td valign="middle" align="center" width="25%"><code>Byte 27</code></td>
</tr>
<tr>
  <td valign="middle" align="center"><code>00000000</code></td>
  <td valign="middle" align="center"><code>&nbsp;</code></td>
  <td valign="middle" align="center"><code>&nbsp;</code></td>
  <td valign="middle" align="center"><code>&nbsp;</code></td>
</tr>
</table>

</td></tr>
</table>

</td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 22. The storage layout for the
    four member datatypes<!-- formerly Figure 49 --></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
<br />

```



```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 23. The storage layout of the combined four members
      <!-- formerly Figure 50 --></b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />
<br />

```

```

<!-- NEW PAGE -->
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/></td>
    </tr>
  <tr align="center">
    <td align="center"></td>
  </tr>

```

```

    </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left" >
        <b>Figure 24. The layout of the dataset
        <!-- formerly Figure 51 --></b>
        <hr color="green" size="3"/></td>
    </tr>
<!-- 9.1.10, the JPG above, Dtypes_fig51.jpg, spells Element incorrectly -->
<!-- 9.1.10, the section above has text and many examples and figures.
Should the text be interspersed with the examples and figures at some
point? -->
</table>
<br />

```

```

<!-- NEW PAGE -->

```

```

<h4>6.7.2. Analyzing and Navigating a Compound Datatype</h4>

```

A complicated compound datatype can be analyzed piece by piece to discover the exact storage layout. In the example above, the outer datatype is analyzed to discover that it is a compound datatype with four members. Each member is analyzed in turn to construct a complete map of the storage layout.

The example below <!-- formerly Figure 52 --> shows an example of code that partially analyzes a nested compound datatype. The name and overall offset and size of the component datatype is discovered, and then its type is analyzed depending on the datatype class. Through this method, the complete storage layout can be

discovered.</p>

```
<table width="600" cellpadding="0" align="center">
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<hr color="green" size="3"/>
```

```
<pre>
```

```
s1_tid = H5Dget_type(dataset);
```

```
if (H5Tget_class(s1_tid) == H5T_COMPOUND) {
```

```
    printf(&ldquo;COMPOUND DATATYPE {\n&rdquo;);
```

```
    sz = H5Tget_size(s1_tid);
```

```
    nmemb = H5Tget_nmembers(s1_tid);
```

```
    printf(&ldquo; %d bytes\n&rdquo;,sz);
```

```
    printf(&ldquo; %d members\n&rdquo;,nmemb);
```

```
    for (i =0; i &lt; nmemb; i++) {
```

```
        s2_tid = H5Tget_member_type(s1_tid, i);
```

```
        if (H5Tget_class(s2_tid) == H5T_COMPOUND) {
```

```
            /* recursively analyze the nested type. */
```

```
        } else if (H5Tget_class(s2_tid) == H5T_ARRAY) {
```

```
            sz2 = H5Tget_size(s2_tid);
```

```
            printf(&ldquo; %s: NESTED ARRAY DATATYPE offset %d size %d {\n&rdquo;;
```

```
                H5Tget_member_name(s1_tid, i),
```

```
                H5Tget_member_offset(s1_tid, i),
```

```
                sz2);
```

```
            H5Tget_array_dims(s2_tid, dim);
```

```
            s3_tid = H5Tget_super(s2_tid);
```

```
            /* Etc., analyze the base type of the array */
```

```
        } else {
```

```

/* analyze a simple type */
printf("&ldquo;  %s: type code %d offset %d size %d\\n&rdquo;;",
       H5Tget_member_name(s1_tid, i),
       H5Tget_class(s2_tid),
       H5Tget_member_offset(s1_tid, i),
       H5Tget_size(s2_tid));
}
/* and so on.... */</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 41. Analyzing a compound datatype and its members
    <!-- formerly Figure 52--></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

```

<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="LCDtypeObj">
<div align=right>
<a href="#TOP"><font size="-1">(Top)</font></a>
</div>
</a>
");
-->

```

</SCRIPT>

<!-- NEW PAGE -->

<h3 class=pagebefore>6.8. Life Cycle of the Datatype Object</h3>

<p>Application programs access HDF5 datatypes through identifiers. Identifiers are obtained by creating a new datatype or by copying or opening an existing datatype. The identifier can be used until it is closed or until the library shuts down. See items a and b in the figure below. <!-- formerly Figure 53a,b --> By default, a datatype is transient, and it disappears when it is closed. </p>

<p>When a dataset or attribute is created (<code>H5Dcreate</code> or <code>H5Acreate</code>), its datatype is stored in the HDF5 file as part of the dataset or attribute object. See item c in the figure below. Once an object created, its datatype cannot be changed or deleted. The datatype can be accessed by calling <code>H5Dget_type</code>, <code>H5Aget_type</code>, <code>H5Tget_super</code>, or <code>H5Tget_member_type</code>. See item d in the figure below. These calls return an identifier to a transient copy of the datatype of the dataset or attribute unless the datatype is a committed datatype. </p>

<p>Note that when an object is created, the stored datatype is a copy of the transient datatype. If two objects are created with the same datatype, the information is stored in each object with the same effect as if two different datatypes were created and used. </p>

A transient datatype can be stored using `H5Tcommit` in the HDF5 file as an independent, named object, called a committed datatype.

Committed datatypes were formerly known as named datatypes.

See item e in the figure below. Subsequently, when a committed datatype is opened with `H5Topen` (item f), or is obtained with `H5Tget_type` or similar call (item k), the return is an identifier to a transient copy of the stored datatype. The identifier can be used in the same way as other datatype identifiers except that the committed datatype cannot be modified. When a committed datatype is copied with `H5Tcopy`, the return is a new, modifiable, transient datatype object (item f).

When an object is created using a committed datatype (`H5Dcreate`, `H5Acreate`), the stored datatype is used without copying it to the object. See item j in the figure below. In this case, if multiple objects are created using the same committed datatype, they all share the exact same datatype object. This saves space and makes clear that the datatype is shared. Note that a committed datatype can be shared by objects within the same HDF5 file, but not by objects in other files. For more information on copying committed datatypes to other HDF5 files, see the “Copying Committed Datatypes with `H5Ocopy`” topic in the “Additional Resources” chapter.

A committed datatype can be deleted from the file by calling `H5Ldelete` which replaces `H5Gunlink`. See item i in the figure below. If one or more objects are still using the datatype, the committed datatype cannot be accessed with `H5Topen`, but will not be removed from the file until it is no longer used. `H5Tget_type` and similar calls will return a transient

copy of the datatype.</p>

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center">

<tr valign="top">

<td align="center">

<hr color="green" size="3"/>

</td>

</tr>

<tr><td><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td align="left" >

Figure 25. Life cycle of a datatype

<!-- formerly Figure 53 -->

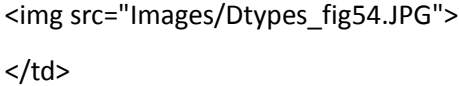
<hr color="green" size="3"/></td>

</tr>

</table>

<p>Transient datatypes are initially modifiable. Note that when a datatype is copied or when it is written to the file (when an object is created) or the datatype is used to create a composite datatype, a copy of the current state of the datatype is used. If the datatype is then modified, the changes have no effect on datasets, attributes, or datatypes that have already been created. See the figure below.</p>

A transient datatype can be made `read-only` (`H5Tlock`). Note that the datatype is still transient, and otherwise does not change. A datatype that is `immutable` is `read-only` but cannot be closed except when the entire library is closed. The predefined types such as `H5T_NATIVE_INT` are `immutable transient` types.


<p>Figure 26. Transient datatype states: modifiable, read-only, and immutable <small>formerly Figure 54</small></p>

To create two or more datasets that share a common datatype, first commit the datatype, and then use that datatype to create the datasets. See the example below.


```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
hid_t t1 = ...some transient type...;
H5Tcommit (file, &ldquo;shared_type&rdquo;;, t1, H5P_DEFAULT, H5P_DEFAULT,
  H5P_DEFAULT);
hid_t dset1 = H5Dcreate (file, &ldquo;dset1&rdquo;;, t1, space, H5P_DEFAULT,
  H5P_DEFAULT, H5P_DEFAULT);
hid_t dset2 = H5Dcreate (file, &ldquo;dset2&rdquo;;, t1, space, H5P_DEFAULT,
  H5P_DEFAULT, H5P_DEFAULT);

hid_t dset1 = H5Dopen (file, &ldquo;dset1&rdquo;;, H5P_DEFAULT);
hid_t t2 = H5Dget_type (dset1);
hid_t dset3 = H5Dcreate (file, &ldquo;dset3&rdquo;;, t2, space, H5P_DEFAULT,
  H5P_DEFAULT, H5P_DEFAULT);
hid_t dset4 = H5Dcreate (file, &ldquo;dset4&rdquo;;, t2, space, H5P_DEFAULT,
  H5P_DEFAULT, H5P_DEFAULT);</pre> </td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      <b>Example 42. Create a shareable datatype
      <!-- formerly Figure 55 --></b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

<!-- NEW PAGE -->

<table width="600" cellspacing="0" align="center" cellpadding="0">

 <tr valign="bottom">

 <td colspan="2" align="left" valign="bottom">

 Table 23. Datatype APIs

 <!-- formerly Table 24 --></td>

 </tr>

<tr><td colspan="2"><hr color="green" size="3" /></td></tr>

<tr valign="top">

 <td width="50%">Function</td>

 <td width="50%">Description</td>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

 <td><code>hid_t H5Topen (hid_t location,
const
 char *name)</code></td>

 <td>A committed datatype can be opened by

calling this function, which returns a datatype identifier. The
identifier should eventually be released by calling

<code>H5Tclose()</code> to release resources. The committed
datatype returned by this function is read-only or a negative
value is returned for failure. The location is either a file or
group identifier.</td>

</tr>

<tr><td colspan="2"><hr color="green" size="1" /></td></tr>

<tr valign="top">

```
<td><code>herr_t H5Tcommit (hid_t location,
const char *name, hid_t type, H5P_DEFAULT, H5P_DEFAULT,
<br />H5P_DEFAULT)</code></td>
```

A transient datatype (not immutable) can be written to a file and turned into a committed datatype by calling this function. The location is either a file or group identifier and when combined with name refers to a new committed datatype.</td>

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td><code>htri_t H5Tcommitted
(hid_t type)</code></td>
```

A type can be queried to determine if it is a committed type or a transient type. If this function returns a positive value then the type is committed. Datasets which return committed datatypes with `H5Dget_type()` are able to share the datatype with other datasets in the same file.</td>

```
</tr>
```

```
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
```

```
</table>
```

```
<br />
```

```
<SCRIPT language="JavaScript">
```

```
<!--
```

```
document.writeln ("
```

```
<a name="Dtransfer">
```

```
<div align=right>
```

```
<a href="#TOP"><font size="-1">(Top)</font></a>
```

```
</div>
```

```
</a>
");
-->
</SCRIPT>
<br />
<!-- NEW PAGE -->
<a name="Dtransfer">
<h3 class=pagebefore>6.9. Data Transfer: Datatype Conversion and Selection</h3>
</a>
```

When data is transferred (write or read), the storage layout of the data elements may be different. For example, an integer might be stored on disk in big-endian byte order and read into memory with little-endian byte order. In this case, each data element will be transformed by the HDF5 Library during the data transfer.

The conversion of data elements is controlled by specifying the datatype of the source and specifying the intended datatype of the destination. The storage format on disk is the datatype specified when the dataset is created. The datatype of memory must be specified in the library call.

In order to be convertible, the datatype of the source and destination must have the same datatype class (with the exception of enumeration type). Thus, integers can be converted to other integers, and floats to other floats, but integers cannot (yet) be converted to floats. For each atomic datatype class, the possible conversions are defined. An enumeration datatype can be converted to an integer or a floating-point number datatype.

Basically, any datatype can be converted to another datatype of the same

datatype class. The HDF5 Library automatically converts all properties.

If the destination is too small to hold the source value then an overflow or underflow exception occurs. If a handler is defined with the

`H5Pset_type_conv_cb` function,

<!-- editingComment

[[[

(see Chapter??)

]]

-->

it will be called. Otherwise,

a default action will be performed. The table below <!-- formerly Table 25-->

summarizes the default actions.</p>

<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="3" align="left" valign="bottom">

Table 24. Default actions for datatype conversion exceptions

<!-- formerly Table 25 --></td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td>Datatype Class</td>

<td>Possible Exceptions</td>

<td>Default Action</td>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>Integer</td>

<td>Size, offset, pad</td>

<td> </td>

</tr>

```

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Float</td>
  <td>Size, offset, pad, ebits</td>
  <td>&nbsp;</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>String</td>
  <td>Size</td>
  <td>Truncates, zero terminate if required.</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Enumeration</td>
  <td>No field</td>
  <td>All bits set</td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

For example, when reading data from a dataset, the source datatype is the datatype set when the dataset was created, and the destination datatype is the description of the storage layout in memory. The destination datatype must be specified in the `H5Dread` call. The example below formerly Figure 56 shows an example of reading a dataset

of 32-bit integers. The figure <!-- formerly Figure 57 -->below the example shows the data transformation that is performed.</p>

<pre> /* Stored as H5T_STD_BE32 */ /* Use the native memory order in the destination */ mem_type_id = H5Tcopy(H5T_NATIVE_INT); status = H5Dread(dataset_id, mem_type_id, mem_space_id, file_space_id, xfer_plist_id, buf); </pre>
<hr/>
<p>Example 43. Specify the destination datatype with <code>H5Dread</code><!-- formerly Figure 56 --></p> <hr/>

<!-- NEW PAGE -->

```
<table width="600" cellspacing="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="center">
```

```
    <hr color="green" size="3"/>
```

```
<table align="center" width="100%">
```

```
<tr><td>
```

```
<table align="left">
```

```
  <tr>
```

```
  <td align="left">Source Datatype: <code>H5T_STD_BE32</code></td>
```

```
  </tr>
```

```
</table>
```

```
</td></tr>
```

```
<tr><td>
```

```
<table align="left" border="1" width="100%">
```

```
  <tr>
```

```
  <td valign="middle" align="center" width="25%"><code>Byte 0</code></td>
```

```
  <td valign="middle" align="center" width="25%"><code>Byte 1</code></td>
```

```
  <td valign="middle" align="center" width="25%"><code>Byte 2</code></td>
```

```
  <td valign="middle" align="center" width="25%"><code>Byte 3</code></td>
```

```
  </tr>
```

```
<tr>
```

```
  <td valign="middle" align="center"><code>aaaaaaaa</code></td>
```

```
  <td valign="middle" align="center"><code>bbbbbbbb</code></td>
```

```
  <td valign="middle" align="center"><code>cccccccc</code></td>
```

```
  <td valign="middle" align="center"><code>dddddddd</code></td>
```

```
</tr>
```



```
<tr>

<td valign="middle" align="center"><code>Byte 4</code></td>

<td valign="middle" align="center"><code>Byte 5</code></td>

<td valign="middle" align="center"><code>Byte 6</code></td>

<td valign="middle" align="center"><code>Byte 7</code></td>

</tr>

<tr>

<td valign="middle" align="center"><code>wwwwwwwww</code></td>

<td valign="middle" align="center"><code>xxxxxxx</code></td>

<td valign="middle" align="center"><code>yyyyyyy</code></td>

<td valign="middle" align="center"><code>zxxxxxx</code></td>

</tr>

</table>

</td></tr>

<tr>

<td>. &nbsp; .&nbsp; .&nbsp; .</td>

</tr>

<tr><td>

<table align="center" width="100%">

<tr>

<td width="45%">&nbsp;</td>

<td width="10%" align="center"></td>

<td width="45%" align="left">Automatically byte swapped<br /> during the <code>H5Dread</code></td>

</tr>

</table>

</td></tr>

<tr><td>
```

```

    <table align="left">
    <tr>
    <td align="left">Destination Datatype: <code>H5T_STD_LE32</code></td>
    </tr>
    </table>

    </td></tr>
    <tr><td>

    <table align="left" border="1" width="100%">
    <tr>
    <td valign="middle" align="center" width="25%"><code>Byte 0</code></td>
    <td valign="middle" align="center" width="25%"><code>Byte 1</code></td>
    <td valign="middle" align="center" width="25%"><code>Byte 2</code></td>
    <td valign="middle" align="center" width="25%"><code>Byte 3</code></td>
    </tr>
    <tr>
    <td valign="middle" align="center"><code>bbbbbbbb</code></td>
    <td valign="middle" align="center"><code>aaaaaaaa</code></td>
    <td valign="middle" align="center"><code>dddddddd</code></td>
    <td valign="middle" align="center"><code>cccccccc</code></td>
    </tr>
    <tr>
    <td valign="middle" align="center"><code>Byte 4</code></td>
    <td valign="middle" align="center"><code>Byte 5</code></td>
    <td valign="middle" align="center"><code>Byte 6</code></td>
    <td valign="middle" align="center"><code>Byte 7</code></td>
    </tr>
    <tr>
    <td valign="middle" align="center"><code>xxxxxxx</code></td>

```

```
<td valign="middle" align="center"><code>wwwwwwww</code></td>
<td valign="middle" align="center"><code>zzzzzzzz</code></td>
<td valign="middle" align="center"><code>yyyyyyyy</code></td>
</tr>
</table>
```

```
</td></tr>

<tr>

    <td>.&nbsp;&nbsp;&nbsp;&nbsp;&.&nbsp;&nbsp;&nbsp;&.&nbsp;&nbsp;&.</td>

</tr>

<tr><td>


</table>



</td></tr>

<tr><td><hr color="green" size="1" /></td></tr>

<tr valign="top">

    <td align="left" >

        <b>Figure 27. Layout of a datatype conversion

        <!-- formerly Figure 57 --></b>


        <hr color="green" size="3"/></td>

    </tr>

</table>

<br />
```

<p>One thing to note in the example above <!-- formerly Figure 56 -->is the use of the predefined native datatype <code>H5T_NATIVE_INT</code>. Recall that in this example, the data was stored as a 4-bytes in big-endian order. The application wants to read this data into an array

of integers in memory. Depending on the system, the storage layout of memory might be either big or little-endian, so the data may need to be transformed on some platforms and not on others. The `H5T_NATIVE_INT` type is set by the HDF5 Library to be the correct type to describe the storage layout of the memory on the system. Thus, the code in the example above  will work correctly on any platform, performing a transformation when needed.

There are predefined native types for most atomic datatypes, and these can be combined in composite datatypes. In general, the predefined native datatypes should always be used for data stored in memory.

<div data-bbox="207 961 276 993" data-label="Text"><p><tr ></p></div> <div data-bbox="233 1016 714 1050" data-label="Text"><p><td style="background-color:#E6F2E6"></p></div> <div data-bbox="233 1071 584 1104" data-label="Text"><p><hr color="green" size="3"/></p></div> <div data-bbox="233 1125 643 1161" data-label="Text"><p>Storage Properties
</p></div> <div data-bbox="233 1180 995 1272" data-label="Text"><p><p>Predefined native datatypes describe the storage properties of memory.</p></p></div> <div data-bbox="233 1291 649 1327" data-label="Text"><p><hr color="green" size="3"/></td></p></div> <div data-bbox="233 1348 306 1381" data-label="Text"><p></tr></p></div> <div data-bbox="194 1402 306 1438" data-label="Text"><p></table></p></div> <div data-bbox="180 1457 266 1495" data-label="Text"><p>
</p></div>

<tr >

<td style="background-color:#E6F2E6">

<hr color="green" size="3"/>

Storage Properties

<p>Predefined native datatypes describe the storage properties of memory.</p>

<hr color="green" size="3"/></td>

</tr>

</table>

For composite datatypes, the component atomic datatypes will be converted. For a variable-length datatype, the source and destination must have compatible base datatypes. For a fixed-size string datatype, the length and padding of the strings will be converted. Variable-length strings

are converted as variable-length datatypes.

For an array datatype, the source and destination must have the same rank and dimensions, and the base datatype must be compatible. For example an array datatype of 4 x 3 32-bit big-endian integers can be transferred to an array datatype of 4 x 3 little-endian integers, but not to a 3 x 4 array.

For an enumeration datatype, data elements are converted by matching the symbol names of the source and destination datatype. The figure below
 (formerly Figure 58) shows an example of how two enumerations with the same names and different values would be converted. The value '2' in the source dataset would be converted to '0x0004' in the destination.

If the source data stream contains values which are not in the domain of the conversion map then an overflow exception is raised within the library.

--

--

--

--

--

--	--

--	--

--	--

--	--

--	--

--	--


```
<td width="%">&nbsp;  &nbsp; </td>
<td width="%">BLACK&nbsp;  &nbsp; </td>
<td width="%"></td>
<td align="right" width="%">&nbsp;  &nbsp; BLACK</td>
<td width="%">&nbsp;  &nbsp; </td>
<td width="%">&nbsp;  0x0010</td>
</tr>
</table>
</td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td align="left" >
<b>Figure 28. An enum datatype conversion
<!-- formerly Figure 58 --></b>
<hr color="green" size="3"/></td>
</tr>
</table>
<br />
```

The library also allows conversion from enumeration to a numeric datatype. A numeric datatype is either an integer or a floating-point number. This conversion can simplify the application program because the base type for an enumeration datatype is an integer datatype. The application program can read the data from a dataset of enumeration datatype in file into a memory buffer of numeric datatype. And it can write enumeration data from memory into a dataset of numeric datatype in file, too.

For compound datatypes, each field of the source and destination

datatype is converted according to its type. The name of the fields must be the same in the source and the destination in order for the data to be converted.

The example below formerly Figure 59 shows the compound datatypes shows sample code to create a compound datatype with the fields aligned on word boundaries (`s1_tid`) and with the fields packed (`s2_tid`). The former is suitable as a description of the storage layout in memory, the latter would give a more compact store on disk. These types can be used for transferring data, with `s2_tid` used to create the dataset, and `s1_tid` used as the memory datatype.

<pre> typedef struct s1_t { int a; char b; double c; } s1_t; s1_tid = H5Tcreate (H5T_COMPOUND, sizeof(s1_t)); H5Tinsert(s1_tid, "a_name", HOFFSET(s1_t, a), H5T_NATIVE_INT); H5Tinsert(s1_tid, "b_name", HOFFSET(s1_t, b), H5T_NATIVE_CHAR); H5Tinsert(s1_tid, "c_name", HOFFSET(s1_t, c), H5T_NATIVE_DOUBLE); s2_tid = H5Tcopy(s1_tid); H5Tpack(s2_tid); </pre>

```

    </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 44. Create an aligned and packed compound datatype
        <!-- formerly Figure 59 --></b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

When the data is transferred, the fields within each data element will be aligned according to the datatype specification. The figure below <!-- formerly Figure 60 --> shows how one data element would be aligned in memory and on disk. Note that the size and byte order of the elements might also be converted during the transfer.</p>

It is also possible to transfer some of the fields of compound datatypes. Based on the example above, <!-- formerly Figure 59 --> the example below <!-- formerly Figure 61 --> shows a compound datatype that selects the first and third fields of the `s1_tid`. The second datatype can be used as the memory datatype, in which case data is read from or written to these two fields, while skipping the middle field. The second figure below <!-- formerly Figure 62 --> shows the layout for two data elements.</p>

```

<table width="600" cellpadding="0" align="center">
    <tr valign="top">

```

```

<td align="center">
<hr color="green" size="3"/>

</td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 29. Alignment of a compound datatype
    <!-- formerly Figure 60 --></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
<br />

```

```

<!-- NEW PAGE -->

```

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
typedef struct s1_t {
  int  a;
  char b;
  double c;
} s1_t;

```

```
typedef struct s2_t { /* two fields from s1_t */
    int  a;
    double c;
} s2_t;
```

```
    s1_tid = H5Tcreate (H5T_COMPOUND, sizeof(s1_t));
    H5Tinsert(s1_tid, &ldquo;a_name&rdquo;, HOFFSET(s1_t, a), H5T_NATIVE_INT);
    H5Tinsert(s1_tid, &ldquo;b_name&rdquo;, HOFFSET(s1_t, b), H5T_NATIVE_CHAR);
    H5Tinsert(s1_tid, &ldquo;c_name&rdquo;, HOFFSET(s1_t, c), H5T_NATIVE_DOUBLE);
```

```
s2_tid = H5Tcreate (H5T_COMPOUND, sizeof(s2_t));
H5Tinsert(s1_tid, &ldquo;a_name&rdquo;, HOFFSET(s2_t, a), H5T_NATIVE_INT);
H5Tinsert(s1_tid, &ldquo;c_name&rdquo;, HOFFSET(s2_t, c), H5T_NATIVE_DOUBLE);
```

```
</pre> </td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 45. Transfer some fields of a compound datatype
```

```
<!-- formerly Figure 61 --></b>
```

```
<hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

```
<br />
```

```
<!-- NEW PAGE -->
```

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 30. Layout when an element is skipped
      <!-- formerly Figure 62 --></b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

```

<!-- NEW PAGE -->

```

```

<a name="TextDescriptions">

```

```

<h3 class=pagebefore>6.10. Text Descriptions of Datatypes: Conversion to
and from</h3></a>

```

HDF5 provides a means for generating a portable and human-readable text description of a datatype and for generating a datatype from such a text description. This capability is particularly useful for creating complex datatypes in a single step, for creating a text description of a datatype for debugging purposes,

and for creating a portable datatype definition that can then be used to recreate the datatype on many platforms or in other applications.

These tasks are handled by two functions provided in the HDF5 high-level library ([H5HL](../../HL/RM_H5LT.html)):

| | |
 H5LTtext_to_dtype | | Creates an HDF5 datatype in a single step. | | H5LTdtype_to_text | | Translates an HDF5 datatype into a text description. |

Note that this functionality requires that the HDF5 High-Level Library (H5LT) be installed.

```


```

or in other applications.

These two functions work for all valid HDF5 datatypes except time, bitfield, and reference datatypes.

The currently supported text format used by `H5Ltext_to_dtype` and `H5Ldtype_to_text` is the data description language (DDL) and conforms to the [HDF5 DDL](http://hdfgroup.org/hdf/5.0/ddl.html). The portion of the `HDF5 DDL` that defines HDF5 datatypes appears below.

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
&lt;datatype&gt; ::= &lt;atomic_type&gt; | &lt;compound_type&gt; | &lt;array_type&gt; |
    &lt;variable_length_type&gt;

&lt;atomic_type&gt; ::= &lt;integer&gt; | &lt;float&gt; | &lt;time&gt; | &lt;string&gt; |
    &lt;bitfield&gt; | &lt;opaque&gt; | &lt;reference&gt; | &lt;enum&gt;

&lt;integer&gt; ::= H5T_STD_I8BE | H5T_STD_I8LE |
    H5T_STD_I16BE | H5T_STD_I16LE |
    H5T_STD_I32BE | H5T_STD_I32LE |
    H5T_STD_I64BE | H5T_STD_I64LE |
    H5T_STD_U8BE | H5T_STD_U8LE |

```

```

H5T_STD_U16BE | H5T_STD_U16LE |
H5T_STD_U32BE | H5T_STD_U32LE |
H5T_STD_U64BE | H5T_STD_U64LE |
H5T_NATIVE_CHAR | H5T_NATIVE_UCHAR |
H5T_NATIVE_SHORT | H5T_NATIVE_USHORT |
H5T_NATIVE_INT | H5T_NATIVE_UINT |
H5T_NATIVE_LONG | H5T_NATIVE_ULONG |
H5T_NATIVE_LLONG | H5T_NATIVE_ULLONG

```

```

<float> ::= H5T_IEEE_F32BE | H5T_IEEE_F32LE |
H5T_IEEE_F64BE | H5T_IEEE_F64LE |
H5T_NATIVE_FLOAT | H5T_NATIVE_DOUBLE |
H5T_NATIVE_LDOUBLE

```

```

<time> ::= TBD

```

```

<string> ::= H5T_STRING { STRSIZE <strsize>;
STRPAD <strpad>;
CSET <cset>;
CTYPE <ctype>; }

```

```

<strsize> ::= <int_value> | H5T_VARIABLE
<strpad> ::= H5T_STR_NULLTERM | H5T_STR_NULLPAD | H5T_STR_SPACEPAD
<cset> ::= H5T_CSET_ASCII | H5T_CSET_UTF8
<ctype> ::= H5T_C_S1 | H5T_FORTRAN_S1

```

```

<bitfield> ::= TBD

```

```

<opaque> ::= H5T_OPAQUE { OPQ_SIZE <opq_size>;
OPQ_TAG <opq_tag>; }
opq_size ::= <int_value>

```

```
opq_tag ::= "<string>,"
```

```
<reference> ::= Not supported
```

```
<compound_type> ::= H5T_COMPOUND { <member_type_def>+ }
```

```
<member_type_def> ::= <datatype> <field_name> <offset><font size=1.7>opt</font> ;
```

```
<field_name> ::= "<identifier>,"
```

```
<offset> ::= : <int_value>;
```

```
<variable_length_type> ::= H5T_VLEN { <datatype> }
```

```
<array_type> ::= H5T_ARRAY { <dim_sizes> <datatype> }
```

```
<dim_sizes> ::= [<dim_size>] | [<dim_size>] <dim_sizes>;
```

```
<dim_size> ::= <int_value>;
```

```
<enum> ::= H5T_ENUM { <enum_base_type>; <enum_def>+ }
```

```
<enum_base_type> ::= <integer>;
```

```
// Currently enums can only hold integer type data, but they may be
```

```
//expanded in the future to hold any datatype
```

```
<enum_def> ::= <enum_symbol> <enum_val>;
```

```
<enum_symbol> ::= "<identifier>,"
```

```
<enum_val> ::= <int_value>;
```

```
</pre>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 46. The definition of HDF5 datatypes from the
```

```
<!-- formerly Figure 63: -->
```

```
<a href=" ../ddl.html" target="ExtWin"><cite>HDF5 DDL</cite></a></b>
```

```
<hr color="green" size="3" /></td>
```



```

    </tr>
</table>
<br />

```

The definitions of opaque and compound datatype above are revised for HDF5 Release 1.8. In Release 1.6.5. and earlier, they were defined as follows:

```

</p>

```

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
&lt;opaque&gt; ::= H5T_OPAQUE { &lt;identifier&gt; }

```

```

&lt;compound_type&gt; ::= H5T_COMPOUND { &lt;member_type_def&gt;+ }
&lt;member_type_def&gt; ::= &lt;datatype&gt; &lt;field_name&gt;;
&lt;field_name&gt; ::= &lt;identifier&gt;;</pre>

```

```

<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 47.
    <!-- formerly Figure 64: -->
    Old definitions of the opaque and compound datatypes</b>
    <hr color="green" size="3"/></td>
  </tr>

```

</table>

<h4>Examples</h4>

<p>The code sample below illustrates the use of

H5LTtext_to_dtype to generate a variable-length string datatype.

</p>

<table width="600" cellspacing="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

hid_t dtype;

if((dtype = H5LTtext_to_dtype(&ldquo;H5T_STRING {

STRSIZE H5T_VARIABLE;

STRPAD H5T_STR_NULLPAD;

CSET H5T_CSET_ASCII;

CTYPE H5T_C_S1;

&rdquo;, H5LT_DDL))&lt;0)

goto out;</pre>

<tr><td><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td align="left">

Example 48. Creating a variable-length string datatype from

```

        a text description<!-- formerly Figure 65: --></b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

The code sample below illustrates the use of

`H5LTtext_to_dtype` to generate a complex array datatype.

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
hid_t dtype;
if((dtype = H5LTtext_to_dtype("H5T_ARRAY { [5][7][13] H5T_ARRAY
                                { [17][19] H5T_COMPOUND
                                {
                                    H5T_STD_I8BE
                                    \&ldquo;arr_compound_1\&rdquo;;
                                    H5T_STD_I32BE
                                    \&ldquo;arr_compound_2\&rdquo;;
                                }
                                }
                                }&rdquo;, H5LT_DDL))<0)

```

```
goto out;</pre>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 49. <!-- formerly Figure 66: -->
    Creating a complex array datatype from a text description</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
<br />
<br />

<!-- NEW PAGE -->
</body>
</html>
```

```

<!doctype HTML public "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
<head>
<title>Chapter 7: HDF5 Dataspaces and Partial I/O</title>

<!--(Meta)=====-->

<!--(Links)=====-->

<!--( Begin styles definition )=====-->
<link href="ed_styles/NewUglect.css" rel="stylesheet" type="text/css">
<!--( End styles definition )=====-->

</head>

<body>

<!-- #BeginLibraryItem "/ed_libs/Copyright.lbi" -->
<!--
* * * * *
* Copyright by The HDF Group. *
* Copyright by the Board of Trustees of the University of Illinois. *
* All rights reserved. *
* *
* This file is part of HDF5. The full HDF5 copyright notice, including *
* terms governing use, modification, and redistribution, is contained in *
* the files COPYING and Copyright.html. COPYING can be found at the root *
* of the source code distribution tree; Copyright.html can be found at the *
* root level of an installed copy of the electronic HDF5 document set and *
* is linked from the top-level documents page. It can also be found at *
* http://www.hdfgroup.org/HDF5/doc/Copyright.html. If you do not have *

```

```

* access to either file, you may request a copy from help@hdfgroup.org.  *
* * * * *
-->
<!-- #EndLibraryItem --><!-- HEADER LEFT "HDF5 User's Guide" -->
<!-- HEADER RIGHT "HDF5 Dataspaces and Partial I/O" -->

<!-- ( TOC )=====-->
<SCRIPT language="JavaScript">
<!--
document.writeln ('\
<table x-use-null-cells\
        align="right"\
width="240"\
cellspacing="0"\
class="tocTable">\
  <tr valign="top"> \
    <td class="tocTableHeaderCell" colspan="2"> \
      <span class="TableHead">Chapter Contents</span></td>\
    </tr>\
-->
<!-- Table Version 3 -->\
<!--
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#Intro">1.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#Intro">Introduction</a></td> \
  </tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#DSpaceFunctSums">2.</a></td>\

```

```

    <td class="tocTableContentCell3">\
<a href="#DSpaceFunctSums">Dataspace (H5S) Function Summaries</a></td>\
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#DefDataObjs">3.1</a></td>\
    <td class="tocTableContentCell3">\
<a href="#DefDataObjs">Dataspace Objects</a></td> \
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#ProgModel">3.2</a></td>\
    <td class="tocTableContentCell3">\
<a href="#ProgModel">Programming Model</a></td>\
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#DTransfer">4.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#DTransfer">Dataspaces and Data Transfer</a></td> \
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#DSelectTransfer">5.</a></td>\
    <td class="tocTableContentCell3">\
<a href="#DSelectTransfer">Selection Operations and Data Transfer</a></td> \
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#DRegions">6.</a></td>\
    <td class="tocTableContentCell3">\

```

```

<a href="#DRegions">References to Dataset Regions</a>\
</td>\
</tr>\
\
<tr valign="top"> \
  <td class="tocTableContentCell"> \
-->
<!-- editingComment -- "tocTableContentCell" and "tocTableContentCell4" \
-->\
<!-- are the table-closing cell class.\
  <td class="tocTableContentCell2"> \
-->\
<!--
  <a href="#Programs">7.</a></td>\
  <td class="tocTableContentCell4">\
  <a href="#Programs">Sample Programs</a>\
  </td></tr>\
</table>\
')
-->
</SCRIPT>
<!--(End TOC)=====-->

<div align="center">
<a name="TOP">

```


8. HDF5 Dataspaces and Partial I/O

</div>

<!--

<dir>

<h1 class="editingcomment" align="center">- - - DRAFT - - -</h1>

</dir>

-->

<h3>7.1. Introduction</h3>

<p>The HDF5 *dataspace* is a required component of an HDF5 dataset or attribute definition. The dataspace defines the size and shape of the dataset or attribute raw data. In other words, a dataspace defines the number of dimensions and the size of each dimension of the multidimensional array in which the raw data is represented. The dataspace must be defined when the dataset or attribute is created.</p>

<p>The *dataspace* is also used during dataset I/O operations, defining the elements of the dataset that participate in the I/O operation.</p>

<p>This chapter explains the *dataspace* object and its use in dataset and attribute creation and data transfer. It also describes selection operations on a dataspace used to implement sub-setting, sub-sampling, and scatter-gather access to datasets.

</p>

<p>The rest of this chapter is structured as follows:</p>

- Section 2, “Dataspace Function Summaries,”
provides a categorized list of dataspace functions,
also known as the H5S APIs
- Section 3, “Definition of Dataspace Objects and
the Dataspace Programming Model,”
describes dataspace objects and the programming model,
including the creation and use of dataspace
- Section 4, “Dataspaces and Data Transfer,”
describes the use of dataspace in data transfer
- Section 5, “Dataspace Selection Operations and Data
Transfer,” describes selection operations on dataspace
and their usage in data transfer
- Section 6, “References to Dataset Regions,”
briefly discusses references to dataset regions
- Section 7, “Sample Programs,”
contains the full programs from which several of the code samples
in this chapter were derived

<SCRIPT language="JavaScript">

<!--

document.writeln ("

<div align="right">

(Top)

</div>


```
");
```

```
-->
```

```
</SCRIPT>
```

```
<br />
```

```
<!-- NEW PAGE -->
```

```
<a name="DSpaceFuncSums">
```

```
<h3 class="pagebefore">7.2. Dataspace (H5S) Function Summaries</h3>
```

```
</a>
```

```
<p>This section provides a reference list of dataspace functions,
the H5S APIs, with brief descriptions.
```

```
The functions are presented in the following catagories:</p>
```

```
<ul>
```

```
<li>Dataspace management functions</li>
```

```
<li>Dataspace query functions</li>
```

```
<li>Dataspace selection functions: hyperslabs</li>
```

```
<li>Dataspace selection functions: points</li>
```

```
</ul>
```

```
<p>The rest of the chapter will provide examples and explanations
of how to use these functions.</p>
```

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<td colspan="3" align="left" valign="bottom">
```

```
<b>Function Listing 1.
```

Dataspace management functions

<hr color="green" size="3"/>		
C Function Fortran Function		
Purpose		

<hr color="green" size="1"/>		
<code>H5Screate</code> <code>h5screate_f</code>		
Creates a new dataspace of a specified type.		

<hr color="green" size="1"/>		
<code>H5Scopy</code> <code>h5scopy_f</code>		
Creates an exact copy of a dataspace.		

<hr color="green" size="1"/>		

<td>

`<code>H5Sclose
h5sclose_f</code>`

</td><td> </td>

<td>

Releases and terminates access to a dataspace.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

`<code>H5Sdecode
h5sdecode_f</code>`

</td><td> </td>

<td>

Decode a binary object description of a dataspace and return a new object identifier.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

`<code>H5Sencode
h5sencode</code>`

</td><td> </td>

<td>

Encode a dataspace object description into a binary buffer.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

`<code>H5Screate_simple
h5screate_simple_f</code>`

</td><td> </td>

<td>

Creates a new simple dataspace and opens it for access.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Sis_simple
h5sis_simple_f</code>

</td><td> </td>

<td>

Determines whether a dataspace is a simple dataspace.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Sextent_copy
h5sextent_copy_f</code>

</td><td> </td>

<td>

Copies the extent of a dataspace.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Sextent_equal
h5sextent_equal_f</code>

</td><td> </td>

<td>

Determines whether two dataspace extents are equal.

</td>

</tr>

```

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Sset_extent_simple<br />h5sset_extent_simple_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets or resets the size of an existing dataspace.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Sset_extent_none<br />h5sset_extent_none_f</code>
  </td><td>&nbsp;</td>
  <td>
    Removes the extent from a dataspace.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="3" align="left" valign="bottom">
      <b>Function Listing 2. Dataspace query functions</b></td>
    </tr>

```

<hr/>		
C Function Fortran Function		
Purpose		
<hr/>		
<code>H5Sget_simple_extent_dims</code> <code>h5sget_simple_extent_dims_f</code>		
Retrieves dataspace dimension size and maximum size.		
<hr/>		
<code>H5Sget_simple_extent_ndims</code> <code>h5sget_simple_extent_ndims_f</code>		
Determines the dimensionality of a dataspace.		
<hr/>		
<code>H5Sget_simple_extent_npoints</code>		


```

h5sget_simple_extent_npoints_f</code>
</td><td>&nbsp;</td>
<td>
Determines the number of elements in a dataspace.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5Sget_simple_extent_type<br />h5sget_simple_extent_type_f</code>
</td><td>&nbsp;</td>
<td>
Determine the current class of a dataspace.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />
<br />

```

```

<table width="600" cellspacing="0" align="center" cellpadding="0">
<tr valign="bottom">
<td colspan="3" align="left" valign="bottom">
<b>Function Listing 3. Dataspace selection functions: hyperslabs
</b></td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
<tr valign="top">
<td>

```


Get number of hyperslab blocks.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Sget_select_hyper_blocklist

h5sget_select_hyper_blocklist_f</code>

</td><td> </td>

<td>

Gets the list of hyperslab blocks currently selected.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Sget_select_bounds
h5sget_select_bounds_f</code>

</td><td> </td>

<td>

Gets the bounding box containing the current selection.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Sselect_all
h5sselect_all_f</code>

</td><td> </td>

<td>

Selects the entire dataspace.

</td>

</tr>

```

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Sselect_none<br />h5sselect_none_f</code>
  </td><td>&nbsp;</td>
  <td>
    Resets the selection region to include no elements.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Sselect_valid<br />h5sselect_valid_f</code>
  </td><td>&nbsp;</td>
  <td>
    Verifies that the selection is within the extent of the dataspace.
  </td>
</tr>
<!-- NEW PAGE -->
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Sselect_hyperslab<br />h5sselect_hyperslab_f</code>
  </td><td>&nbsp;</td>
  <td>
    Selects a hyperslab region to add to the current selected region.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```


<table width="600" cellspacing="0" align="center" cellpadding="0">

<tr valign="bottom">

<td colspan="3" align="left" valign="bottom">

Function Listing 4. Dataspace selection functions: points

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

<tr valign="top">

<td>

C Function
Fortran Function

</td><td> </td>

<td>

Purpose

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Sget_select_npoints
h5sget_select_npoints_f</code>

</td><td> </td>

<td>

Determines the number of elements in a dataspace selection.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<code><td></code> <code><code>H5Sget_select_elem_npoints
h5sget_select_elem_npoints_f</code></code> <code></td><td>&nbsp;</td></code> <code><td></code> Gets the number of element points in the current selection. <code></td></code> <code></tr></code>	<hr/>		
<code><td></code> <code><code>H5Sget_select_elem_pointlist
</code> <code>h5sget_select_elem_pointlist_f</code></code> <code></td><td>&nbsp;</td></code> <code><td></code> Gets the list of element points currently selected. <code></td></code> <code></tr></code>	<hr/>		
<code><td></code> <code><code>H5Sselect_elements
h5sselect_elements_f</code></code> <code></td><td>&nbsp;</td></code> <code><td></code> Selects array elements to be included in the selection for a dataspace. <code></td></code> <code></tr></code>	<hr/>		

<SCRIPT language="JavaScript">

```
<!--
document.writeln ("
<a name="DefDataObjs">
<div align="right">
<a href="#TOP"><font size=-1>(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>
```

```
<br />
```

```
<!-- NEW PAGE -->
```

```
<a name="DefDataObjs">
```

```
<h3 class="pagebefore">7.3. Definition of Dataspace Objects and
the Dataspace Programming Model</h3></a>
```

<p>This section introduces the notion of the HDF5 dataspace object and a programming model for creating and working with dataspaces.</p>

```
<h4>7.3.1. Dataspace Objects</h4>
```

<p>An HDF5 dataspace is a required component of an HDF5 dataset or attribute.

A dataspace defines the size and the shape of a dataset's or an attribute's raw data.

Currently, HDF5 supports the following types of the dataspace: </p>

```
<ul>
```

```
<li>Scalar dataspace</li>
```

- Simple dataspace
- Null dataspace

A *scalar dataspace*, `H5S_SCALAR`, represents just one element, a scalar.

Note that the datatype of this one element may be very complex, e.g., a compound structure with members being of any allowed HDF5 datatype, including multidimensional arrays, strings, and nested compound structures.

By convention, the rank of a scalar dataspace is always `0` (zero); think of it geometrically as a single, dimensionless point, though that point may be complex.

A *simple dataspace*, `H5S_SIMPLE`, is a multidimensional array of elements.

The dimensionality of the dataspace (or the rank of the array) is fixed and is defined at creation time.

The size of each dimension can grow during the life time of the dataspace from the *current size* up to the *maximum size*.

Both the current size and the maximum size are specified at creation time.

The sizes of dimensions at any particular time in the life of a dataspace are called the *current dimensions*, or the *dataspace extent*. They can be queried along with the maximum sizes.

A *null dataspace*, `H5S_NULL`, contains no data elements.

Note that no selections can be applied to a null dataset as there is nothing to select.

As shown in the UML diagram in the figure below, an HDF5 simple dataspace object has three attributes: the rank or number of dimensions; the current sizes, expressed as an array of length `rank` with each element of the array denoting the current size of the corresponding dimension; and the maximum sizes, expressed as an array of length `rank` with each element of the array denoting the maximum size of the corresponding dimension.

```
<table width="400" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <table border="1">
        <tr><td align="center">
          <code>Simple dataspace</code>
        </td></tr>
        <tr><td align="left">
          <code>
            &nbsp;&nbsp;&nbsp;rank:int<br />
            &nbsp;&nbsp;&nbsp;current_size:hsize_t[rank]&nbsp;&nbsp;&nbsp;<br />
            &nbsp;&nbsp;&nbsp;maximum_size:hsize_t[rank]</code>
          </td></tr>
        <tr><td align="left">&nbsp;&nbsp;&nbsp;
          </td></tr>
      </table>
    </td></tr>
  <tr><td><hr color="green" size="1" /></td></tr>
</table>
```

--

Figure 1. A simple dataspace

<p>A simple dataspace is defined by its rank, the current size of each dimension, and the maximum size of each dimension.</p>

<hr/>

--

--

--

The size of a current dimension cannot be greater than the maximum size, which can be unlimited, specified as `H5S_UNLIMITED`.

Note that while the HDF5 file format and library impose no maximum size on an unlimited dimension, practically speaking its size will always be limited to the biggest integer available on the particular system being used.

[[Prior excessively casual phrasing replaced (...the caveat that the value of infinity is limited to...).]]

-->

Dataspace rank is restricted to 32, the standard limit in C on the rank of an array, in the current implementation of the HDF5 Library.

The HDF5 file format, on the other hand, allows any rank up to the maximum integer value on the system, so the library restriction can be raised in the future if higher dimensionality is required.

Note that most of the time Fortran applications calling HDF5 will

work with dataspace of rank less than or equal to seven,
 since seven is the maximum number of dimensions in a Fortran array.
 But dataspace rank is not limited to seven for Fortran applications.

<!-- editingComment

[[or "But with the use of XXX, Fortran applications can [easily?] work
 with dataspace rank of up to 32."]]

-->

The current dimensions of a dataspace, also referred to as the
 dataspace extent, define the bounding box for dataset elements that
 can participate in I/O operations.

<h4>7.3.2. Programming Model</h4>

<p>

The programming model for creating and working with HDF5 dataspace
 can be summarized as follows:

Create a dataspace

Use the dataspace to create a dataset in the file or
 to describe a data array in memory

Modify the dataspace to define dataset elements that will
 participate in I/O operations

Use the modified dataspace while reading/writing dataset raw
 data or to create a region reference

Close the dataspace when no longer needed

The rest of this section will address steps 1, 2, and 5 of the programming model; steps 3 and 4 will be discussed in later sections of this chapter.

7.3.2.1. Creating a Dataspace

A dataspace can be created by calling the `H5Screate` function (`h5screate_f` in Fortran). Since the definition of a simple dataspace requires the specification of dimensionality (or rank) and initial and maximum dimension sizes, the HDF5 Library provides a *convenience* API, `H5Screate_simple` (`h5screate_simple_f`) to create a simple dataspace in one step.

The following examples illustrate the usage of these APIs.

7.3.2.2. Creating a Scalar Dataspace

A scalar dataspace is created with the `H5Screate` or the `h5screate_f` function.

In C:

```
hid_t space_id;
...
space_id = H5Screate(H5S_SCALAR);
```

<p>In Fortran:</p>

<pre>

```
    INTEGER(HID_T) :: space_id  
    ...  
    CALL h5screate_f(H5S_SCALAR_F, space_id, error)
```

</pre>

<p>As mentioned above, the dataspace will contain only one element.

Scalar dataspaces are used more often for describing attributes

that have just one value, e.g. the attribute

temperature with the value

celsius is used to indicate that the

dataset with this attribute stores temperature values using the celsius scale.</p>

<h4>7.3.2.3. Creating a Null Dataspace</h4>

<p>A null dataspace is created with the <code>H5Screate</code>

or the <code>h5screate_f</code> function.</p>

<p>In C:</p>

<pre>

```
    hid_t space_id;  
    ...  
    space_id = H5Screate(H5S_NULL);
```

</pre>

<p>In Fortran: </p>

(<code>H5S_NULL</code> not yet implemented in Fortran.)

<pre>

```
    INTEGER(HID_T) :: space_id
```

```
...  
CALL h5screate_f(H5S_NULL_F, space_id, error)  
</pre>
```

As mentioned above, the dataspace will contain no elements.

```
<!--  
    NEED MORE INFO.  
    SPECIFICALLY, HOW ARE SUCH DATASPACEs USED?  
    AND WHAT ATTRIBUTES ARE RELEVANT?  
-->
```

<h4>7.3.2.4. Creating a Simple Dataspace</h4>

<p>Let's assume that an application wants to store a two-dimensional array of data, A(20,100).

During the life of the application, the first dimension of the array can grow up to 30; there is no restriction on the size of the second dimension.

The following steps are used to declare a dataspace for the dataset in which the array data will be stored.</p>

<p>In C:</p>

```
<pre>  
    hid_t space_id;  
    int rank = 2;  
    hsize_t current_dims[2] = {20, 100};  
    hsize_t max_dims[2] = {30, H5S_UNLIMITED};  
    ...
```

```
space_id = H5Screate(H5S_SIMPLE);
H5Sset_extent_simple(space_id,rank,current_dims,max_dims);
</pre>
```

<p>In Fortran:</p>

```
<pre>
    INTEGER(HID_T) :: space_id
    INTEGER :: rank = 2
    INTEGER(HSIZE_T) :: current_dims = /( 20, 100)/
    INTEGER(HSIZE_T) :: max_dims = /(30, H5S_UNLIMITED_F)/
    INTEGER error
    ...
    CALL h5screate_f(H5S_SIMPLE_F, space_id, error)
    CALL h5sset_extent_simple_f(space_id, rank, current_dims, max_dims, error)
</pre>
```

<p>Alternatively, the convenience APIs

H5Screate_simpleh5screate_simple_f
can replace the H5Screateh5screate_f
and H5Sset_extent_simpleh5sset_extent_-
simple_f
calls.</p>

<p>In C:</p>

```
<pre>
    space_id = H5Screate_simple(rank, current_dims, max_dims);
</pre>
```

<p>In Fortran:</p>

```
<pre>
    CALL h5screate_simple_f(rank, current_dims, space_id, error, max_dims)
```

```
</pre>
```

In this example, a dataspace with current dimensions of 20 by 100 is created. The first dimension can be extended only up to 30. The second dimension, however, is declared unlimited; it can be extended up to the largest available integer value on the system.

Note that when there is a difference between the current dimensions and the maximum dimensions of an array, then chunking storage must be used. In other words, if the number of dimensions may change over the life of the dataset, then chunking must be used. If the array dimensions are fixed (if the number of current dimensions is equal to the maximum number of dimensions when the dataset is created), then contiguous storage can be used. See the “

[Data Transfer](10_Datasets.html#DTransfer)” section in the “[” chapter.](UG_frame10Datasets.html)

Maximum dimensions can be the same as current dimensions. In such a case, the sizes of dimensions cannot be changed during the life of the dataspace object.

In C, `NULL` can be used to indicate to the `H5Screate_simple` and `H5Sset_extent_simple` functions that the maximum sizes of all dimensions are the same as the current sizes.

In Fortran, the maximum size parameter is optional for `h5screate_simple_f` and can be omitted when the sizes are the same.

<p>In C:</p>

```
<pre>
```

```
    space_id = H5Screate_simple(rank, current_dims, NULL);
```

```
</pre>
```

<p>In Fortran:</p>

```
<pre>
```

```
    CALL h5screate_f(rank, current_dims, space_id, error)
```

```
</pre>
```

<p>The created dataspace will have current and maximum dimensions of 20 and 100 correspondingly, and the sizes of those dimensions cannot be changed.</p>

<h4>7.3.2.5. C versus Fortran Dataspaces</h4>

<p>Dataspace dimensions are numbered from 1 to

rank. HDF5 uses C storage conventions, assuming that the last listed dimension is the fastest-changing dimension and the first-listed dimension is the slowest changing.

<!-- editingComment

[[Fortran, on the other hand, ?]]

-->

The HDF5 file format storage layout specification adheres to the C convention and the HDF5 Library adheres to the same convention when storing dataspace dimensions in the file.

This affects how C programs and tools interpret data written from Fortran programs and vice versa.

The example below illustrates the issue.</p>

When a Fortran application describes a dataspace to store an array as A(20,100), it specifies the value of the first dimension to be 20 and the second to be 100.

Since Fortran stores data by columns, the first-listed dimension with the value 20 is the fastest-changing dimension and the last-listed dimension with the value 100 is the slowest-changing.

In order to adhere to the HDF5 storage convention, the HDF5 Fortran wrapper transposes dimensions, so the first dimension becomes the last. The dataspace dimensions stored in the file will be 100,20 instead of 20,100 in order to correctly describe the Fortran data that is stored in 100 columns, each containing 20 elements.

When a Fortran application reads the data back, the HDF5 Fortran wrapper transposes the dimensions once more, returning the first dimension to be 20 and the second to be 100, describing correctly the sizes of the array that should be used to read data in the Fortran array A(20,100).

When a C application reads data back, the dimensions will come out as 100 and 20, correctly describing the size of the array to read data into, since the data was written as 100 records of 20 elements each. Therefore C tools such as `h5dump` and `h5ls` always display transposed dimensions and values for the data written by a Fortran application.

Consider the following simple example of equivalent C 3 x 5 and Fortran 5 x 3 arrays.

As illustrated in the figure <!-- formerly Figure 3 -->below,

a C applications will store

a 3 x 5 2-dimensional array as three 5-element rows.

In order to store the same data in the same order,

a Fortran application must view the array as as a 5 x 3 array with three 5-element columns.

The dataspace of this dataset, as written from Fortran,

will therefore be described as 5 x 3 in the application

but stored and described in the file according to the C convention

as a 3 x 5 array.

This ensures that C and Fortran applications will always read

the data in the order in which it was written.

The HDF5 Fortran interface handles this transposition automatically.</p>

<p>In C (from <code>h5_write.c</code>):</p>

<pre>

```

#define NX    3          /* dataset dimensions */
#define NY    5

...

int    data[NX][NY];    /* data to write */
...
/*
 * Data and output buffer initialization.
 */
for (j = 0; j < NX; j++) {
    for (i = 0; i < NY; i++)
        data[j][i] = i + 1 + j*NY;
}
/*
 * 1 2 3 4 5
 * 6 7 8 9 10

```

The HDF Group

685

```

* 11 12 13 14 15

*/

...

dims[0] = NX;
dims[1] = NY;
dataspace = H5Screate_simple(RANK, dims, NULL);
</pre>

```

<!-- NEW PAGE -->

<p>In Fortran (from <code>h5_write.f90</code>:</p>

```

<pre>
    INTEGER, PARAMETER :: NX = 3
    INTEGER, PARAMETER :: NY = 5
    ...
    INTEGER(HSIZE_T), DIMENSION(2) :: dims = (/3,5/) ! Dataset dimensions
    ---
    INTEGER    :: data(NX,NY)
    ...
    !
    ! Initialize data
    !
    do i = 1, NX
        do j = 1, NY
            data(i,j) = j + (i-1)*NY
        enddo
    enddo
    !
    ! Data
    !
    ! 1 2 3 4 5
    ! 6 7 8 9 10

```

```
! 11 12 13 14 15
```

```
...
```

```
CALL h5screate_simple_f(rank, dims, dspace_id, error)
```

```
</pre>
```

In Fortran (from [h5_write_tr_f90](#h5_write_tr_f90)

`h5_write_tr.f90`):

```
<pre>
```

```
INTEGER, PARAMETER :: NX = 3
```

```
INTEGER, PARAMETER :: NY = 5
```

```
...
```

```
INTEGER(HSIZE_T), DIMENSION(2) :: dims = (/NY, NX/) ! Dataset dimensions
```

```
...
```

```
!
```

```
! Initialize data
```

```
!
```

```
do i = 1, NY
```

```
do j = 1, NX
```

```
data(i,j) = i + (j-1)*NY
```

```
enddo
```

```
enddo
```

```
!
```

```
! Data
```

```
!
```

```
! 1 6 11
```

```
! 2 7 12
```

```
! 3 8 13
```

```
! 4 9 14
```

```
! 5 10 15
```

```
...
```

```
CALL h5screate_simple_f(rank, dims, dspace_id, error)
```

```
</pre>
```

```
<br />
```

```
<!-- NEW PAGE -->
```

```
<table width="600" cellpadding="0" align="center">
```

```
<tr valign="top">
```

```
<td align="center">
```

```
<hr color="green" size="3"/>
```

```
<br />
```

```
<table border="0" width="95%">
```

```
<tr align="left" valign="top">
```

```
<td width="35%">
```

```
<p>A dataset stored by a<br />C program in a 3 x 5 array:</p>
```

```
<table border="1" width="100%">
```

```
<tr align="center">
```

```
<td width="20%">1</td>
```

```
<td width="20%">2</td>
```

```
<td width="20%">3</td>
```

```
<td width="20%">4</td>
```

```
<td width="20%">5</td>
```

```
</tr>
```

```
<tr align="center">
```

```
<td width="20%">6</td>
```

```
<td width="20%">7</td>
```

```
<td width="20%">8</td>
```

```
<td width="20%">9</td>
```

```

    <td width="20%">10</td>
</tr>
<tr align="center">
    <td width="20%">11</td>
    <td width="20%">12</td>
    <td width="20%">13</td>
    <td width="20%">14</td>
    <td width="20%">15</td>
</tr>
</table>
</td>
<td width="15%">&nbsp;</td>
<td width="45%">
    <p>The same dataset stored by a<br />Fortran program in a 5 x 3 array:</p>
    <table border="1" width="48%">
    <tr align="center">
        <td width="33%">1</td>
        <td width="34%">6</td>
        <td width="33%">11</td>
    </tr>
    <tr align="center">
        <td width="33%">2</td>
        <td width="34%">7</td>
        <td width="33%">12</td>
    </tr>
    <tr align="center">
        <td width="33%">3</td>
        <td width="34%">8</td>
        <td width="33%">13</td>
    </tr>
    <tr align="center">

```

```

<td width="33%">4</td>
<td width="34%">9</td>
<td width="33%">14</td>
</tr>
<tr align="center">
<td width="33%">5</td>
<td width="34%">10</td>
<td width="33%">15</td>
</tr>
</table>
</td>
<td width="5%">&nbsp;</td>
</tr>
</table>

```

```

<table width="95%" align="center">

```

```

<tr>

```

```

<td align="left"><br />

```

The left-hand dataset above as written to an HDF5
file from C or the right-hand dataset as written from Fortran:

```

</td>

```

```

</tr>

```

```

</table>

```

```

<br />

```

```

<table border="1" width="95%">

```

```

<tr>

```

```

<td align="center" width="6.5%">1</td>

```

```

<td align="center" width="6.5%">2</td>

```

```

<td align="center" width="6.5%">3</td>

```

```

<td align="center" width="6.5%">4</td>

```

```

<td align="center" width="6.5%">5</td>

```



```

<td align="center" width="6.5%">6</td>
<td align="center" width="6.5%">7</td>
<td align="center" width="6.5%">8</td>
<td align="center" width="6.5%">9</td>
<td align="center" width="6.5%">10</td>
<td align="center" width="6.5%">11</td>
<td align="center" width="6.5%">12</td>
<td align="center" width="6.5%">13</td>
<td align="center" width="6.5%">14</td>
<td align="center" width="6.5%">15</td>
</tr>
</table>

```

```

<table width="95%" align="center">
  <tr>
    <td align="left"><br />
      The left-hand dataset above as written to an HDF5
      file from Fortran:
    </td>
  </tr>
</table>
<br />

```

```

<table border="1" width="95%">
<tr>
  <td align="center" width="6.5%">1</td>
  <td align="center" width="6.5%">6</td>
  <td align="center" width="6.5%">11</td>
  <td align="center" width="6.5%">2</td>
  <td align="center" width="6.5%">7</td>
  <td align="center" width="6.5%">12</td>

```

```

<td align="center" width="6.5%">3</td>
<td align="center" width="6.5%">8</td>
<td align="center" width="6.5%">13</td>
<td align="center" width="6.5%">4</td>
<td align="center" width="6.5%">9</td>
<td align="center" width="6.5%">14</td>
<td align="center" width="6.5%">5</td>
<td align="center" width="6.5%">10</td>
<td align="center" width="6.5%">15</td>
</tr>
</table>
<br />
</td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 2. Comparing C and Fortran dataspaces
    <!-- formerly Figure 3 --></b><br />
    The HDF5 Library stores arrays along the fastest-changing
    dimension. This approach is often referred to as being “in
    C order.” C, C++, and Java work with arrays in row-major
    order. In other words, the row, or the last dimension, is the
    fastest-changing dimension. Fortran, on the other hand, handles
    arrays in column-major order making the column, or the first
    dimension, the fastest-changing dimension. Therefore, the C and
    Fortran arrays illustrated in the top portion of this figure are
    stored identically in an HDF5 file. This ensures that data
    written by any language can be meaningfully read, interpreted,
    and manipulated by any other.
    <hr color="green" size="3" /></td>
  </tr>

```

```
</table>
```

```
<br />
```

```
<!-- editingComment this entire section needs to be -->
```

```
<!-- written properly then reincluded.      -->
```

```
<!--
```

```
<h4>Extending a dataspace</h4>
```

```
<p>
```

```
<span class="editingComment">[ [ Text here describing the extension of the dataset, per the following
diagram. ] ]</span>
```

```
<p>
```

```
<table x-use-null-cells
```

```
width="600"
```

```
cellspacing="0"
```

```
      class="fullImgTable"
```

```
align="center">
```

```
<tr valign="top">
```

```
<td class="fullImgTableImgCell" align="center">
```

```

```

```
</td></tr>
```

```
<tr>
```

```
<td align="center" class="fullImgTableCapCell">
```

```
<span class="figurenumber">Figure 4</span>
```

```
Extend the simple dataspace array to 12x130
```

```
-->
```

```
<!-- The next 10 lines were already commented out -->
```


[[Isn't that all of them? What other types are there?]]
-->

<p>To find out the dimensionality, or rank, of a dataspace,
use H5Sget_simple_extent_ndims
(h5sget_simple_extent_ndims_f).
H5Sget_simple_extent_dims
can also be used to find out the rank.

See the example below.

If both functions return 0 for the
value of rank, then the dataspace is scalar.</p>

<p>To query the sizes of the current and maximum dimensions,
use H5Sget_simple_extent_dims
(h5sget_simple_extent_dims_f). </p>

<p>The following example illustrates querying the rank and dimensions
of a dataspace using these functions. </p>

<!-- NEW PAGE -->

<p>In C:</p>

<pre>

```
hid_t space_id;
int rank;
hsize_t *current_dims;
hsize_t *max_dims;
```

```
-----
```

```
rank=H5Sget_simple_extent_ndims(space_id);
    (or rank=H5Sget_simple_extent_dims(space_id, NULL, NULL);)
current_dims= (hsize_t)malloc(rank*sizeof(hsize_t));
```

```
max_dims=(hsize_t)malloc(rank*sizeof(hsize_t));
H5Sget_simple_extent_dims(space_id, current_dims, max_dims);
Print values here for the previous example
```

```
</pre>
```

```
<!-- editingComment
```

```
<p>In Fortran:
```

```
<pre>
```

```
Example ??????????
```

```
</pre>
```

```
-->
```

```
<SCRIPT language="JavaScript">
```

```
<!--
```

```
document.writeln ("
```

```
<a name="DTransfer">
```

```
<div align="right">
```

```
<a href="#TOP"><font size=-1>(Top)</font></a>
```

```
</div>
```

```
</a>
```

```
";
```

```
-->
```

```
</SCRIPT>
```

```
<br />
```

```
<!-- NEW PAGE -->
```

```
<a name="DTransfer">
```

```
<h3 class="pagebefore">7.4. Dataspaces and Data Transfer</h3>
```

```
</a>
```



```

<tr valign="top">
  <td align="left" >
    <b>Figure 3. Data layout before and after a read operation
    <!-- formerly Figure 4 --></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
<br />

```

```

<!-- NEW PAGE -->

```

```

<table width="500" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td></tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 4. Moving data from disk to memory
      <!-- formerly Figure 5 --></b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```


Both the source and destination are stored as contiguous blocks of storage with the elements in the order specified by the `dataspace`.

The figure above formerly Figure 5 shows one way the elements might be organized. In item a formerly Figure 5a, the elements are stored as 3 blocks of 4 elements. The destination is an array of 12 elements in memory (see item c formerly Figure 5c).

As the figure suggests, the transfer reads the disk blocks into a memory buffer (see item b formerly Figure 5b), and then writes the elements to the correct locations in memory. A similar process occurs in reverse when data is written to disk.

7.4.1. Data Selection

In addition to rearranging data, the transfer may select the data elements from the source and destination.

Data selection is implemented by creating a `dataspace` object that describes the selected elements (within the hyper rectangle) rather than the whole array. Two `dataspace` objects with selections can be used in data transfers to read selected elements from the source and write selected elements to the destination. When data is transferred using the `dataspace` object, only the selected elements will be transferred.

This can be used to implement partial I/O, including:

-

- Sub-setting - reading part of a large dataset

- Sampling - reading selected elements (e.g., every second element of a dataset)

- Scatter-gather - read non-contiguous elements into contiguous locations
(gather) or read contiguous elements into non-contiguous locations
(scatter) or both

To use selections, the following steps are followed:

- Get or define the dataspace for the source and destination
- Specify one or more selections for source and destination dataspace
- Transfer data using the dataspace with selections

NEW PAGE

A selection is created by applying one or more selections to a `dataspace`. A selection may override any other selections (`H5T_SELECT_SET`) or may be “Ored” with previous selections on the same dataspace (`H5T_SELECT_OR`).

In the latter case, the resulting selection is the union of the selection and all previously selected selections. Arbitrary sets of points from a dataspace can be selected by specifying an appropriate set of selections.

Two selections are used in data transfer, so the source and destination must be compatible, as described below.

There are two forms of selection, hyperslab and point. A selection must be either a point selection or a set of hyperslab selections. Selections cannot be mixed.

The definition of a selection within a dataspace, not the data

in the selection, cannot be saved to the file unless the selection definition is saved as a region reference. See the ["References to Dataset Regions"](#DRegions) section for more information.

7.4.1.1. Hyperslab selection

A hyperslab is a selection of elements from a hyper rectangle. An HDF5 hyperslab is a rectangular pattern defined by four arrays. The four arrays are summarized in the table below formerly Table 1--.

The *offset* defines the origin of the hyperslab in the original dataspace.

The *stride* is the number of elements to increment between selected elements. A stride of 1 is every element, a stride of 2 is every second element, etc. Note that there may be a different stride for each dimension of the dataspace. The default stride is 1.

The *count* is the number of elements in the hyperslab selection. When the stride is 1, the selection is a hyper rectangle with a corner at the offset and size count[0] by count[1] by.... When stride is greater than one, the hyperslab bounded by the offset and the corners defined by stride[n] * count[n].

--	--

--	--

--	--

--	--

--	--

--	--


```

<tr valign="top">
  <td width="15%"><b>Parameter</b></td>
  <td width="85%"><b>Description</b></td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Offset</td>
  <td>The starting location for the hyperslab.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Stride</td>
  <td>The number of elements to separate each element or block
  to be selected.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Count</td>
  <td>The number of elements or blocks to select along each
  dimension.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Block</td>
  <td>The size of the block selected from the dataspace.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

The **block** is a count on the number of repetitions of the hyperslab. The default block size is `1`, which is one hyperslab. A block of 2 would be two hyperslabs in that dimension, with the second starting at $\text{offset}[n] + (\text{count}[n] * \text{stride}[n]) + 1$.

A hyperslab can be used to access a sub-set of a large dataset. The figure below formerly Figure 6 shows an example of a hyperslab that reads a rectangle from the middle of a larger two dimensional array. The destination is the same shape as the source.

<div style="text-align: center;"> <hr style="border: 1px solid green; width: 300px;"/>  </div>
<div style="text-align: center;"> <hr style="border: 1px solid green; width: 100px;"/> </div>
<div style="text-align: left;"> <p>Figure 5. Access a sub-set of data with a hyperslab<small>formerly Figure 6</small></p> <hr style="border: 1px solid green; width: 300px;"/> </div>

Hyperslabs can be combined to select complex regions of the source and

destination. The figure below <!-- formerly Figure 7 -->shows an example of a transfer from one non-rectangular region into another non-rectangular region. The source is defined as the union of two hyperslabs, and the destination is the union of three hyperslabs.</p>

```
<table width="500" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td></tr>
    <tr><td><hr color="green" size="1" /></td></tr>
    <tr valign="top">
      <td align="left" >
        <b>Figure 6. Build complex regions with
        hyperslab unions<!-- formerly Figure 7--></b>
        <hr color="green" size="3"/></td>
      </tr>
</table>
<br />
```

<!-- NEW PAGE -->

<p>Hyperslabs may also be used to collect or scatter data from regular patterns. The figure below <!-- formerly Figure 8 -->shows an example where the source is a repeating pattern of blocks, and the destination is a single, one dimensional array.</p>

```

<table width="500" cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td></tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 7. Use hyperslabs to combine or disperse data
      <!-- formerly Figure 8--></b>
      <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

<h4>7.4.1.2. Select Points</h4>

<p>The second type of selection is an array of points, i.e., coordinates.

Essentially, this selection is a list of all the points to include.

The figure below <!-- formerly Figure 9 -->shows an example of a transfer of seven elements from a two dimensional dataspace to a three dimensional dataspace using a point selection to specify the points.</p>

```

<table width="500" cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td></tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 7. Use hyperslabs to combine or disperse data
      <!-- formerly Figure 8--></b>
      <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

```

    </td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left" >
        <b>Figure 8. Point selection
        <!-- formerly Figure 9--></b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

<h4>7.4.1.3. Rules for Defining Selections</h4>

<p>A selection must have the same number of dimensions (rank) as the dataspace it is applied to, although it may select from only a small region, e.g., a plane from a 3D dataspace. Selections do not affect the extent of the dataspace, the selection may be larger than the dataspace. The boundaries of selections are reconciled with the extent at the time of the data transfer.</p>

<h4>7.4.1.4. Data Transfer with Selections</h4>

<p>A data transfer (read or write) with selections is the same as any read or write, except the source and destination dataspace have compatible selections. </p>

<p>During the data transfer, the following steps are executed by the library:</p>

The source and destination `dataspace` are checked to assure that the selections are compatible.

Each selection must be within the current extent of the `dataspace`.

A selection may be defined to extend outside the current extent of the `dataspace`, but the `dataspace` cannot be accessed if the selection is not valid at the time of the access.

The total number of points selected in the source and destination must be the same. Note that the dimensionality of the source and destination can be different (e.g., the source could be 2D, the destination 1D or 3D), and the shape can be different, but the number of elements selected must be the same.

The data is transferred, element by element.

<p>Selections have an iteration order for the points selected, which can be any permutation of the dimensions involved (defaulting to `‘C’`; array order) or a specific order for the selected points, for selections composed of single array elements with `<code>H5Sselect_elements</code>`.</p>

<p>The elements of the selections are transferred in row-major, or C order. That is, it is assumed that the first dimension varies slowest, the second next slowest, and so forth. For hyperslab selections, the order can be any permutation of the dimensions involved (defaulting to `‘C’`; array order). When multiple hyperslabs are combined, the hyperslabs are coalesced into contiguous reads and writes.</p>

In the case of point selections, the points are read and written in the order specified.

7.4.2. Programming Model

7.4.2.1. Selecting Hyperslabs

Suppose we want to read a 3x4 hyperslab from a dataset in a file beginning at the element <1,2> in the dataset, and read it into a 7 x 7 x 3 array in memory. See the figure below. <!-- formerly (Figure 10).--> In order to do this, we must create a dataspace that describes the overall rank and dimensions of the dataset in the file as well as the position and size of the hyperslab that we are extracting from that dataset.

<!-- NEW PAGE -->

--

--

--



--

--

--

Figure 9. Selecting a hyperslab

<!-- formerly Figure 10-->

The code in the first example below <!-- formerly Figure 11 --> illustrates the selection of the hyperslab in the file dataspace. The second example below <!-- formerly Figure 12 --> shows the definition of the destination dataspace in memory. Since the in-memory dataspace has three dimensions, the hyperslab is an array with three dimensions with the last dimension being 1: $[3,4,1]$. The third example below <!-- formerly Figure 13 --> shows the read using the source and destination dataspaces with selections.</p>

<pre> /* * get the file dataspace. */ dataspace = H5Dget_space(dataset); /* dataspace identifier */ /* * Define hyperslab in the dataset. */ offset[0] = 1; offset[1] = 2; </pre>

```

count[0] = 3;
count[1] = 4;
status = H5Sselect_hyperslab(dataspace, H5S_SELECT_SET, offset, NULL,
    count, NULL);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 1. Selecting a hyperslab
        <!-- formerly Figure 11--></b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />
<br />

```

```

<!-- NEW PAGE -->

```

```

<table width="600" cellpadding="0" align="center">
    <tr valign="top">
        <td align="left">
            <hr color="green" size="3"/>
            <pre>
/*
* Define memory dataspace.
*/
dimsm[0] = 7;
dimsm[1] = 7;
dimsm[2] = 3;

```

```

memspace = H5Screate_simple(3,dimsm,NULL);

/*
 * Define memory hyperslab.
 */
offset_out[0] = 3;
offset_out[1] = 0;
offset_out[2] = 0;
count_out[0] = 3;
count_out[1] = 4;
count_out[2] = 1;
status = H5Sselect_hyperslab(memspace, H5S_SELECT_SET, offset_out, NULL,
    count_out, NULL);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 2. Defining the destination memory
    <!-- formerly Figure 12--></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
<br />

```

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>

```

```

ret = H5Dread(dataset, H5T_NATIVE_INT, memspace, dataspace, H5P_DEFAULT,
    data);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 3. A sample read specifying source
        and destination dataspaces
        <!-- formerly Figure 13--></b>
        <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

<h4>7.4.2.2. Example with Strides and Blocks</h4>

<p>Consider an 8 x 12 dataspace into which we want to write eight 3 x 2 blocks in a two dimensional array from a source dataspace in memory that is a 50-element one dimensional array.

See the figure below.<!-- formerly (Figure 14).--></p>

```

<!-- NEW PAGE -->
<table width="600" cellspacing="0" align="center">
    <tr valign="top">
        <td align="center">
            <hr color="green" size="3"/>
        </td>
    </tr>
</table>

```

```

<tr>
<td width="80">&nbsp;</td>
<td width="440">a) The source is a 1D array with 50 elements</td>
<td width="80">&nbsp;</td>
</tr>
<tr valign="top"><td>&nbsp;</td>
<td align="center">

</td><td>&nbsp;</td></tr>
<tr>
<td width="80">&nbsp;</td>
<td width="440">b) The destination on disk is a 2D array
with 48 selected elements</td>
<td width="80">&nbsp;</td>
</tr>
</table>
</td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td align="left" >
<b>Figure 10. Write from a one dimensional
array to a two dimensional array<!-- Figure 14--></b>
<hr color="green" size="3"/></td>
</tr>
</table>
<br />

```

The example below <!-- formerly Figure 15 -->shows code to write 48 elements from the one dimensional array to the file dataset starting with the second element in vector. The destination

hyperslab has the following parameters: offset=(0,1), stride=(4,3), count=(2,4), block=(3,2). The source has the parameters: offset=(1), stride=(1), count=(48), block=(1). After these operations, the file dataspace will have the values shown in item b in the figure above

<!-- formerly Figure 14-->. Notice that the values are inserted in the file dataset in row-major order.</p>

<!-- NEW PAGE -->

<table width="600" cellpadding="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

/* Select hyperslab for the dataset in the file, using 3 x 2 blocks, (4,3) stride

* (2,4) count starting at the position (0,1).

*/

offset[0] = 0; offset[1] = 1;

stride[0] = 4; stride[1] = 3;

count[0] = 2; count[1] = 4;

block[0] = 3; block[1] = 2;

ret = H5Sselect_hyperslab(fid, H5S_SELECT_SET, offset, stride, count, block);

/*

* Create dataspace for the first dataset.

*/

mid1 = H5Screate_simple(MSPACE1_RANK, dim1, NULL);

/*

* Select hyperslab.

* We will use 48 elements of the vector buffer starting at the second element.

* Selected elements are 1 2 3 . . . 48


```

*/
offset[0] = 1;
stride[0] = 1;
count[0] = 48;
block[0] = 1;
ret = H5Sselect_hyperslab(mid1, H5S_SELECT_SET, offset, stride, count, block);

/*
 * Write selection from the vector buffer to the dataset in the file.
 *
ret = H5Dwrite(dataset, H5T_NATIVE_INT, midd1, fid, H5P_DEFAULT, vector)</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 4. Write from a one dimensional
    array to a two dimensional array
    <!-- formerly Figure 15--></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

<h4>7.4.2.3. Selecting a Union of Hyperslabs</h4>

<p>The HDF5 Library allows the user to select a union of hyperslabs and write or read the selection into another selection. The shapes of the two selections may differ, but the number of elements must be equal. </p>

```

<!-- NEW PAGE -->
<table width="400" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
      <br />
      
      <br />
      
    </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 11. Transferring hyperslab unions
      <!-- formerly Figure 16--></b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

The figure above <!-- formerly Figure 16 --> shows the transfer of a selection that is two overlapping hyperslabs from the dataset into a union of hyperslabs in the memory dataset. Note that the destination dataset has a different shape from the source dataset. Similarly, the selection in the memory dataset could have a different shape than the selected union of hyperslabs in the original file. For simplicity,

the selection is that same shape at the destination.

To implement this transfer, it is necessary to:

- Get the source dataspace
- Define one hyperslab selection for the source
- Define a second hyperslab selection, unioned with the first
- Get the destination dataspace
- Define one hyperslab selection for the destination
- Define a second hyperslab selection, unioned with the first
- Execute the data transfer (`H5Dread` or `H5Dwrite`) using the source and destination dataspace

The example below formerly Figure 17 shows example code to create the selections for the source

dataspace (the file). The first hyperslab is size 3 x 4 and the left upper corner at the position (1,2). The hyperslab is a simple rectangle, so the stride and block are 1. The second hyperslab is 6 x 5 at the position (2,4). The second selection is a union with the first hyperslab (`H5S_SELECT_OR`).

| |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> </div> </div> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

    <pre>
    fid = H5Dget_space(dataset);

    /*
    * Select first hyperslab for the dataset in the file.
    *
    */
    offset[0] = 1; offset[1] = 2;
    block[0] = 1; block[1] = 1;
    stride[0] = 1; stride[1] = 1;
    count[0] = 3; count[1] = 4;
    ret = H5Sselect_hyperslab(fid, H5S_SELECT_SET, offset, stride, count, block);
    /*
    * Add second selected hyperslab to the selection.
    */
    offset[0] = 2; offset[1] = 4;
    block[0] = 1; block[1] = 1;
    stride[0] = 1; stride[1] = 1;
    count[0] = 6; count[1] = 5;
    ret = H5Sselect_hyperslab(fid, H5S_SELECT_OR, offset, stride, count, block);</pre>
    </td></tr>
    <tr><td><hr color="green" size="1" /></td></tr>
    <tr valign="top">
    <td align="left">
    <b>Example 5. Select source hyperslabs
    <!-- formerly Figure 17 --></b>
    <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

The example below `<!-- formerly Figure 18 -->` shows example code to create the selection for the destination in memory. The steps are similar. In this example, the hyperslabs are the same shape, but located in different positions in the dataspace. The first hyperslab is 3 x 4 and starts at (0,0), and the second is 6 x 5 and starts at (1,2).

Finally, the `H5Dread` call transfers the selected data from the file dataspace to the selection in memory.

In this example, the source and destination selections are two overlapping rectangles. In general, any number of rectangles can be OR'ed, and they do not have to be contiguous. The order of the selections does not matter, but the first should use `H5S_SELECT_SET`; subsequent selections are unioned using `H5S_SELECT_OR`.

It is important to emphasize that the source and destination do not have to be the same shape (or number of rectangles). As long as the two selections have the same number of elements, the data can be transferred.

`<!-- NEW PAGE -->`

`<table width="600" cellpadding="0" align="center">`

`<tr valign="top">`

`<td align="left">`

`<hr color="green" size="3"/>`

`<pre>`

`/*`

```

* Create memory dataspace.
*/
mid = H5Screate_simple(MSPACE_RANK, mdim, NULL);

/*
* Select two hyperslabs in memory. Hyperslabs has the same
* size and shape as the selected hyperslabs for the file dataspace.
*/
offset[0] = 0; offset[1] = 0;
block[0] = 1; block[1] = 1;
stride[0] = 1; stride[1] = 1;
count[0] = 3; count[1] = 4;
ret = H5Sselect_hyperslab(mid, H5S_SELECT_SET, offset, stride, count, block);
offset[0] = 1; offset[1] = 2;
block[0] = 1; block[1] = 1;
stride[0] = 1; stride[1] = 1;
count[0] = 6; count[1] = 5;
ret = H5Sselect_hyperslab(mid, H5S_SELECT_OR, offset, stride, count, block);

ret = H5Dread(dataset, H5T_NATIVE_INT, mid, fid, H5P_DEFAULT, matrix_out);</pre>
</td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 6. Select destination hyperslabs</b>
    <!-- formerly Figure 18--></b>
    <hr color="green" size="3" /></td>
  </tr>
</table>
<br />

```

7.4.2.4. Selecting a List of Independent Points

It is also possible to specify a list of elements to read or write using the function `H5Sselect_elements`. The procedure is similar to hyperslab selections.

``

`Get the source dataspace`

`Set the selected points`

`Get the destination dataspace`

`Set the selected points`

`Transfer the data using the source and destination dataspace`

`
`

The figure below `<!-- formerly Figure 19 -->` shows an example where four values are to be written to four separate points in a two dimensional dataspace. The source dataspace is a one dimensional array with the values 53, 59, 61, 67. The destination dataspace is an 8 x 12 array. The elements are to be written to the points (0,0), (3,3), (3,5), and (5,6). In this example, the source does not require a selection. The example below the figure `<!-- formerly Figure 20 -->` shows example code to implement this transfer.

A point selection lists the exact points to be transferred and the order they will be transferred. The source and destination are required to have the same number of elements. A point selection can be used with a hyperslab (e.g., the source could be a point selection and the destination a hyperslab,

or vice versa), so long as the number of elements selected are the same.

```
<table width="400" cellpadding="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="center">
```

```
      <hr color="green" size="3"/>
```

```
      
```

```
      <br />
```

```
      
```

```
      <br />
```

```
      
```

```
    </td></tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
  <td align="left" >
```

```
    <b>Figure 12. Write data to separate points
```

```
    <!-- formerly Figure 19--></b>
```

```
    <hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

```
<br />
```

```
<table width="600" cellpadding="0" align="center">
```

```
  <tr valign="top">
```

```
    <td align="left">
```

```
      <hr color="green" size="3"/>
```

```
      <pre>
```



```

hsize_t dim2[] = {4};
int    values[] = {53, 59, 61, 67};

hssize_t coord[4][2]; /* Array to store selected points
                        from the file dataspace */

/*
 * Create dataspace for the second dataset.
 */
mid2 = H5Screate_simple(1, dim2, NULL);

/*
 * Select sequence of NPOINTS points in the file dataspace.
 */
coord[0][0] = 0; coord[0][1] = 0;
coord[1][0] = 3; coord[1][1] = 3;
coord[2][0] = 3; coord[2][1] = 5;
coord[3][0] = 5; coord[3][1] = 6;

ret = H5Sselect_elements(fid, H5S_SELECT_SET, NPOINTS,
                        (const hssize_t **)coord);

ret = H5Dwrite(dataset, H5T_NATIVE_INT, mid2, fid, H5P_DEFAULT, values);</pre>
</td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 7. Write data to separate points</b>
    <!-- formerly Figure 20 --></b>
    <hr color="green" size="3"/></td>

```

```
</tr>
</table>
<br />
```

```
<!-- NEW PAGE -->
```

<h4>7.4.2.5. Combinations of Selections</h4>

<p>Selections are a very flexible mechanism for reorganizing data during a data transfer. With different combinations of dataspaces and selections, it is possible to implement many kinds of data transfers including sub-setting, sampling, and reorganizing the data. The table below <!-- formerly Table 2 --> gives some example combinations of source and destination, and the operations they implement.</p>

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
  <tr valign="bottom">
    <td colspan="3" align="left" valign="bottom">
      <b>Table 2. Selection operations</b></td>
    </tr>
    <tr><td colspan="3"><hr color="green" size="3" /></td></tr>
    <tr valign="top">
      <td width="38%"><b>Source</b></td>
      <td width="37%"><b>Destination</b></td>
      <td width="25%"><b>Operation</b></td>
    </tr>
    <tr><td colspan="3"><hr color="green" size="1" /></td></tr>
    <tr valign="top">
```

```

    <td>All</td>
    <td>All</td>
    <td>Copy whole array</td>
  </tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>All</td>
  <td>All (different shape)</td>
  <td>Copy and reorganize array</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Hyperslab</td>
  <td>All</td>
  <td>Sub-set</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Hyperslab</td>
  <td>Hyperslab (same shape)</td>
  <td>Selection</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Hyperslab</td>
  <td>Hyperslab (different shape)</td>
  <td>Select and rearrange</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>Hyperslab with stride or block</td>

```

All or hyperslab with stride 1	Sub-sample, scatter
--------------------------------	---------------------

<hr/>		
Hyperslab	Points	Scatter

<hr/>		
Points	Hyperslab or all	Gather

<hr/>		
Points	Points (same)	Selection

<hr/>		
Points	Points (different)	Reorder points

<hr/>		
-------	--	--

```
<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="DSelectTransfer">
<div align="right">
<a href="#TOP"><font size=-1>(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>
```

```
<br />
```

```
<a name="DSelectTransfer">
<h3 class="pagebefore">7.5. Dataspace Selection Operations and Data Transfer</h3>
</a>
```

```
<p><em>This section is under construction.</em></p>
<br />
```

```
<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="DRegions">
```

```

<div align="right">
<a href="#TOP"><font size=-1>(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>

```

```

<!-- NEW PAGE -->
<a name="DRegions">
<h3 class="pagebefore">7.6. References to Dataset Regions</h3>
</a>

```

Another use of selections is to store a reference to a region of a dataset. An HDF5 object reference object is a pointer to an object (dataset, group, or committed datatype) in the file. A selection can be used to create a pointer to a set of selected elements of a *dataset*, called a region reference. The selection can be either a point selection or a hyperslab selection. </p>

```

<!-- editingComment

```

```

<span class="editingComment">

```

WORKING TOWARD AN IMPROVED PARAGRAPH:

In addition to the object reference, HDF5 also provides a regions reference.

An HDF5 Region Reference is a pointer to a selection within a dataset.

The selection can be either a point or hyperslab selection.

```

</span>

```

```

-->

```

A more complete description of region references can be found in the chapter ["HDF5 Datatypes"](11_Datatypes.html).

A region reference is an object maintained by the HDF5 Library.

The region reference can be stored in a dataset or attribute, and then read.

The dataset or attribute is defined to have the special datatype,

`H5T_STD_REF_DSETREG`.

To discover the elements and/or read the data, the region reference can be dereferenced. The `H5Rdereference` call returns an identifier for the `dataset`, and then the selected dataspace can be retrieved with `H5Rget_select` call. The selected `dataspace` can be used to read the selected data elements.

7.6.1. Example Uses for Region References

Region references are used to implement stored pointers to data within a dataset. For example, features in a large dataset might be indexed by a table. See the figure below. This table could be stored as an HDF5 dataset with a compound datatype, for example, with a field for the name of the feature and a region reference to point to the feature in the dataset. See the second figure below.

```

<tr valign="top">
  <td align="left" >
    <b>Figure 13. Features indexed by a table
    <!-- formerly Figure 21--></b>
    <hr color="green" size="3"/></td>
</tr>
</table>
<br />
<br />

```

```

<!-- NEW PAGE -->

```

```

<table width="500" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      
    </td></tr>
    <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 14. Storing the table with a
      compound datatype<!-- formerly Figure 22--></b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```


7.6.2. Creating References to Regions

To create a region reference:

- Create or open the dataset that contains the region
- Get the dataspace for the dataset
- Define a selection that specifies the region
- Create a region reference using the dataset and dataspace with selection
- Write the region reference(s) to the desired dataset or attribute

The figure below formerly Figure 23 shows a diagram of a file with three datasets. Dataset D1 and D2 are two dimensional arrays of integers. Dataset R1 is a one dimensional array of references to regions in D1 and D2. The regions can be any valid selection of the dataspace of the target dataset.

NEW PAGE

--

--

--

--

a) 1 D array of region pointers, each pointer refers to a

selection in one Dataset.



--

--

--

```

<tr valign="top">
  <td align="left" >
    <b>Figure 15. A file with three datasets
    <!-- formerly Figure 23--></b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

The example below <!-- formerly Figure 24 --> shows code to create the array of region references.

The references are created in an array of type `hdset_reg_ref_t`.

Each region is defined as a selection on the dataspace of the dataset, and a reference is created using `H5Rcreate()`. The call to `H5Rcreate()` specifies the file, dataset, and the dataspace with selection.

```

<!-- NEW PAGE -->
<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
/* create an array of 4 region references */
hdset_reg_ref_t ref[4];
/*
 * Create a reference to the first hyperslab in the first Dataset.
 */
offset[0] = 1; offset[1] = 1;

```

```
count[0] = 3; count[1] = 2;
status = H5Sselect_hyperslab(space_id, H5S_SELECT_SET, offset, NULL,
    count, NULL);
status = H5Rcreate(&ref[0], file_id, "D1", H5R_DATASET_REGION,
    space_id);
```

```
/*
 * The second reference is to a union of hyperslabs in the first
 * Dataset
 */
```

```
offset[0] = 5; offset[1] = 3;
count[0] = 1; count[1] = 4;
status = H5Sselect_none(space_id);
status = H5Sselect_hyperslab(space_id, H5S_SELECT_SET, offset,
    NULL, count, NULL);
offset[0] = 6; offset[1] = 5;
count[0] = 1; count[1] = 2;
status = H5Sselect_hyperslab(space_id, H5S_SELECT_OR, offset, NULL,
    count, NULL);
status = H5Rcreate(&ref[1], file_id, "D1", H5R_DATASET_REGION,
    space_id);
```

```
/*
 * the fourth reference is to a selection of points in the first
 * Dataset
 */
```

```
status = H5Sselect_none(space_id);
coord[0][0] = 4; coord[0][1] = 4;
coord[1][0] = 2; coord[1][1] = 6;
coord[2][0] = 3; coord[2][1] = 7;
```

```

coord[3][0] = 1; coord[3][1] = 5;
coord[4][0] = 5; coord[4][1] = 8;
status = H5Sselect_elements(space_id, H5S_SELECT_SET,num_points,
                           (const hssize_t **)coord);
status = H5Rcreate(&ref[3], file_id, "D1", H5R_DATASET_REGION,
                  space_id);
/*
 * the third reference is to a hyperslab in the second Dataset
 */
offset[0] = 0; offset[1] = 0;
count[0] = 4; count[1] = 6;
status = H5Sselect_hyperslab(space_id2, H5S_SELECT_SET, offset, NULL,
                             count, NULL);
status = H5Rcreate(&ref[2], file_id, "D2", H5R_DATASET_REGION,
                  space_id2);

```

Example 8. Create an array of region references

Example 8. Create an array of region references

Example 8. Create an array of region references

Example 8. Create an array of region references

Example 8. Create an array of region references

Example 8. Create an array of region references

Example 8. Create an array of region references

Example 8. Create an array of region references

When all the references are created, the array of references is written to the dataset R1. The dataset is declared to have datatype

`H5T_STD_REF_DSETREG`. See the example below.

`<!-- formerly Figure 25-->`

```
<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
Hsize_t dimsr[1];
dimsr[0] = 4;
/*
 * Dataset with references.
 */
spacer_id = H5Screate_simple(1, dimsr, NULL);
dsetr_id = H5Dcreate(file_id, "R1", H5T_STD_REF_DSETREG,
    spacer_id, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

/*
 * Write dataset with the references.
 */
status = H5Dwrite(dsetr_id, H5T_STD_REF_DSETREG, H5S_ALL, H5S_ALL,
    H5P_DEFAULT, ref);</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      <b>Example 9. Write the array of references to a dataset
      <!-- formerly Figure 25--></b>
      <hr color="green" size="3"/></td>
```

</tr>
</table>

<p>When creating region references, the following rules are enforced.</p>

-
- The selection must be a valid selection for the target dataset, just as when transferring data
 - The dataset must exist in the file when the reference is created (<code>H5Rcreate</code>)
 - The target dataset must be in the same file as the stored reference
-

<h4>7.6.3. Reading References to Regions</h4>

<p>To retrieve data from a region reference, the reference must be read from the file, and then the data can be retrieved. The steps are:</p>

-
- Open the dataset or attribute containing the reference objects
 - Read the reference object(s)
 - For each region reference, get the dataset (<code>H5R_dereference</code>) and dataspace (<code>H5Rget_space</code>)
 - Use the dataspace and datatype to discover what space is needed to store the data, allocate the correct storage and create a dataspace and datatype to define the memory data layout
-

The example below <!-- formerly Figure 26 --> shows code to read an array of region references from a dataset, and then read the data from the first selected region. Note that the region reference has information that records the dataset (within the file) and the selection on the `dataspace` of the `dataset`. After dereferencing the regions reference, the `datatype`, number of points, and some aspects of the selection can be discovered. (For a union of hyperslabs, it may not be possible to determine the exact set of hyperslabs that has been combined.) The table below the code example <!-- formerly Table 3 --> shows the inquiry functions.

<!-- NEW PAGE -->

When reading data from a region reference, the following rules are enforced:

- The target `dataset` must be present and accessible in the file
- The selection must be a valid selection for the `dataset`

--	--

--	--

--	--

--	--

```

dsetr_id = H5Dopen (file_id, "R1", H5P_DEFAULT);

```

```
status = H5Dread(dsetr_id, H5T_STD_REF_DSETREG, H5S_ALL, H5S_ALL,
                H5P_DEFAULT, ref_out);

/*
 * Dereference the first reference.
 * 1) get the dataset (H5Rdereference)
 * 2) get the selected dataspace (H5Rget_region)
 */
dsetv_id = H5Rdereference(dsetr_id, H5R_DATASET_REGION,
                          &ref_out[0]);
space_id = H5Rget_region(dsetr_id, H5R_DATASET_REGION, &ref_out[0]);

/*
 * Discover how many points and shape of the data
 */
ndims = H5Sget_simple_extent_ndims(space_id);

H5Sget_simple_extent_dims(space_id, dimsx, NULL);

/*
 * Read and display hyperslab selection from the dataset.
 */
dimsy[0] = H5Sget_select_npoints(space_id);
spacex_id = H5Screate_simple(1, dimsy, NULL);

status = H5Dread(dsetv_id, H5T_NATIVE_INT, H5S_ALL, space_id,
                H5P_DEFAULT, data_out);
printf("Selected hyperslab: ");
for (i = 0; i < 8; i++)
```



```

{
    printf("\n");
    for (j = 0; j < 10; j++)
        printf("%d ", data_out[i][j]);
}
printf("\n");</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 10. Read an array of region references, and then
        read from the first selection
        <!-- formerly Figure 26--></b>
        <hr color="green" size="3" /></td>
    </tr>
</table>
<br />
<br />

<!-- NEW PAGE -->
<table width="600" cellspacing="0" align="center" cellpadding="0">
    <tr valign="bottom">
        <td colspan="2" align="left" valign="bottom">
            <b>Table 3. The inquiry functions</b></td>
        </tr>
        <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
        <tr valign="top">
            <td width="45%"><b>Function</b></td>
            <td width="55%"><b>Information</b></td>
        <tr><td colspan="2"><hr color="green" size="1" /></td></tr>

```

```

<tr valign="top">
  <td><code>H5Sget_select_npoints</code></td>
  <td>The number of elements in the selection (hyperslab
  or point selection).</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5Sget_select_bounds</code></td>
  <td>The bounding box that encloses the selected
  points (hyperslab or point selection).</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5Sget_select_hyper_nblocks</code></td>
  <td>The number of blocks in the selection.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5Sget_select_hyper_blocklist</code></td>
  <td>A list of the blocks in the selection.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5Sget_select_elem_npoints</code></td>
  <td>The number of points in the selection.</td>
</tr>
<tr><td colspan="2"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td><code>H5Sget_select_elem_pointlist</code></td>
  <td>The points.</td>
</tr>

```

```

    <tr><td colspan="2"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<SCRIPT language="JavaScript">
<!--
document.writeln ("
<a name="Programs">
<div align="right">
<a href="#TOP"><font size=-1>(Top)</font></a>
</div>
</a>
");
-->
</SCRIPT>

```

```

<br />
<!-- NEW PAGE -->
<a name="Programs">
<h3 class="pagebefore">7.7. Sample Programs</h3>
</a>

```

This section contains the full programs from which several of the code examples in this chapter were derived.

The `h5dump` output from the program's output file immediately follows each program.

<h4>7.7.1. <code>h5_write.c</code></h4>

<pre>

#include "hdf5.h"

#define H5FILE_NAME "SDS.h5"

#define DATASETNAME "C Matrix"

#define NX 3 /* dataset dimensions */

#define NY 5

#define RANK 2

int

main (void)

{

hid_t file, dataset; /* file and dataset identifiers */

hid_t datatype, dataspace; /* identifiers */

hsize_t dims[2]; /* dataset dimensions */

herr_t status;

int data[NX][NY]; /* data to write */

int i, j;

/*

* Data and output buffer initialization.

*/

for (j = 0; j < NX; j++) {

for (i = 0; i < NY; i++)

data[j][i] = i + 1 + j*NY;

}

/*

```
* 1 2 3 4 5
* 6 7 8 9 10
* 11 12 13 14 15
*/

/*
* Create a new file using H5F_ACC_TRUNC access,
* default file creation properties, and default file
* access properties.
*/
file = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

/*
* Describe the size of the array and create the data space for fixed
* size dataset.
*/
dims[0] = NX;
dims[1] = NY;
dataspace = H5Screate_simple(RANK, dims, NULL);

/*
* Create a new dataset within the file using defined dataspace and
* datatype and default dataset creation properties.
*/
dataset = H5Dcreate(file, DATASETNAME, H5T_NATIVE_INT, dataspace,
                    H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);</pre>
<!-- NEW PAGE -->

```

```
status = H5Dwrite(dataset, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
                  H5P_DEFAULT, data);

/*
 * Close/release resources.
 */
H5Sclose(dataspace);
H5Dclose(dataset);
H5Fclose(file);

return 0;
}
```

SDS.out

```
HDF5 "SDS.h5" {
  GROUP "/" {
    DATASET "C Matrix" {
      DATATYPE H5T_STD_I32BE
      DATASPACE SIMPLE { ( 3, 5 ) / ( 3, 5 ) }
      DATA {
        1, 2, 3, 4, 5,
        6, 7, 8, 9, 10,
        11, 12, 13, 14, 15
      }
    }
  }
}
```

</pre>

<h4>7.7.2. <code>h5_write.f90</code></h4>

<pre>

```
PROGRAM DSETEXAMPLE
```

```
USE HDF5 ! This module contains all necessary modules
```

```
IMPLICIT NONE
```

```
CHARACTER(LEN=7), PARAMETER :: filename = "SDSf.h5" ! File name
```

```
CHARACTER(LEN=14), PARAMETER :: dsetname = "Fortran Matrix" ! Dataset name
```

```
INTEGER, PARAMETER :: NX = 3
```

```
INTEGER, PARAMETER :: NY = 5
```

```
INTEGER(HID_T) :: file_id    ! File identifier
```

```
INTEGER(HID_T) :: dset_id    ! Dataset identifier
```

```
INTEGER(HID_T) :: dspace_id  ! Dataspace identifier
```

```
INTEGER(HSIZE_T), DIMENSION(2) :: dims = (/3,5/) ! Dataset dimensions
```

```
INTEGER    :: rank = 2          ! Dataset rank
```

```
INTEGER    :: data(NX,NY)
```

```
INTEGER    :: error ! Error flag
```

```
INTEGER    :: i, j</pre>
```

<pre>

```
!
```

```
! Initialize data
```

```
!  
do i = 1, NX  
  do j = 1, NY  
    data(i,j) = j + (i-1)*NY  
  enddo  
enddo  
  
!  
! Data  
!  
! 1 2 3 4 5  
! 6 7 8 9 10  
! 11 12 13 14 15  
  
!  
! Initialize FORTRAN interface.  
!  
CALL h5open_f(error)  
  
!  
! Create a new file using default properties.  
!  
CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id, error)  
  
!  
! Create the dataspace.  
!  
CALL h5screate_simple_f(rank, dims, dspace_id, error)  
  
!  
! Create and write dataset using default properties.  
!
```



```
CALL h5dcreate_f(file_id, dsetname, H5T_NATIVE_INTEGER, dspace_id, &
    dset_id, error, H5P_DEFAULT_F, H5P_DEFAULT_F, &
    H5P_DEFAULT_F)
```

```
CALL h5dwrite_f(dset_id, H5T_NATIVE_INTEGER, data, dims, error)
```

```
!
```

```
! End access to the dataset and release resources used by it.
```

```
!
```

```
CALL h5dclose_f(dset_id, error)
```

```
!
```

```
! Terminate access to the data space.
```

```
!
```

```
CALL h5sclose_f(dspace_id, error)
```

```
!
```

```
! Close the file.
```

```
!
```

```
CALL h5fclose_f(file_id, error)
```

```
!
```

```
! Close FORTRAN interface.
```

```
!
```

```
CALL h5close_f(error)
```

```
END PROGRAM DSETEXAMPLE</pre>
```

```
<pre>
```

```
SDSf.out
```

```
-----
```

```

HDF5 "SDSf.h5" {
GROUP "/" {
  DATASET "Fortran Matrix" {
    DATATYPE H5T_STD_I32BE
    DATASPACE SIMPLE { ( 5, 3 ) / ( 5, 3 ) }
    DATA {
      1, 6, 11,
      2, 7, 12,
      3, 8, 13,
      4, 9, 14,
      5, 10, 15
    }
  }
}
}

```

</pre>

<h4>7.7.3. <code>h5_write_tr.f90</code></h4>

<pre>

```

PROGRAM DSETEXAMPLE

```

```

USE HDF5 ! This module contains all necessary modules

```

```

IMPLICIT NONE

```

```

CHARACTER(LEN=10), PARAMETER :: filename = "SDSf_tr.h5" ! File name

```

```

CHARACTER(LEN=24), PARAMETER :: dsetname = "Fortran Transpose Matrix"

```

```

! Dataset name

```

```

INTEGER, PARAMETER :: NX = 3

```

```
INTEGER, PARAMETER :: NY = 5
```

```
INTEGER(HID_T) :: file_id    ! File identifier
```

```
INTEGER(HID_T) :: dset_id    ! Dataset identifier
```

```
INTEGER(HID_T) :: dspace_id  ! Dataspace identifier
```

```
INTEGER(HSIZE_T), DIMENSION(2) :: dims = (/NY, NX/) ! Dataset dimensions
```

```
INTEGER    :: rank = 2          ! Dataset rank
```

```
INTEGER    :: data(NY,NX)
```

```
INTEGER    :: error ! Error flag
```

```
INTEGER    :: i, j
```

```
!
```

```
! Initialize data
```

```
!
```

```
do i = 1, NY
```

```
  do j = 1, NX
```

```
    data(i,j) = i + (j-1)*NY
```

```
  enddo
```

```
enddo
```

```
!
```

```
! Data
```

```
!
```

```
! 1 6 11
```

```
! 2 7 12
```

```
! 3 8 13
```

```
! 4 9 14
```

```
! 5 10 15
```

```
!  
! Initialize FORTRAN interface.  
!  
CALL h5open_f(error)  
  
!  
! Create a new file using default properties.  
!  
CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id, error)  
  
!  
! Create the dataspace.  
!  
CALL h5screate_simple_f(rank, dims, dspace_id, error)  
  
!  
! Create and write dataset using default properties.  
!  
CALL h5dcreate_f(file_id, dsetname, H5T_NATIVE_INTEGER, dspace_id, &  
    dset_id, error, H5P_DEFAULT_F, H5P_DEFAULT_F, &  
    H5P_DEFAULT_F)  
  
CALL h5dwrite_f(dset_id, H5T_NATIVE_INTEGER, data, dims, error)  
  
!  
! End access to the dataset and release resources used by it.  
!  
CALL h5dclose_f(dset_id, error)  
  
!  
! Terminate access to the data space.
```

```
!  
CALL h5sclose_f(dspace_id, error)
```

```
!  
! Close the file.
```

```
!  
CALL h5fclose_f(file_id, error)
```

```
!  
! Close FORTRAN interface.  
!  
CALL h5close_f(error)
```

```
END PROGRAM DSETEXAMPLE</pre>
```

```
<!-- NEW PAGE -->
```

```
<pre>
```

```
SDSf_tr.out
```

```
-----
```

```
HDF5 "SDSf_tr.h5" {  
  GROUP "/" {  
    DATASET "Fortran Transpose Matrix" {  
      DATATYPE H5T_STD_I32LE  
      DATASPACE SIMPLE { ( 3, 5 ) / ( 3, 5 ) }  
      DATA {  
        1, 2, 3, 4, 5,  
        6, 7, 8, 9, 10,  
        11, 12, 13, 14, 15  
      }  
    }  
  }  
}
```

```
}</pre>
```

```
<br /><br />
```

```
</body>
```

```
</html>
```

I

```

<!doctype HTML public "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
<head>
<title>Chapter 8: HDF5 Attributes</title>

<!--(Meta)=====-->

<!--(Links)=====-->

<link href="ed_styles/NewUGelect.css" rel="stylesheet" type="text/css">

<!--( Begin styles definition )=====-->
<!--   Replaced with external stylesheet 'styles_NewUG.css'.   -->
<!--( End styles definition )=====-->

</head>

<body>

<!-- #BeginLibraryItem "/ed_libs/Copyright.lbi" -->
<!--
*****

* Copyright by The HDF Group.                      *
* Copyright by the Board of Trustees of the University of Illinois.      *
* All rights reserved.                               *
*                                                     *

* This file is part of HDF5.  The full HDF5 copyright notice, including   *
* terms governing use, modification, and redistribution, is contained in  *
* the files COPYING and Copyright.html.  COPYING can be found at the root  *
* of the source code distribution tree; Copyright.html can be found at the *

```

```

* root level of an installed copy of the electronic HDF5 document set and *
* is linked from the top-level documents page. It can also be found at *
* http://www.hdfgroup.org/HDF5/doc/Copyright.html. If you do not have *
* access to either file, you may request a copy from help@hdfgroup.org. *
*****

-->
<!-- #EndLibraryItem --><!-- HEADER LEFT "HDF5 User's Guide" -->
<!-- HEADER RIGHT "HDF5 Attributes" -->

<!--( TOC )=====-->
<SCRIPT language="JavaScript">
<!--
document.writeln ('\
<table x-use-null-cells\
        align="right"\
width=240\
cellspacing=0\
class="tocTable">\
  <tr valign="top"> \
    <td class="tocTableHeaderCell" colspan="2"> \
      <span class="TableHead">Chapter Contents</span></td>\
    </tr>\
-->
<!-- Table Version 3 -->\
<!--
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#Intro">1.</a></td>\
    <td class="tocTableContentCell3">\
  <a href="#Intro">Introduction</a></td> \
  </tr>\

```

```

<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#Model">2.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#Model">Programming Model</a></td>\
</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#Functions">3.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#Functions">Attribute (H5A) Function Summaries</a></td> \
</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#Working1">4.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#Working1">Working with Attributes</a><br />\
  Attribute structure<br /> \
  Create, write, read<br /> \
  Access by name or index<br /> \
  Obtain information<br /> \
  Iterate, delete, close \
</td>\
</tr>\
\
<tr valign="top"> \
  <td class="tocTableContentCell"> \
-->
<!-- editingComment -- "tocTableContentCell" and "tocTableContentCell4" \
-->\
<!-- are the table-closing cell class.\

```

```
<td class="tocTableContentCell2"> \
-->\
<!--
  <a href="#SpecIssues">5.</a></td>\
  <td class="tocTableContentCell4">\
  <a href="#SpecIssues">Special Issues</a><br />\
  Large attributes<br />\
  Attribute names<br />\
  No partial I/O\
  </td></tr>\
</table>\
')
-->
</SCRIPT>
<!--(End TOC)=====-->
```

```
<!-- editingComment
  <span class="editingComment">[ [
  ] ]</span>
-->
```

```
<!-- editingComment
-->
```

```
<div align="center">
<a name="TOP">
```

9. HDF5 Attributes

</div>

<h3>8.1. Introduction</h3>

<p>An HDF5

a small metadata object describing the nature and/or

intended usage of a

A primary data object may be a dataset, group, or committed datatype.

</p>

<p>Attributes are assumed to be very small as data objects go, so storing them as standard HDF5 datasets would be quite inefficient.

HDF5 attributes are therefore managed through a special

attributes interface, H5A, which is designed to easily

attach attributes to primary data objects as

small datasets containing metadata information and

to minimize storage requirements.</p>

<p>Consider, as examples of the simplest case, a set of laboratory readings taken under known temperature and pressure conditions of 18.0 degrees celsius and 0.5 atmospheres, respectively.

The temperature and pressure stored as attributes of the

dataset could be described as the following name/value pairs:</p>

<pre>

temp=18.0

pressure=0.5</pre>

<p>While HDF5 attributes are not standard HDF5 datasets,
they have much in common:</p>

- An attribute has a user-defined dataspace and
the included metadata has a user-assigned datatype
- Metadata can be of any valid HDF5 datatype
- Attributes are addressed by name

<p>But there are some very important differences:</p>

- There is no provision for special storage such as
compression or chunking
- There is no partial I/O or sub-setting capability for attribute
data
- Attributes cannot be shared
- Attributes cannot have attributes
- Being small, an attribute is stored in the object header
of the object it describes and is thus attached directly to
that object

<p>The “Special Issues” section below
describes how to handle attributes that are large in size and how to handle
large numbers of attributes.</p>

<!-- NEW PAGE -->

<p>This chapter discusses or lists the following:</p>

The HDF5 attributes programming model

H5A function summaries

Working with HDF5 attributes

The structure of an attribute

Creating, writing, and reading attributes

Accessing attributes by name or index

Obtaining information regarding
an object's attributes

Iterating across an object's attributes

Deleting an attribute

Closing attributes

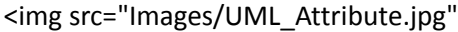
Special issues regarding attributes

<p>In the following discussions, attributes are generally attached to datasets. Attributes attached to other primary data objects, i.e., groups or committed datatypes, are handled in exactly the same manner.</p>

<h3 class="pagebefore">8.2. Programming Model</h3>

<p>The figure below shows the UML model for an HDF5 attribute and its associated dataspace and datatype.</p>

<!--(UML Model Box)=====-->

<div style="text-align: center;"> <hr style="border: 1px solid green;"/>  <p>alt="Image of UML model for an HDF5 attribute and its associated dataspace and datatype"</p> <hr style="border: 1px solid green;"/> </div>
<hr style="border: 1px solid green;"/>
<div> <div> Figure 1. The UML model for an HDF5 attribute </div> <div> <!-- formerly Figure 2:--> </div> <hr style="border: 1px solid green;"/> </div>

<!--(End UML Model)=====-->

<p>Creating an attribute is similar to creating a dataset.

To create an attribute, the application must specify the object to which the attribute is attached, the datatype and dataspace of the attribute data, and the attribute creation property list.

</p>

<!-- NEW PAGE -->

<p>The following steps are required to create and write an HDF5 attribute:</p>

Obtain the object identifier for the attribute's primary data object

Define the characteristics of the attribute and specify the attribute creation property list

Define the datatype

Define the dataspace

Specify the attribute creation property list

Create the attribute

Write the attribute data (optional)

Close the attribute (and datatype, dataspace, and attribute creation property list, if necessary)

Close the primary data object (if appropriate)

<h4>8.2.2. To Open and Read or Write an Existing Attribute</h4>

<p>The following steps are required to open and read/write an existing attribute. Since HDF5 attributes allow no partial I/O, you need specify only the attribute and the attribute's memory datatype to read it:</p>

Obtain the object identifier for the attribute's primary data object

Obtain the attribute's name or index

Text text text.

<p>Now consider the more generalized case, text text text.

<dir><pre>

```
return_var = H5Xfunction (param1, param2, param3)
```

```
&lt;...<em>text text text</em>...&gt;
```

```
return_var = H5Xfunction (param1, param2, param3)
```

```
&lt;...<em>text text text</em>...&gt;
```

```
return_var = H5Xfunction (param1, param2, param3)
```

</pre></dir>

<p>Notes:

Text, text text.

<div class=pagenever>

<h4>2.2 Programming model element 2</h4>

-->

<!--

<p> </p>

<h3 class="pagebefore">3 Using <code>h5dump</code></h3>

<dir>

</dir>

-->

```
<a name="Functions">
```

```
<p>&nbsp;</p><!-- NEW PAGE -->
```

```
<h3 class="pagebefore">8.3. Attribute (H5A) Function Summaries</h3>
```

```
</a>
```

```
<p>Functions that can be used with attributes (H5A functions) and functions  
that can be used with property lists (H5P functions) are listed below.</p>
```

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<td colspan="3" align="left" valign="bottom">
```

```
<b>Function Listing 1. Attribute functions (H5A)
```

```
</b></td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<b>C Function<br />Fortran Function</b>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

```
<b>Purpose</b>
```

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Acreate</code>
```

```
<br />
```

```
<code>h5acreate_f</code></td><td>&nbsp;</td>
```

```
<td>
```

Creates a dataset as an attribute of another group, dataset, or committed datatype.

The C function is a macro: see [./RM/APICompatMacros.html](http://hdf5.org/doc/1.8.12/RM/APICompatMacros.html)

“API Compatibility Macros in HDF5.”

`H5Acreate_by_name`

`h5acreate_by_name_f`

Creates an attribute attached to a specified object.

`H5Aexists`

`h5aexists_f`

Determines whether an attribute with a given name exists on an object.

`H5Aexists_by_name`

`h5aexists_by_name_f`

Determines whether an attribute with a given name exists on an object.

```
</td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Aclose</code>
```

```
<br />
```

```
<code>h5aclose_f</code></td><td>&nbsp;</td>
```

```
<td>
```

Closes the specified attribute.

```
</td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Adelete</code>
```

```
<br />
```

```
<code>h5adelete_f</code></td><td>&nbsp;</td>
```

```
<td>
```

Deletes an attribute.

```
</td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Adelete_by_idx</code>
```

```
<br />
```

```
<code>h5adelete_by_idx_f</code></td><td>&nbsp;</td>
```

```
<td>
```

Deletes an attribute from an object according to index order.

```
</td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

<code><td></code> <code><code>H5Adelete_by_name</code></code> <code>
</code> <code><code>h5adelete_by_name_f</code></td><td>&nbsp;</td></code> <code><td></code> Removes an attribute from a specified location. <code></td></tr></code>	
<hr style="border: 1px solid green;"/>	
<code><td></code> <code><code>H5Aget_create_plist</code></code> <code>
</code> <code><code>h5aget_create_plist_f</code></td><td>&nbsp;</td></code> <code><td></code> Gets an attribute creation property list identifier. <code></td></tr></code>	
<hr style="border: 1px solid green;"/>	
<code><td></code> <code><code>H5Aget_info</code></code> <code>
</code> <code><code>h5aget_info_f</code></td><td>&nbsp;</td></code> <code><td></code> Retrieves attribute information by attribute identifier. <code></td></tr></code>	
<hr style="border: 1px solid green;"/>	
<code><td></code> <code><code>H5Aget_info_by_idx</code></code> <code>
</code> <code><code>h5aget_info_by_idx_f</code></td><td>&nbsp;</td></code>	

<td>

Retrieves attribute information by attribute index position.

</td></tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Aget_info_by_name</code>

<code>h5aget_info_by_name_f</code></td><td> </td>

<td>

Retrieves attribute information by attribute name.

</td></tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Aget_name</code>

<code>h5aget_name_f</code></td><td> </td>

<td>

Gets an attribute name.

</td></tr>

<!-- NEW PAGE -->

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Aget_name_by_idx</code>

<code>h5aget_name_by_idx_f</code></td><td> </td>

<td>

Gets an attribute name by attribute index position.

</td></tr>

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

--	--	--

<code>(none)</code></td><td> </td>

<td>

Calls a user's function for each attribute

attached to a data object.

The C function is a macro: see

“API Compatibility Macros in HDF5.”

</td></tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Aiterate_by_name</code>

<code>(none)</code></td><td> </td>

<td>

Calls user-defined function for each attribute on an object.

</td></tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Aopen</code>

<code>h5aopen_f</code></td><td> </td>

<td>

Opens an attribute for an object specified by object identifier and attribute name.

</td></tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Aopen_by_idx</code>


```
<br />
```

```
<code>h5aopen_by_idx_f</code></td><td>&nbsp;</td>
```

```
<td>
```

Opens an existing attribute that is attached to an object specified by location and name.

```
</td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Aopen_by_name</code>
```

```
<br />
```

```
<code>h5aopen_by_name_f</code></td><td>&nbsp;</td>
```

```
<td>
```

Opens an attribute for an object by object name and attribute name.

```
</td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Aread</code>
```

```
<br />
```

```
<code>h5aread_f</code></td><td>&nbsp;</td>
```

```
<td>
```

Reads an attribute.

```
</td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Arename</code>
```

```
<br />
```

```
<code>h5arename_f</code></td><td>&nbsp;</td>
```

```
<td>
```

Renames an attribute.

```
</td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Arename_by_name</code>
```

```
<br />
```

```
<code>h5arename_by_name_f</code></td><td>&nbsp;</td>
```

```
<td>
```

Renames an attribute.

```
</td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Awrite</code>
```

```
<br />
```

```
<code>H5awrite_f</code></td><td>&nbsp;</td>
```

```
<td>
```

Writes an attribute.

```
</td></tr>
```

```
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
```

```
</table>
```

```
<br />
```

```
<br />
```

```
<!-- NEW PAGE -->
```

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```

<tr valign="bottom">
  <td colspan="3" align="left" valign="bottom">
    <b>Function Listing 2. Attribute creation property list
    functions (H5P) </b></td>
  </tr>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
<tr valign=top>
  <td>
    <b>C Function<br />Fortran Function</b>
  </td><td>&nbsp;</td>
  <td>
    <b>Purpose</b>
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign=top>
  <td>
    <code>H5Pset_char_encoding<br />h5pset_char_encoding_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets the character encoding used to encode a string.
    Use to set ASCII or UTF-8 character encoding for object names.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign=top>
  <td>
    <code>H5Pget_char_encoding<br />h5pget_char_encoding_f</code>
  </td><td>&nbsp;</td>
  <td>
    Retrieves the character encoding used to create a string.
  </td>
</tr>

```

```

</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign=top>
  <td>
    <code>H5Pget_attr_creation_order<br />h5pget_attr_creation_order_f</code>
  </td><td>&nbsp;</td>
  <td>
    Retrieves tracking and indexing settings for attribute creation order.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign=top>
  <td>
    <code>H5Pget_attr_phase_change<br />h5pget_attr_phase_change_f</code>
  </td><td>&nbsp;</td>
  <td>
    Retrieves attribute storage phase change thresholds.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign=top>
  <td>
    <code>H5Pset_attr_creation_order<br />h5pget_attr_creation_order_f</code>
  </td><td>&nbsp;</td>
  <td>
    Sets tracking and indexing of attribute creation order.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign=top>

```

<code><td></code> <code><code>H5Pset_attr_phase_change
h5pset_attr_phase_change_f</code></code> <code></td><td>&nbsp;</td></code> <code><td></code> Sets attribute storage phase change thresholds. <code></td></code> <code></tr></code>	<code><tr><td colspan="3"><hr color="green" size="3" /></td></tr></code>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------

`</table>`

`
`

``
`<p> </p><!-- NEW PAGE -->`
`<h3 class="pagebefore">8.4. Working with Attributes</h3>`
``

`<h4>8.4.1. The Structure of an Attribute</h4>`

`<p>An attribute has two parts: name and value(s)</p>`
`<p>HDF5 attributes are sometimes discussed as name/value pairs`
`in the form <code>name=value</code>.</p>`

`<p>An attribute's name is a null-terminated ASCII or UTF-8`
`character string. Each attribute attached to an object has a unique name.`
`</p>`

`<p>The value portion of the attribute contains`
`one or more data elements of the same datatype.`

8.4.3. Accessing Attributes by Name or Index

Attributes can be accessed by name or index value.

The use of an index value makes it possible to iterate through all of the attributes associated with a given object.

To access an attribute by its name, use the `H5Aopen_by_name` function. `H5Aopen_by_name` returns an attribute identifier that can then be used by any function that must access an attribute such as `H5Aread`. Use the function `H5Aget_name` to determine an attribute's name.

To access an attribute by its index value, use the `H5Aopen_by_idx` function. To determine an attribute index value when it is not already known, use the `H5Oget_info` function. `H5Aopen_by_idx` is generally used in the course of opening several attributes for later access. Use `H5Aiterate` if the intent is to perform the same operation on every attribute attached to an object.

8.4.4. Obtaining Information Regarding an Object's Attributes

In the course of working with HDF5 attributes, one may need to obtain any of several pieces of information:

- An attribute name
 - The dataspace of an attribute
 - The datatype of an attribute
 - The number of attributes attached to an object
-

To obtain an attribute's name,
call `H5Aget_name` with an attribute identifier,
`attr_id`:

```
<pre>  
    <code>ssize_t H5Aget_name (hid_t attr_id, size_t buf_size,  
        char *buf)</code>  
</pre>
```

<p>As with other attribute functions, `attr_id`
identifies the attribute; `buf_size` defines the size of the
buffer; and `buf` is the buffer to which the attribute's name
will be read.
</p>

<p>If the length of the attribute name, and hence the value required for
`buf_size`, is unknown, a first call to
`H5Aget_name` will return that size. If the value of
`buf_size` used in that first call is too small,
the name will simply be truncated in `buf`.
A second `H5Aget_name` call can then be used to retrieve the
name in an appropriately-sized buffer.
</p>

To determine the dataspace or datatype of an attribute, call `H5Aget_space` or `H5Aget_type`, respectively:

```
<pre>
```

```
    hid_t H5Aget_space (hid_t attr_id)
```

```
    hid_t H5Aget_type (hid_t attr_id)
```

```
</pre>
```

`H5Aget_space` returns the dataspace identifier for the attribute `attr_id`.

`H5Aget_type` returns the datatype identifier for the attribute `attr_id`.

To determine the number of attributes attached to an object, use the `H5Oget_info` function. The function signature is below.

```
<pre>
```

```
herr_t H5Oget_info( hid_t object_id, H5O_info_t *object_info )
```

```
</pre>
```

The number of attributes will be returned in the `object_info` buffer. This is generally the preferred first step in determining attribute index values. If the call returns `N`, the

attributes attached to the object `object_id` have index values of `0` through `N-1`.

8.4.5. Iterating across an Object's Attributes

It is sometimes useful to be able to perform the identical operation across all of the attributes attached to an object. At the simplest level, you might just want to open each attribute. At a higher level, you might wish to perform a rather complex operation on each attribute as you iterate across the set.

To iterate an operation across the attributes attached to an object, one must make a series of calls to `H5Aiterate`:

```

    herr_t H5Aiterate (hid_t obj_id,
                      H5_index_t index_type,
                      H5_iter_order_t order,
                      hsize_t *n,
                      H5A_operator2_t op,
                      ...

```

```
void *op_data)
```

</dl>

`H5Aiterate` successively marches across all of the attributes attached to the object specified in `loc_id`, performing the operation(s) specified in `op_func` with the data specified in `op_data` on each attribute.

When `H5Aiterate` is called, `index` contains the index of the attribute to be accessed in this call. When `H5Aiterate` returns, `index` will contain the index of the next attribute. If the returned `index` is the null pointer, then all attributes have been processed, and the iterative process is complete.

`op_func` is a user-defined operation that adheres to the `H5A_operator_t` prototype. This prototype and certain requirements imposed on the operator's behavior are described in the `H5Aiterate` entry in the [HDF5 Reference Manual](http://www.hdfgroup.org/HDF5/ReferenceManual/attributes.html).

`op_data` is also user-defined to meet the requirements of `op_func`.

<h3 class="pagebefore">8.5. Special Issues</h3>

<p>Some special issues for attributes are discussed below.</p>

<h4>Large Numbers of Attributes Stored in Dense Attribute Storage</h4>

<p>The dense attribute storage scheme was added in version 1.8 so that datasets, groups, and committed datatypes that have large numbers of attributes could be processed more quickly.</p>

<p>Attributes start out being stored in an object's header. This is known as compact storage. See the “Datasets” chapter for more information on compact, contiguous, and chunked storage.</p>

<p>As the number of attributes grows, attribute-related performance slows. To improve performance, dense attribute storage can be initiated with the <code>H5Pset_attr_phase_change</code> function. See the <cite>HDF5 Reference Manual</cite> for more information. </p>

<p>When dense attribute storage is enabled, a threshold is defined for the number of attributes kept in compact storage. When the number is exceeded, the library moves all of the attributes into dense storage at another location. The library handles the movement of attributes and the pointers between the locations automatically. If some of the attributes are deleted so that the number falls below the threshold, then the attributes are moved back to compact storage by the library.</p>

The improvements in performance from using dense attribute storage are the result of holding attributes in a heap and indexing the heap with a B-tree.

Note that there are some disadvantages to using dense attribute storage. One is that this is a new feature. Datasets, groups, and committed datatypes that use dense storage cannot be read by applications built with earlier versions of the library. Another disadvantage is that attributes in dense storage cannot be compressed.

Large Attributes Stored in Dense Attribute Storage

We generally consider the maximum size of an attribute to be 64K bytes. The library has two ways of storing attributes larger than 64K bytes: in dense attribute storage or in a separate dataset. Using dense attribute storage is described in this section, and storing in a separate dataset is described in the next section.

To use dense attribute storage to store large attributes, set the number of attributes that will be stored in compact storage to 0 with the `H5Pset_attr_phase_change` function. This will force all attributes to be put into dense attribute storage and will avoid the 64KB size limitation for a single attribute in compact attribute storage.

The example code below illustrates how to create a large attribute that will be kept in dense storage.

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
```

```
<td align="left">
<hr color="green" size="3"/>
<pre>
/*
Test use of dense attribute

*/
#define N 82000000
#include "hdf5.h"
#include <stdio.h>
#include <stdlib.h>

int main(){

hid_t fid, gid, sid, aid, gpid, fpid;
hsize_t dims[] = {N};
double *buf;
int i;
herr_t status;

buf = (double *) malloc(sizeof(double) * N);
for (i=0; i < N; i++) { buf[i] = -100.0; }
fpid = H5Pcreate (H5P_FILE_ACCESS);
status = H5Pset_libver_bounds (fpid, H5F_LIBVER_LATEST, H5F_LIBVER_LATEST);
fid = H5Fcreate("adense.h5", H5F_ACC_TRUNC, H5P_DEFAULT, fpid);
gpid = H5Pcreate (H5P_GROUP_CREATE);
status = H5Pset_attr_phase_change (gpid, 0, 0);

gid = H5Gcreate(fid, "testgrp", H5P_DEFAULT, gpid, H5P_DEFAULT);
sid = H5Screate_simple(1, dims, NULL);
```

```
aid = H5Acreate(gid, "bar", H5T_NATIVE_DOUBLE, sid, H5P_DEFAULT, H5P_DEFAULT);
status = H5Awrite(aid, H5T_NATIVE_DOUBLE, buf);
```

```
/* If you remove these two lines, it doesn't crash */
```

```
status = H5Aclose(aid);
status = H5Pclose (gpId);
status = H5Pclose (fpId);
status = H5Gclose(gid);
status = H5Fclose (fid);
```

```
return 0;
}
```

```
</pre></td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 1. Create a large attribute in dense storage</b>
```

```
<hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

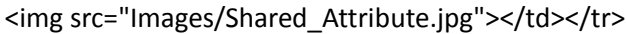
```
<h4>Large Attributes Stored in a Separate Dataset</h4>
```

```
<p>In addition to dense attribute storage (see above), a large attribute
can be stored in a separate dataset. In the figure below, DatasetA holds
```

an attribute that is too large for the object header in Dataset1. By putting a pointer to DatasetA as an attribute in Dataset1, the attribute becomes available to those working with Dataset1.

<!-- formerly Figure 3 -->

This way of handling large attributes can be used in situations where backward compatibility is important and where compression is important. Applications built with versions before 1.8.x can read large attributes stored in separate datasets. Datasets can be compressed while attributes cannot.



<!-- formerly Figure 3: -->

`DatasetA` is an attribute of `Dataset1` that is too large to store in `Dataset1's` header. `DatasetA` is associated with `Dataset1` by means of an object reference pointer attached as an attribute to `Dataset1`. The attribute in `DatasetA` can be shared among multiple datasets by means of additional object reference pointers attached

to additional datasets.

<hr color="green" size="3"/></td>

</tr>

</table>

<h4>Shared Attributes</h4>

<p>Attributes written and managed through the H5A interface cannot be shared. If shared attributes are required, they must be handled in the manner described above for large attributes and illustrated in the figure above<!-- formerly Figure 3 -->.</p>

<h4>Attribute Names</h4>

<p>While any ASCII or UTF-8 character may be used in the name given to an attribute, it is usually wise to avoid the following kinds of characters:</p>

Commonly used separators or delimiters such as slash, backslash, colon, and semi-colon (\, /, :, ;)

Escape characters

Wild cards such as asterisk and question mark (*, ?)

<p>NULL can be used within a name, but HDF5 names are terminated with a NULL: whatever comes after the NULL will be ignored by HDF5.</p>

<p>The use of ASCII or UTF-8 characters is determined by the character encoding property. See <code>H5Pset_char_encoding</code> in the <cite>HDF5 Reference Manual</cite>.</p>

<h4>No Special I/O or Storage</h4>

<p>HDF5 attributes have all the characteristics of HDF5 datasets
except the following:</p>

Attributes are written and read only in full:

there is no provision for partial I/O or sub-setting

No special storage capability is provided for attributes:

there is no compression or chunking, and

attributes are not extendable

<!--

<p> </p>

<h3 class="pagebefore">13.00 Code Examples for Text Text Text</h3>

<dir>

<p class=editingcomment>[[[Comprehensive example set yet to be prepared.]]]</p>

<h4>13.00.1 Example using text text text</h4>

<p>The following example:

</p>

<dir><pre>

code

code

code

</pre></dir>

<h4>13.00.2 Example using text text text</h4>

<p>This example shows how:

</p>

<dir><pre>

code

code

code

</pre></dir>

</dir>

-->

</body>

</html>

I

```

<!doctype HTML public "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
<head>
<title>Chapter 9: HDF5 Error Handling</title>

<!--(Meta)=====-->

<!--(Links)=====-->

<!--( Begin styles definition )=====-->
<link href="ed_styles/NewUglect.css" rel="stylesheet" type="text/css">
<!--( End styles definition )=====-->

</head>

<body>

<!-- #BeginLibraryItem "/ed_libs/Copyright.lbi" -->
<!--
* * * * *
* Copyright by The HDF Group. *
* Copyright by the Board of Trustees of the University of Illinois. *
* All rights reserved. *
*
* This file is part of HDF5. The full HDF5 copyright notice, including *
* terms governing use, modification, and redistribution, is contained in *
* the files COPYING and Copyright.html. COPYING can be found at the root *
* of the source code distribution tree; Copyright.html can be found at the *
* root level of an installed copy of the electronic HDF5 document set and *
* is linked from the top-level documents page. It can also be found at *
* http://www.hdfgroup.org/HDF5/doc/Copyright.html. If you do not have *
```

* access to either file, you may request a copy from help@hdfgroup.org. *

-->

<!-- #EndLibraryItem --><!--(TOC)=====

->

<SCRIPT language="JavaScript">

<!--

document.writeln ('\

<table x-use-null-cells\

align="right"\

width="240"\

cellspacing="0"\

class="tocTable">\

<tr valign="top"> \

<td class="tocTableHeaderCell" colspan="2"> \

Chapter Contents</td>\

</tr>\

-->

<!-- Table Version 3 -->\

<!--

<tr valign="top"> \

<td class="tocTableContentCell2"> \

1.</td>\

<td class="tocTableContentCell3">\

Introduction</td> \

</tr>\

<tr valign="top"> \

<td class="tocTableContentCell2"> \

2.</td>\

<td class="tocTableContentCell3">\

Programming Model</td>\

```

</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#ErrorHandling">3.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#ErrorHandling">Error Handling (H5E) Function Summaries</a></td> \
</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#BasicErrorHandling">4.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#BasicErrorHandling">Basic Error Handling Operations</a>\
</td>\
</tr>\
<tr valign="top">\
  <td class="tocTableContentCell">\
    <a href="#AdvancedErrorHandling">5.</a></td>\
  <td class="tocTableContentCell4">\
    <a href="#AdvancedErrorHandling">Advanced Error Handling Operations</a>\
  </td>\
\
-->
<!-- editingComment -- "tocTableContentCell" and "tocTableContentCell4" \
-->\
<!-- are the table-closing cell class.\
-->\
<!--
\
</table>\
')
-->

```

</SCRIPT>

<!--(End TOC)=====-->

<!-- HEADER LEFT "HDF5 User's Guide" -->

<!-- HEADER RIGHT "HDF5 Error Handling" -->

<div align="center">

10. HDF5 Error Handling

</div>

<h3>9.1. Introduction</h3>

<p>The HDF5 Library provides an error reporting mechanism for both the library itself and for user application programs. It can trace errors through function stack and error information like file name, function name, line number, and error description. </p>

<p>Section 2 of this chapter discusses the HDF5 error handling programming model. </p>

<p>Section 3 presents summaries of HDF5's error handling functions.</p>

<p>Section 4 discusses the basic error concepts such as error stack, error record, and error message and describes the related API functions.

These concepts and functions are sufficient for application programs to trace errors inside the HDF5 Library.</p>

<p>Section 5 talks about the advanced concepts of error class and error stack handle and talks about the related functions. With these concepts and functions, an application library or program using the HDF5 Library can have its own error report blended with HDF5's error report.</p>

Starting with Release 1.8, we have a new set of Error Handling API functions.

For the purpose of backward compatibility with version 1.6 and before, we still keep the old API functions, `H5Epush`, `H5Eprint`, `H5Ewalk`, `H5Eclear`, `H5Eget_auto`, `H5Eset_auto`. These functions do not have the error stack as parameter. The library allows them to operate on the default error stack. Users do not have to change their code to catch up with the new Error API but are encouraged to do so.

The old API is similar to functionality discussed in Section 4. The functionality discussed in Section 5, the ability of allowing applications to add their own error records, is the library new design for the Error API.

[ProgModel](#)

9.2. Programming Model

This section is under construction.

<!-- NEW PAGE -->

[ErrorHandling](#)

9.3. Error Handling (H5E) Function Summaries

Functions that can be used to handle errors (H5E functions) are listed below.

|--|

--

<div> <div>Function Listing 1. Error handling functions (H5E)</div> </div>		
<hr/>		
<div> <div>C Function</div> <div>Fortran Function</div> </div>		
<div> <div>Purpose</div> </div>		
<hr/>		
<div> <div>H5Eauto_is_v2</div> <div>(none)</div> </div>		
<div> <div>Determines the type of error stack.</div> </div>		
<hr/>		
<div> <div>H5Eclear</div> <div>h5eclear_f</div> </div>		
<div> <div>Clears the error stack for the current thread.</div> </div>		
<div> The C function is a macro: see http://www.hdfgroup.org/HDF5/RM/APICompatMacros.html </div>		
<div> "API Compatibility Macros in HDF5." </div>		

```

</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Eclear_stack<br />(none)</code>
  </td><td>&nbsp;</td>
  <td>
    Clears the error stack for the current thread.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Eclose_msg<br />(none)</code>
  </td><td>&nbsp;</td>
  <td>
    Closes an error message identifier.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Eclose_stack<br />(none)</code>
  </td><td>&nbsp;</td>
  <td>
    Closes object handle for error stack.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">

```


<div data-bbox="240 178 763 325"> <code><td></code> <code><code>H5Ecreate_msg
(none)</code></code> <code></td><td>&nbsp;</td></code> </div>	<div data-bbox="240 346 747 441"> <code><td></code> Add major error message to an error class. </div>
<div data-bbox="240 451 308 546"> <code></td></code> <code></tr></code> </div>	
<div data-bbox="215 567 941 598"> <code><tr><td colspan="3"><hr color="green" size="1" /></td></tr></code> </div>	
<div data-bbox="215 619 422 661"> <code><tr valign="top"></code> </div>	
<div data-bbox="240 682 828 829"> <code><td></code> <code><code>H5Eget_auto
h5eget_auto_f</code></code> <code></td><td>&nbsp;</td></code> </div>	<div data-bbox="240 850 917 997"> <code><td></code> Returns the current settings for the automatic error stack traversal function and its data. </div>
<div data-bbox="240 1018 1101 1102"> The C function is a macro: see http://www.hdfgroup.org/HDF5/APICompatMacros.html &ldquo;API Compatibility Macros in HDF5.&rdquo; </div>	
<div data-bbox="240 1123 308 1218"> <code></td></code> <code></tr></code> </div>	
<div data-bbox="215 1239 941 1270"> <code><tr><td colspan="3"><hr color="green" size="1" /></td></tr></code> </div>	
<div data-bbox="215 1291 422 1333"> <code><tr valign="top"></code> </div>	
<div data-bbox="240 1354 812 1501"> <code><td></code> <code><code>H5Eget_class_name
(none)</code></code> <code></td><td>&nbsp;</td></code> </div>	<div data-bbox="240 1522 560 1606"> <code><td></code> Retrieves error class name. </div>
<div data-bbox="240 1627 308 1722"> <code></td></code> <code></tr></code> </div>	
<div data-bbox="215 1743 941 1774"> <code><tr><td colspan="3"><hr color="green" size="1" /></td></tr></code> </div>	
<div data-bbox="215 1795 422 1837"> <code><tr valign="top"></code> </div>	
<div data-bbox="240 1858 300 1890"> <code><td></code> </div>	

```
<code>H5Eget_current_stack<br />(none)</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Registers the current error stack.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Eget_msg<br />(none)</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Retrieves an error message.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Eget_num<br />(none)</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Retrieves the number of error messages in an error stack.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Epop<br />(none)</code>
```

```
</td><td>&nbsp;</td>
```

```
<td>
```

Deletes specified number of error messages from the error stack.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Eprint
h5eprint_f</code>

</td><td> </td>

<td>

Prints the error stack in a default manner.

The C function is a macro: see

“API Compatibility Macros in HDF5.”

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Epush
(none)</code>

</td><td> </td>

<td>

Pushes new error record onto error stack.

The C function is a macro: see

“API Compatibility Macros in HDF5.”

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Eregister_class
(none)</code>

</td><td> </td>

<td>

Registers a client library or application program

to the HDF5 error API.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Eset_auto
h5eset_auto_f</code>

</td><td> </td>

<td>

Turns automatic error printing on or off.

The C function is a macro: see

“API Compatibility Macros in HDF5.”

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Eset_current_stack
(none)</code>

</td><td> </td>

<td>

Replaces the current error stack.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Eunregister_class
(none)</code>

</td><td> </td>

<td>

Removes an error class.

</td>

```

    </tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Ewalk<br />(none)</code>
    </td><td>&nbsp;</td>
    <td>
        Walks the error stack for the current thread,
        calling a specified function.
        The C function is a macro: see <a href=" ../RM/APICompatMacros.html">
        "API Compatibility Macros in HDF5."</a>
    </td>
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
</table>
<br />

```

```

<!-- NEW PAGE -->
<a name="BasicErrorHandling">
<h3>9.4. Basic Error Handling Operations</h3>
</a>

```

```

<h4>9.4.1. Introduction</h4>

```

Let us first try to understand the error stack. An *error stack* is a collection of error records. Error records can be pushed onto or popped off the error stack. By default, when an error occurs deep within the HDF5 Library, an error record is pushed onto an error stack and that function returns a failure indication. Its caller detects the failure, pushes another record onto the stack, and returns a failure indication. This

continues until the API function called by the application returns a failure indication. The next API function being called will reset the error stack. All HDF5 Library error records belong to the same error class (explained in Section 5).

9.4.2. Error Stack and Error Message

In normal circumstances, an error causes the stack to be printed on the standard error stream automatically. This automatic error stack is the library's default stack. For all the functions in this section, whenever an error stack ID is needed as a parameter, `H5E_DEFAULT` can be used to indicate the library's default stack. The first error record of the error stack, number `#000`, is produced by the API function itself and is usually sufficient to indicate to the application what went wrong.

Example: An Error Report

If an application calls `H5Tclose` on a predefined datatype, then the message in the example below is printed on the standard error stream.

This is a simple error that has only one component, the API function; other errors may have many components.

```
<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
HDF5-DIAG: Error detected in HDF5 (1.6.4) thread 0.
```

```
#000: H5T.c line 462 in H5Tclose(): predefined datatype
major: Function argument
minor: Bad value</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 1. An error report</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />
```

In the example above, we can see that an `error record` has a major message and a minor message. A `major message` generally indicates where the error happens. The location can be a dataset or a dataspace, for example. A `minor message` explains further details of the error. An example is `"unable to open file"`. Another specific detail about the error can be found at the end of the first line of each error record. This `error description` is usually added by the library designer to tell what exactly goes wrong. In the example above, the `"predefined datatype"` is an error description.

9.4.3. Print and Clear an Error Stack

Besides the automatic error report, the error stack can also be printed and cleared by the functions `H5Eprint()` and

`H5Eclear_stack()`. If an application wishes to make explicit calls to `H5Eprint()` to print the error stack, the automatic printing should be turned off to prevent error messages from being displayed twice (see `H5Eset_auto()` below). </p>

<!-- NEW PAGE -->

<p>To print an error stack</p>

<p><code>herr_t H5Eprint(hid_t error_stack, FILE * stream)</code></p>

<p>This function prints the error stack specified by `error_stack` on the specified stream, `stream`. If the error stack is empty, a one-line message will be printed. The following is an example of such a message. This message would be generated if the error was in the HDF5 Library. </p>

<p><code>HDF5-DIAG: Error detected in HDF5 Library
version: 1.5.62 thread 0.</code></p>

<p>To clear an error stack</p>

<p><code>herr_t H5Eclear_stack(hid_t error_stack)</code></p>

<p>The `H5Eclear_stack` function shown above clears the error stack specified by `error_stack`. `H5E_DEFAULT` can be passed in to clear the current error stack. The current stack is also cleared whenever an API function is called; there are certain exceptions to this

rule such as `H5Eprint()`.

9.4.4. Mute Error Stack

Sometimes an application calls a function for the sake of its return value, fully expecting the function to fail; sometimes the application wants to call `H5Eprint()` explicitly. In these situations, it would be misleading if an error message were still automatically printed. Using the `H5Eset_auto()` function can control the automatic printing of error messages.

To enable or disable automatic printing of errors

```
herr_t H5Eset_auto(hid_t  
error_stack, H5E_auto_t func,  
void *client_data)
```

The `H5Eset_auto` function can be used to turn on or off the automatic printing of errors for the error stack specified by `error_stack`. When turned on (non-null `func` pointer), any API function which returns an error indication will first call `func`, passing it `client_data` as an argument. When the library is first initialized the auto printing function is set to `H5Eprint()` (cast appropriately) and `client_data` is the standard error stream pointer, `stderr`.

To see the current settings

```
herr_t H5Eget_auto(hid_t  
error_stack, H5E_auto_t * func,  
void **client_data )
```

The function above returns the current settings for the automatic error stack traversal function, `func`, and its data, `client_data`. If either or both of the arguments are null, then the value is not returned.

<!-- NEW PAGE -->

<h4>Example: Error Control</h4>

An application can temporarily turn off error messages while “probing” a function. See the example below.

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
/* Save old error handler */
H5E_auto2_t oldfunc;
void *old_client_data;

H5Eget_auto(error_stack, &old_func, &old_client_data);

/* Turn off error handling */
H5Eset_auto(error_stack, NULL, NULL);

/* Probe. Likely to fail, but that's okay */
status = H5Fopen (.....);

/* Restore previous error handler */
```

```

H5Eset_auto(error_stack, old_func, old_client_data);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 2. Turn off error messages while probing a function</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

Or automatic printing can be disabled altogether and error messages can be explicitly printed.</p>

```

<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
/* Turn off error handling permanently */
H5Eset_auto(error_stack, NULL, NULL);

/* If failure, print error message */
if (H5Fopen (...)&lt;0) {
  H5Eprint(H5E_DEFAULT, stderr);
  exit (1);
}</pre></td>
  </tr>

```

```

<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 3. Disable automatic printing and explicitly print
    error messages</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

<h4>9.4.5. Customized Printing of an Error Stack</h4>

<p>Applications are allowed to define an automatic error traversal function other than the default <code>H5Eprint()</code>. For instance, one can define a function that prints a simple, one-line error message to the standard error stream and then exits. The first example below defines a such a function. The second example below installs the function as the error handler.</p>

```
<!-- NEW PAGE -->
```

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
herr_t
my_hdf5_error_handler(void *unused)
{
    fprintf (stderr, &ldquo;An HDF5 error was detected. Bye.\n&rdquo;);

```

```

        exit (1);
    }</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td align="left">
        <b>Example 4. Defining a function to print a simple error message</b>
        <hr color="green" size="3"/></td></tr>
</table>
<br />
<br />

```

```

<table width="600" cellspacing="0" align="center">
    <tr valign="top">
        <td align="left">
            <hr color="green" size="3"/>
            <pre>
H5Eset_auto(H5E_DEFAULT, my_hdf5_error_handler, NULL);</pre></td>
        </tr>
    <tr><td><hr color="green" size="1" /></td></tr>
    <tr valign="top">
        <td align="left">
            <b>Example 5. The user-defined error handler</b>
            <hr color="green" size="3"/></td>
        </tr>
</table>
<br />

```

9.4.6. Walk through the Error Stack

The `H5Eprint()` function is actually just a wrapper around the more complex `H5Ewalk()` function which traverses an error stack and calls a user-defined function for each member of the stack. The example below shows how `H5Ewalk` is used.

```
<em>herr_t</em> H5Ewalk(<em>hid_t</em>
err_stack, <em>H5E_direction_t</em>
direction, <em>H5E_walk_t</em> func,
<em>void</em> *client_data)
```

The error stack `err_stack` is traversed and `func` is called for each member of the stack. Its arguments are an integer sequence number beginning at zero (regardless of `direction`) and the `client_data` pointer. If `direction` is `H5E_WALK_UPWARD`, then traversal begins at the inner-most function that detected the error and concludes with the API function. Use `H5E_WALK_DOWNWARD` for the opposite order.

9.4.7. Traverse an Error Stack with a Callback Function

An error stack traversal callback function takes three arguments: `n` is a sequence number beginning at zero for each traversal, `eptr` is a pointer to an error stack member, and `client_data` is the same pointer used in the example above passed to `H5Ewalk()`. See the example below.

```
typedef <em>herr_t</em> (*H5E_walk_t)(<em>unsigned</em>
```

n, `H5E_error2_t` *eptr, `void` *client_data)

The `H5E_error2_t` structure is shown below.

<!-- NEW PAGE -->

```
<pre>
```

```
typedef struct {
    hid_tcls_id;
    hid_t    maj_num;
    hid_t    min_num;
    unsigned line;
    const char *func_name;
    const char *file_name;
    const char *desc;
} H5E_error2_t;
</pre>
```

The `maj_num` and `min_num` are major and minor error IDs, `func_name` is the name of the function where the error was detected, `file_name` and `line` locate the error within the HDF5 Library source code, and `desc` points to a description of the error.

Example: Callback Function

The following example shows a user-defined callback function.

```
<table width="600" cellspacing="0" align="center">
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<hr color="green" size="3"/>
```

```
<pre>
```

```
#define MSG_SIZE    64

herr_t
custom_print_cb(unsigned n, const H5E_error2_t *err_desc, void* client_data)
{
    FILE*stream = (FILE *)client_data;
    char      maj[MSG_SIZE];
    char      min[MSG_SIZE];
    char      cls[MSG_SIZE];
    const intindent = 4;

    /* Get descriptions for the major and minor error numbers */
    if(H5Eget_class_name(err_desc->cls_id, cls, MSG_SIZE)<0)
        TEST_ERROR;

    if(H5Eget_msg(err_desc->maj_num, NULL, maj, MSG_SIZE)<0)
        TEST_ERROR;

    if(H5Eget_msg(err_desc->min_num, NULL, min, MSG_SIZE)<0)
        TEST_ERROR;

    fprintf (stream, &ldquo;%*serror #%03d: %s in %s(): line %u\n&rdquo;,
            indent, &ldquo;&rdquo;, n, err_desc->file_name,
            err_desc->func_name, err_desc->line);
    fprintf (stream, &ldquo;%*sclass: %s\n&rdquo;, indent*2, &ldquo;&rdquo;, cls);
    fprintf (stream, &ldquo;%*smajor: %s\n&rdquo;, indent*2, &ldquo;&rdquo;, maj);
    fprintf (stream, &ldquo;%*sminor: %s\n&rdquo;, indent*2, &ldquo;&rdquo;, min);

    return 0;

error:
```



```

    return -1;
}

```

Example 6. A user-defined callback function

Programming Note for C++ Developers Using C Functions

If a C routine that takes a function pointer as an argument is called from within C++ code, the C routine should be returned from normally.

Examples of this kind of routine include callbacks such as `H5Pset_elink_cb` and `H5Pset_type_conv_cb` and functions such as `H5Tconvert` and `H5Ewalk2`.

Exiting the routine in its normal fashion allows the HDF5 C Library to clean up its work properly. In other words, if the C++ application jumps out of the routine back to the C++ "catch" statement, the library is not given the opportunity to close any temporary data structures that were set up when the routine was called. The C++ application should save some state as the routine is started so that any problem that occurs might be diagnosed.

!-- NEW PAGE -->

[AdvancedErrorHandling](#)

9.5. Advanced Error Handling Operations

9.5.1. Introduction

Section 4 discusses the basic error handling operations of the library.

In that section, all the error records on the error stack are from the library itself. In this section, we are going to introduce the operations that allow an application program to push its own error records onto the error stack once it declares an error class of its own through the HDF5 Error API.

Example: An Error Report

An error report shows both the library's error record and the application's error records. See the example below.

--

--

--

--

--

--

--

--

--

--

--

--

--

HDF5-DIAG: Error detected in HDF5 (1.7.5) thread 8192:

```
#002: ../../hdf5/src/H5Dio.c line 420 in H5Dwrite(): not a dataset
```

```
major: Invalid arguments to routine
```

```
minor: Inappropriate type
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 7. An error report</b>
```

```
<hr color="green" size="3" /></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

In the line above error record `#002` in the example above, the starting phrase is `HDF5`. This is the error class name of the HDF5 Library. All of the library's error messages (major and minor) are in this default error class.

The `Error Test` in the beginning of the line above error record `#000` is the name of the application's error class.

The first two error records, `#000` and `#001`, are from application's error class.

By definition, an error class is a group of major and minor error messages for a library (the HDF5 Library or an application library built on top of the HDF5 Library) or an application program. The error class can be registered for a library or program through the HDF5 Error API. Major

and minor messages can be defined in an error class. An application will have object handles for the error class and for major and minor messages for further operation. See the example below.

```
<!-- NEW PAGE -->
```

```
<table width="600" cellspacing="0" align="center">
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<hr color="green" size="3"/>
```

```
<pre>
```

```
#define MSG_SIZE    64
```

```
herr_t
```

```
custom_print_cb(unsigned n, const H5E_error2_t *err_desc, void* client_data)
```

```
{
```

```
    FILE*stream = (FILE *)client_data;
```

```
    char      maj[MSG_SIZE];
```

```
    char      min[MSG_SIZE];
```

```
    char      cls[MSG_SIZE];
```

```
    const intindent = 4;
```

```
    /* Get descriptions for the major and minor error numbers */
```

```
    if(H5Eget_class_name(err_desc->cls_id, cls, MSG_SIZE)&lt;0)
```

```
        TEST_ERROR;
```

```
    if(H5Eget_msg(err_desc->maj_num, NULL, maj, MSG_SIZE)&lt;0)
```

```
        TEST_ERROR;
```

```
    if(H5Eget_msg(err_desc->min_num, NULL, min, MSG_SIZE)&lt;0)
```

```
        TEST_ERROR;
```

```

fprintf (stream, &ldquo;%*serror #03d: %s in %s(): line %u\n&rdquo;;,
        indent, &ldquo;&rdquo;;, n, err_desc-&gt;file_name,
        err_desc-&gt;func_name, err_desc-&gt;line);
fprintf (stream, &ldquo;%*sclass: %s\n&rdquo;;, indent*2, &ldquo;&rdquo;;, cls);
fprintf (stream, &ldquo;%*smajor: %s\n&rdquo;;, indent*2, &ldquo;&rdquo;;, maj);
fprintf (stream, &ldquo;%*sminor: %s\n&rdquo;;, indent*2, &ldquo;&rdquo;;, min);

return 0;

```

```

error:

```

```

return -1;

```

```

}</pre></td>

```

```

</tr>

```

```

<tr><td><hr color="green" size="1" /></td></tr>

```

```

<tr valign="top">

```

```

<td align="left">

```

```

<b>Example 8. Defining an error class</b>

```

```

<hr color="green" size="3"/></td>

```

```

</tr>

```

```

</table>

```

```

<br />

```

9.5.2. More Error API Functions

The Error API has functions that can be used to register or unregister an error class, to create or close error messages, and to query an error class or error message. These functions are illustrated below.

To register an error class

```
hid_t H5Eregister_class(const char*  
cls_name, const char* lib_name,  
const char* version)
```

This function registers an error class with the HDF5 Library so that the application

library or program can report errors together with the HDF5 Library.

To add an error message to an error class

```
hid_t H5Ecreate_msg(hid_t class,  
H5E_type_t msg_type, const char* msg)
```

This function adds an error message to an error class defined by an application library or program. The error message can be either major or minor which is indicated by parameter `msg_type`.

To get the name of an error class

```
ssize_t H5Eget_class_name(hid_t class_id,  
char* name, size_t size)
```

This function retrieves the name of the error class specified by the class `ID`.

To retrieve an error message

```
ssize_t H5Eget_msg(hid_t msg_id,  
H5E_type_t* msg_type, char* msg,  
size_t size)
```

This function retrieves the error message including its length and type.

<p>To close an error message</p>

<p><code>herr_t H5Eclose_msg(hid_t
mesg_id)</code></p>

<p>This function closes an error message.</p>

<p>To remove an error class</p>

<p><code>herr_t H5Eunregister_class(hid_t
class_id)</code></p>

<p>This function removes an error class from the Error API.</p>

<h4>Example: Error Class and its Message</h4>

<p>The example below shows how an application creates an error class and error messages.</p>

<table width="600" cellpadding="0" align="center">

<tr valign="top">

<td align="left">

<hr color="green" size="3"/>

<pre>

/* Create an error class */

class_id = H5Eregister_class(ERR_CLS_NAME, PROG_NAME, PROG_VERS);

/* Retrieve class name */

H5Eget_class_name(class_id, cls_name, cls_size);

```

/* Create a major error message in the class */
maj_id = H5Ecreate_msg(class_id, H5E_MAJOR, &ldquo;... ..&rdquo;);

/* Create a minor error message in the class */
min_id = H5Ecreate_msg(class_id, H5E_MINOR, &ldquo;... ..&rdquo;);</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 9. Create an error class and error messages</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

The example below shows how an application closes error messages and unregisters the error class.</p>

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
H5Eclose_msg(maj_id);
H5Eclose_msg(min_id);
H5Eunregister_class(class_id);</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>

```



```

<tr valign="top">
  <td align="left">
    <b>Example 10. Closing error messages and unregistering the error
    class</b>
    <hr color="green" size="3"/></td>
  </tr>
</table>
<br />

```

<h4>9.5.3. Pushing an Application Error Message onto Error Stack</h4>

<p>An application can push error records onto or pop error records off of the error stack just as the library does internally. An error stack can be registered, and an object handle can be returned to the application so that the application can manipulate a registered error stack.</p>

<p>To register the current stack</p>

<p><code>hid_t H5Eget_current_stack(void)</code></p>

<p>This function registers the current error stack, returns an object handle, and clears the current error stack. An empty error stack will also be assigned an ID.</p>

<p>To replace the current error stack with another</p>

<p><code>herr_t H5Eset_current_stack(hid_t error_stack)</code></p>

This function replaces the current error stack with another error stack specified by `error_stack` and clears the current error stack. The object handle `error_stack` is closed after this function call.

To push a new error record to the error stack

```
herr_t H5Epush(  
    hid_t error_stack,   
    const char* file,   
    const char* func,   
    unsigned line,   
    hid_t cls_id,   
    hid_t major_id,   
    hid_t minor_id,   
    const char* desc,   
    ... )
```

This function pushes a new error record onto the error stack for the current thread.

To delete some error messages

```
herr_t H5Epop(  
    hid_t error_stack,   
    size_t count)
```

This function deletes some error messages from the error stack.

To retrieve the number of error records

```
int H5Eget_num(  
    hid_t error_stack)
```

This function retrieves the number of error records from an error stack.

<!-- NEW PAGE -->

To clear the error stack

```
herr_t H5Eclear_stack(hid_t
error_stack)
```

This function clears the error stack.

To close the object handle for an error stack

```
herr_t H5Eclose_stack(hid_t
error_stack)
```

This function closes the object handle for an error stack and releases its resources.

Example: Working with an Error Stack

The example below shows how an application pushes an error record onto the default error stack.

```


<pre> /* Make call to HDF5 I/O routine */ if((dset_id=H5Dopen(file_id, dset_name, access_plist))&amp;lt;0) {     /* Push client error onto error stack */ </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------


```

```
H5Epush(H5E_DEFAULT,__FILE__,FUNC,__LINE__,cls_id,CLIENT_ERR_MAJ_IO,
        CLIENT_ERR_MINOR_OPEN,&ldquo;H5Dopen failed&rdquo;);
```

```
/* Indicate error occurred in function */
```

```
return(0);
```

```
</pre></td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 11. Pushing an error message to an error stack</b>
```

```
<hr color="green" size="3"/></td>
```

```
</tr>
```

```
</table>
```

```
<br />
```

The example below shows how an application registers the current error stack and creates an object handle to avoid another HDF5 function from clearing the error stack.

```
<!-- NEW PAGE -->
```

```
<table width="600" cellspacing="0" align="center">
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<hr color="green" size="3"/>
```

```
<pre>
```

```
if(H5Dwrite(dset_id, mem_type_id, mem_space_id, file_space_id,
```

```
        dset_xfer_plist_id, buf)&lt;0)
```

```
{
```

```
/* Push client error onto error stack */
H5Epush(H5E_DEFAULT,__FILE__,FUNC,__LINE__,cls_id,CLIENT_ERR_MAJ_IO,
        CLIENT_ERR_MINOR_HDF5,&ldquo;H5Dwrite failed&rdquo;);
```

```
/* Preserve the error stack by assigning an object handle to it */
error_stack = H5Eget_current_stack();
```

```
/* Close dataset */
H5Dclose(dset_id);
```

```
/* Replace the current error stack with the preserved one */
H5Eset_current_stack(error_stack);
```

```
Return(0);
```

```
}</pre></td>
```

```
</tr>
```

```
<tr><td><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td align="left">
```

```
<b>Example 12. Registering the error stack</b>
```

```
<hr color="green" size="3"/></td></tr>
```

```
</table>
```

```
<br />
```

```
<br />
```

```
<br />
```

```
<!-- HEADER RIGHT " " -->
```

```
</body>
```

```
</html>
```

I

```

<!doctype HTML public "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
<head>
<title>Chapter 10: Properties and Property Lists in HDF5</title>

<!--(Meta)=====-->

<!--(Links)=====-->

<!--( Begin styles definition )=====-->
<link href="ed_styles/NewUGelect.css" rel="stylesheet" type="text/css">
<!--( End styles definition )=====-->

</head>

<body>

<!-- CONTENT STARTS AT LINE 162 -->

<!-- #BeginLibraryItem "/ed_libs/styles_UG.lbi" -->
<!--
* * * * *
* Copyright by The HDF Group. *
* Copyright by the Board of Trustees of the University of Illinois. *
* All rights reserved. *
* *
* This file is part of HDF5. The full HDF5 copyright notice, including *
* terms governing use, modification, and redistribution, is contained in *
* the files COPYING and Copyright.html. COPYING can be found at the root *

```

```

* of the source code distribution tree; Copyright.html can be found at the *
* root level of an installed copy of the electronic HDF5 document set and *
* is linked from the top-level documents page. It can also be found at    *
* http://www.hdfgroup.org/HDF5/doc/Copyright.html. If you do not have    *
* access to either file, you may request a copy from help@hdfgroup.org.  *
* * * * *
-->

<!-- #EndLibraryItem --><!-- HEADER LEFT "HDF5 User's Guide" -->
<!-- HEADER RIGHT "Property Lists in HDF5" -->

<!-- ( TOC )=====-->
<!--<SCRIPT language="JavaScript">                                -->
<!--
document.writeln ('\
<table x-use-null-cells\
        align="right"\
width="240"\
cellspacing="0"\
class="tocTable">\
  <tr valign="top"> \
    <td class="tocTableHeaderCell" colspan="2"> \
      <span class="TableHead">Chapter Contents</span></td>\
    </tr>\
-->
<!-- Table Version 3 --><!--\-->
<!--
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#Intro">1.</a></td>\
  <td class="tocTableContentCell3">\
    <a href="#Intro">Introduction</a></td> \

```

```

</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#ProgModel">2.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#ProgModel">Programming Model</a></td>\
</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#H5Dump">3.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#H5Dump">Using <code>h5dump</code></a></td> \
</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#FunctionSumms">4.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#FunctionSumms">File Function Summaries</a></td>\
</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#CrOpen">5.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#CrOpen">Create or Open a File</a></td> \
</tr>\
<tr valign="top"> \
  <td class="tocTableContentCell2"> \
    <a href="#Close">6.</a></td>\
  <td class="tocTableContentCell3">\
<a href="#Close">Close a File</a></td>\
</tr>\

```

```
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#PLists">7.</a></td>\
    <td class="tocTableContentCell3">\
        <a href="#PLists">File Property Lists</a>\
\
<br />\
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
Creating a list\
\
<br />\
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
Creation properties\
\
<br />\
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
Access properties\
\
</td>\
</tr>\
<tr valign="top"> \
    <td class="tocTableContentCell2"> \
        <a href="#Drivers">8.</a></td>\
    <td class="tocTableContentCell3">\
        <a href="#Drivers">Storage Layouts and Drivers</a>\
\
<br />\
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
<span class="smallcaps">SEC2</span>, \
<span class="smallcaps">STDIO</span>,\
<br />
```

```
&nbsp;   &nbsp;  &nbsp;  \
<span class="smallcaps">FAMILY</span>, \
<span class="smallcaps">MULTI</span>,\
<br />\
&nbsp;   &nbsp;  &nbsp;  \
<span class="smallcaps">SPLIT</span>, \
<span class="smallcaps">MPI</span>, \
<span class="smallcaps">CORE</span>,\
<br />\
&nbsp;   &nbsp;  &nbsp;  \
--><!--<span class=code>STREAM</span>, --><!--\
<span class="smallcaps">LOG</span>\
\
</td>\
</tr>\
\
<tr valign="top"> \
    <td class="tocTableContentCell"> \
-->
<!-- editingComment -- "tocTableContentCell" and "tocTableContentCell4" \
--><!--\-->
<!-- are the table-closing cell class.\
    <td class="tocTableContentCell2"> \
--><!--\
        <a href="#Examples">9.</a></td>\
    <td class="tocTableContentCell4">\
        <a href="#Examples">Code Examples</a>\
    </td></tr>\
</table>\
')
-->
```

```
<!-- </SCRIPT> -->
```

```
<!--(End TOC)=====-->
```

```
<!-- editingComment
```

```
-->
```

```
<div align="center">
```

```
<a name="TOP">
```

11. Properties and Property Lists in HDF5

</div>

<!-- ??????? REDACTED FOR REVIEW &/OR PUBLICATION ???????

<i>

<h3>10.0. Background</h3>

<p>

This document will answer the following questions:

What should new users know about property lists?

Is there a key idea?

What are the principles of property lists so to speak?

What would help new users get started?

What might help them use property lists fully and effectively?

</i>

??????? END REDACTION ??????? ??????? -->

10.1. Introduction

HDF5 properties and property lists make it possible to shape or modify an HDF5 file, group, dataset, attribute, committed datatype, or even an I/O stream, in a number of ways.

For example, you can do any of the following:

-

- Customize the storage layout of a file to suit a project or task.

- Create a chunked dataset.

- Apply compression or filters to raw data.

- Use either ASCII or UTF-8 character encodings.

- Create missing groups on the fly.

- Switch between serial and parallel I/O.

- Create consistency within a single file or across an international project.

-

Some properties enable an HDF5 application to take advantage of the capabilities of a specific computing environment while others make a file more compact; some speed the reading or writing of data while others enable more record-keeping at a per-object level. HDF5 offers nearly one hundred specific properties that can be used in literally thousands of combinations to maximize the usability of HDF5-stored data.

The remaining sections in this chapter discuss the following topics:

</p>

What are properties, property lists,
and property list classes?

Property list programming model

Generic property functions

Summary listings of property list functions

Additional resources

</p>

<p>

The discussions and function listings in this chapter focus on
general property operations,
object and link properties,
and related functions.

</p>

<p>

File, group, dataset, datatype, and attribute properties
are discussed in the chapters devoted to those features,

<!--

are discussed in the chapters devoted to those features;
generic property operations are an advanced feature and are
beyond the scope of this guide.

</p>

<p>

Property lists and property list functions that apply only to
specific types of HDF5 objects are generally listed and discussed
in the chapter discussing those objects,

-->

where that information will be most convenient to users.

For example, the [Datasets](10_Datasets.html) chapter

discusses dataset creation property lists and functions,

dataset access property lists and functions, and

dataset transfer property lists and functions.

This chapter does not duplicate those discussions.

</p>

<p>

Generic property operations are an advanced feature and are

beyond the scope of this guide.

</p>

<p>

This chapter assumes an understanding of the following chapters

of this [HDF5 User's Guide](#):

<a href="UG_frame03DataModel.html"

target="_TOP">HDF5 Data Model and File Structure

<a href="UG_frame04ProgModel.html"

target="_TOP">HDF5 Library and Programming Model

</p>

<h3 class="pagebefore">10.2. Property List Classes, Property Lists,

and Properties

</h3>

</p>

<p>

HDF5 property lists and the property list interface

([H5P](../../RM/RM_H5P.html)) provide a mechanism for storing characteristics of objects in an HDF5 file and economically passing them around in an HDF5 application.

In this capacity, property lists significantly reduce the burden of additional function parameters throughout the HDF5 API.

Another advantage of property lists is that features can often be added to HDF5 by adding only property list functions to the API; this is particularly true when all other requirements of the feature can be accomplished internally to the library.

</p>

<p>

For instance, a file creation operation needs to know several things about a file, such as the size of the user-block or the sizes of various file data structures.

Bundling this information as a property list simplifies the interface by reducing the number of parameters to the function

`H5Fcreate`.

</p>

<p>

As illustrated in [the figure above](#PListEcosystem), the HDF5 property environment is a three-level hierarchy:

Property list classes

Property lists

Properties

</p>

<p>

The following subsections discuss property list classes, property lists, and properties in more detail.

</p>

<h4 class="pagebefore">10.2.1. Property List Classes</h4>

A *property list class* defines the roles that property lists of that class can play.

Each class includes all properties that are valid for that class with each property set to its default value.

HDF5 offers a property lists class for each of the following situations.

<p />

<table border="0">

<tr><th> </th>

<th colspan="4" align="left">Table 1:

Property list classes in HDF5</th></tr>

<tr><td> </td>

<td colspan="4"><hr color="green" size="3" /></td></tr>

<tr align="left" valign="bottom">

<th> </th>

<th colspan="2">Property List Class

<hr color="green" size="1" />

</th>

<th><code> </code></th>

<th>For further discussion

```

        <hr color="green" size="1" />
    </th>
</tr>
<tr align="left" valign="top">
    <td width="5%">&nbsp;</td>
    <td>File creation (<small>FCPL</small>)</td>
    <td><code>H5P_FILE_CREATE</code>&nbsp;</td>
    <td>&nbsp;</td>
    <td width="50%" rowspan="2"><font size=-1>
        See various sections of
        &ldquo;<a href="UG_frame08TheFile.html" target="_TOP">The
        HDF5 File</a>&rdquo; chapter.
    </font>
    </td>
</tr>
<tr align="left" valign="top">
    <td>&nbsp;</td>
    <td>File access (<small>FAPL</small>)</td>
    <td><code>H5P_FILE_ACCESS</code>&nbsp;</td>
    <td>&nbsp;</td>
</tr>
<tr align="left" valign="top">
    <td>&nbsp;</td>
    <td>File mount (<small>FMPL</small>)</td>
    <td><code>H5P_FILE_MOUNT</code>&nbsp;</td>
    <td>&nbsp;</td>
    <td><font size=-1>
        Used only as <code>H5P_DEFAULT</code>
        (see <a href="#H5P_FILE_MOUNT">footnote 1</a>).
    </font>
    </td>

```

```

</tr>
<tr><td>&nbsp;</td>
  <td colspan="2"><hr color="green" size="1" /></td>
  <td>&nbsp;</td>
  <td colspan="1"><hr color="green" size="1" /></td></tr>
<tr align="left" valign="top">
  <td>&nbsp;</td>
  <td>Object creation (<small>OCPL</small>)</td>
  <td><code>H5P_OBJECT_CREATE</code>&nbsp;</td>
  <td>&nbsp;</td>
  <td rowspan="2"><font size=-1>
    See table of
    <a href="#ObjectPropFunctions">object property functions</a>
    below.
  </font>
  </td>
</tr>
<tr align="left" valign="top">
  <td>&nbsp;</td>
  <td>Object copy (<small>OCPYPL</small>)</td>
  <td><code>H5P_OBJECT_COPY</code>&nbsp;</td>
  <td>&nbsp;</td>
</tr>
<tr><td>&nbsp;</td>
  <td colspan="2"><hr color="green" size="1" /></td>
  <td>&nbsp;</td>
  <td colspan="1"><hr color="green" size="1" /></td></tr>
<tr align="left" valign="top">
  <td>&nbsp;</td>
  <td>Group creation (<small>GCPL</small>)</td>
  <td><code>H5P_GROUP_CREATE</code>&nbsp;</td>

```

```

<td>&nbsp;</td>
<td rowspan="2"><font size="-1">
  See &ldquo;Programming Model&rdquo; section of
  &ldquo;<a href="UG_frame09Groups.html" target="_TOP">HDF5
  Groups</a>&rdquo; chapter.
</font>
</td>
</tr>
<tr align="left" valign="top">
  <td>&nbsp;</td>
  <td>Group access (<small>GAPL</small>)</td>
  <td><code>H5P_GROUP_ACCESS</code>&nbsp;</td>
  <td>&nbsp;</td>
</tr>
<tr><td>&nbsp;</td>
  <td colspan="2"><hr color="green" size="1" /></td>
  <td>&nbsp;</td>
  <td colspan="1"><hr color="green" size="1" /></td></tr>
<tr align="left" valign="top">
  <td>&nbsp;</td>
  <td>Link creation (<small>LCPL</small>)</td>
  <td><code>H5P_LINK_CREATE</code>&nbsp;</td>
  <td>&nbsp;</td>
  <td rowspan="2"><font size=-1>
    See examples in
    &ldquo;<a href="#ProgModel">Programming Model</a>&rdquo;
    section in this chapter and the table of
    <a href="#LinkCreationTable">link creation property functions</a>
    below.
  </font>
</td>

```

```

</tr>
<tr align="left" valign="top">
  <td>&nbsp;</td>
  <td>Link access (<small>LAPL</small>)</td>
  <td><code>H5P_LINK_ACCESS</code>&nbsp;</td>
  <td>&nbsp;</td>
</tr>
<tr><td>&nbsp;</td>
  <td colspan="2"><hr color="green" size="1" /></td>
  <td>&nbsp;</td>
  <td colspan="1"><hr color="green" size="1" /></td></tr>
<tr align="left" valign="top">
  <td>&nbsp;</td>
  <td>Dataset creation (<small>DCPL</small>)</td>
  <td><code>H5P_DATASET_CREATE</code>&nbsp;</td>
  <td>&nbsp;</td>
  <td rowspan="3"><font size=-1>
    See &ldquo;Programming Model&rdquo; section of
    &ldquo;<a href="UG_frame10Datasets.html" target="_TOP">HDF5
    Datasets</a>&rdquo; chapter.
  </font>
  </td>
</tr>
<tr align="left" valign="top">
  <td>&nbsp;</td>
  <td>Dataset access (<small>DAPL</small>)</td>
  <td><code>H5P_DATASET_ACCESS</code>&nbsp;</td>
  <td>&nbsp;</td>
</tr>
<tr align="left" valign="top">
  <td>&nbsp;</td>

```


Dataset transfer (<small>DXPL</small>)	
<code>H5P_DATASET_XFER</code>	

<hr/>	
<hr/>	

	Datatype creation (<small>TCPL</small>)
<code>H5P_DATATYPE_CREATE</code>	
	See various sections of
	" HDF5 Datatypes " chapter.

	Datatype access (<small>TAPL</small>)
<code>H5P_DATATYPE_ACCESS</code>	

<hr/>	
<hr/>	

--	--

```

<td>&nbsp;</td>
<td>String creation (<small>STRCPL</small>)</td>
<td><code>H5P_STRING_CREATE</code>&nbsp;</td>
<td>&nbsp;</td>
<td><font size=-1>
  See &ldquo;Programming Model&rdquo; sections of
  &ldquo;<a href="UG_frame10Datasets.html" target="_TOP">HDF5
  Datasets</a>&rdquo;
  and
  &ldquo;<a href="UG_frame11Datatypes.html" target="_TOP">HDF5
  Datatypes</a>&rdquo; chapters.
</font>
</td>
</tr>
<tr><td>&nbsp;</td>
  <td colspan="2"><hr color="green" size="1" /></td>
  <td>&nbsp;</td>
  <td colspan="1"><hr color="green" size="1" /></td></tr>
<tr align="left" valign="top">
  <td>&nbsp;</td>
  <td>Attribute&nbsp;creation&nbsp;(<small>ACPL</small>)<code>&nbsp;&nbsp;</code></td>
  <td><code>H5P_ATTRIBUTE_CREATE</code>&nbsp;</td>
  <td>&nbsp;</td>
  <td><font size=-1>
    See &ldquo;Working with Attributes&rdquo;
    (or &ldquo;Attribute Usage&rdquo;) section of
    &ldquo;<a href="UG_frame13Attributes.html" target="_TOP">HDF5
    Attributes</a>&rdquo; chapter.
  </font>
  </td>
</tr>

```

```

<!-- ??????? ??????? -->
<!-- DISCUSSION OF H5P_ROOT will not be included in the chapter. -->
<!-- Yes, it's the root of the property list structure, but there's not -->
<!-- much (maybe nothing?) interesting that the user can do with it. -->
<!-- -->
<!-- Also note that the root of this structure was originally H5P_NO_CLASS; -->
<!-- when that was changed to H5P_ROOT, code was added to set -->
<!-- H5P_NO_CLASS = H5P_ROOT solely to maintain backward compatibility. -->
<!-- -->
<!-- To clean up the public interface, Quincey may move H5P_ROOT out of -->
<!-- H5Ppublic.h. -->
<!-- ??????? ??????? -->
<!--

```

```

<tr align="left" valign="top">
  <td>&nbsp;</td>
  <td>No property list class</td>
  <td><code>H5P_ROOT</code>&nbsp;</td>
  <td>&nbsp;</td>
  <td>Represents the root of the property list class hierarchy.</td>
</tr>
-->

```

```

<tr><td>&nbsp;</td>
  <td colspan="4"><hr color="green" size="1" /></td></tr>
<tr><td>&nbsp;</td>
  <td colspan="4">
    <font size="-1">

```

The abbreviations to the right of each property list class name
in this table are widely used in both HDF5 programmer documentation

and HDF5 source code.

For example,

FCPL is file creation property list,

OCPL is object creation property list,

OCPYPL is object copy property list, and

STRCPL is string creation property list.

These abbreviations may appear in either uppercase or lowercase.

<hr color="green" size="3" />

</td></tr>

</table>

<p>

The “HDF5 property list class inheritance hierarchy”

figure, immediately following, illustrates the

inheritance hierarchy of HDF5’s property list classes.

Properties are defined at the root of the HDF5 property environment

(“Property List Class Root” in the figure below).

Property list classes then inherit properties from that root,

either directly or indirectly through a parent class.

In every case, a property list class inherits only the properties

relevant to its role.

For example, the object creation property list class (OCPL)

inherits all properties that are relevant to the creation of

any object while the group creation property list class (GCPL)

inherits only those properties that are relevant to group creation.

</p>

<p>

<table width="90%" cellspacing="0" align="center" border="0">

```

<tr valign="top">
  <td align="center">
    <hr color="green" size="3"/>
    
    </td>
  </tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Figure 2. HDF5 property list class inheritance hierarchy</b>
    <font size="-1">
      <br />
      Property list classes displayed above in <u>black</u> are
      directly accessible through the programming interface;
      the root of the property environment and the STRCPL and OCPL
      property list classes, in <font color="888888"><u>gray</u></font>
      above, are not user-accessible.
      <br />
      The red <i>empty set</i> symbol (<font color="red">&empty;</font>)
      indicates that the file mount property list class
      (<small>FMPL</small>) is an empty class;
      that is, it has no settable properties
      (see <a href="#H5P_FILE_MOUNT">footnote 1</a>).
      <br />
      Abbreviations used in this figure are defined in the preceding
      table, &ldquo;Property list classes in HDF5&rdquo;.
    </font>&nbsp;   
    <hr color="green" size="3"/>
  </td>

```

</tr>

</table>

<!-- ??????? REDACTED as "certainly out of place" ?????? -->

<!-- ??????? and "probably repetitious". ???????

</p>

<p>

To illustrate the inheritance diagrammed above,

consider the following example:

Before creating an HDF5 dataset, you will need to determine the creation properties the dataset must have.

That is, you must establish the dataset's dataset creation property list (DCPL).

This DCPL will be derived from the dataset creation property list class and will inherit all of the appropriate properties, each set to a default value.

If necessary, you may then change any of those default values in the new DCPL with the appropriate H5P calls;

`H5Pset_layout` and `H5Pset_chunk` calls, for example, would set up dataset chunking.

You can then use the newly-created and modified DCPL to create any dataset that has the same creation property requirements.

</p>

<p>

Code examples appear in later sections of this chapter and in other chapters where properties and property lists are discussed in the context of specific objects.

For example, properties and property lists relevant to datasets are discussed in the "Programming Model" section of the [HDF5 Datasets](UG_frame10Datasets.html) chapter

in this <cite>HDF5 User's Guide</cite>.

??????? END REDACTION ??????? -->

<!-- ??????? REDACTED FOR REVIEW &/OR PUBLICATION ???????
 ??????? AS BEING REDUNDANT IN THE CURRENT DISCUSSION. ???????
 ??????? THEY MAY, HOWEVER, BE USEFUL ELSEWHERE. ???????

</p>

<p>

A dataset creation property list is used when a dataset is created
and may govern features of the dataset or how it is to be created.

For example, if a dataset is to be chunked,

chunking (<code>H5Pset_layout</code>) and

chunk size (<code>H5Pset_chunk</code>)

must be set in the dataset creation property list used to create
the dataset.

</p>

<p>

Creation properties are permanent and <i>immutable</i>;

they cannot be changed during the life of the dataset.

</p>

<p>

A dataset access property list governs access to the dataset.

For example, a user may wish to tune the chunk cache to optimize
the use of the system on which an analysis is to be run.

This optimization might be different on every system.

Chunk cache optimization parameters are set in the
dataset access property list with <code>H5Pset_chache</code>.

</p>

<p>

Access properties are <i>transient</i> and can be changed.

In fact, applications do not generally have any knowledge of access properties used in the past.

??????? ?????? -->

</p>

<h4 class="pagebefore">10.2.2. Property Lists</h4>

A *property list* is a collection of related properties that are used together in specific circumstances.

A new property list created from a property list class inherits the properties of the property list class and each property's default value.

A fresh dataset creation property list, for example, includes all of the HDF5 properties relevant to the creation of a new dataset.

</p>

<p>

Property lists are implemented as containers holding a collection of name/value pairs.

Each pair specifies a property name and a value for the property.

A property list usually contains information for one to many properties.

</p>

<p>

HDF5's default property values are designed to be reasonable for general use cases. Therefore, an application can often use a property list without modification.

On the other hand, adjusting property list settings is a routine action and there are many reasons for an application to do so.

</p>

<p>

A new property list may either be derived from a property list class or copied from an existing property list.

When a property list is created from a property list class, it contains all the properties that are relevant to the class, with each property set to its default value.

A new property list created by copying an existing property list will contain the same properties and property values as the original property list.

In either case, the property values can be changed as needed through the HDF5 API.

</p>

<p>

Property lists can be freely reused to create consistency.

For example, a single set of file, group, and dataset creation property lists might be created at the beginning of a project and used to create hundreds, thousands, even millions, of consistent files, file structures, and datasets over the project's life.

When such consistency is important to a project, this is an economical means of providing it.

<!-- ??????? REDACTED ??????? -->

<!-- ??????? as the statement seems not to be useful. ???????

</p>

<p>

Note that the HDF5 property list functions generate fully populated property lists and that additional properties are rarely added by an application.

If customized properties and property lists are required by an

application or project, they can be created through the use of HDF5's

[generic property](#GenericPLists) functionality.

??????? END REDACTION ???????? -->

[Properties](#)

10.2.3. Properties

A `<i>property</i>` is the basic element of the property list hierarchy.

HDF5 offers nearly one hundred properties controlling things ranging

from file access rights,

to the storage layout of a dataset,

through optimizing the use of a parallel computing environment.

<p>

Further examples include the following:

| | <tr valign="bottom" align="left"> |
 | Purpose | | Examples | | Property List |

```
<tr valign="top">
```

 |

Specify the driver to be used to open a file
A <small>POSIX</small> driver or an MPI IO driver
FAPL

Specify filters to be applied to a dataset
Gzip compression or checksum evaluation
DCPL

Specify whether to record key times associated with an object
Creation time and/or last-modified time
OCPL

Specify the access mode for a file opened via an external link
Read-only or read-write
LAPL

</tr>

</table>

</p>

<p>

Each property is initialized with a default value.

For each property, there are one or more dedicated `H5Pset_*` calls that can be used to change that value.

</p>

<p>

<i>Creation, access, and transfer properties:</i>

Properties fall into one of several major categories:

creation properties, access properties, and transfer properties.

</p>

<p>

Creation properties control *permanent* object characteristics. These characteristics must be established when an object is created, cannot change through the life of the object (they are *immutable*), and the property setting usually has a permanent presence in the file.

</p>

<p>

Examples of creation properties include:

Whether a dataset is stored in a compact, contiguous, or chunked layout

</p>

<p>

The default for this dataset creation property (`H5Pset_layout`) is that a dataset is stored in a contiguous block.

This works well for datasets with a known size limit that will fit easily in system memory.

A chunked layout is important if a dataset is to be compressed, to enable extending the dataset's size, or to enable caching during I/O.

A compact layout is suitable only for very small datasets because the raw data is stored in the object header.

Some datasets, when analyzed later in their lifetime, will be accessed by means of hyperslabs or subsetting; such datasets will often enable better performance if they are stored in a chunked layout so that the entire dataset does not have to be read into memory every time it is accessed.

Many HDF5 datasets are either sparse or very large (or both) and application performance will benefit from compression.

All of these datasets must be stored in a chunked layout.

(Such chunks can also be optimized to facilitate anticipated access patterns.)

Things you can do with chunked but not contiguous datasets:

-- Compression and other filters

-- Caching

-- Extension (very limited, and non-existent behind the curtain, for contiguous;
unlimited for chunked)

-->

Creation of intermediate groups when adding an object
to an HDF5 file

</p>

<p>

This link creation property

(`H5Pset_create_intermediate_group`)

enables an application to add an object in a file without
having to know that the group or group hierarchy containing
that object already exists.

With this property set, HDF5 automatically creates missing groups.

If this property is not set, an application must verify that each
group in the path exists, and create those that do not,
before creating the new object; if any group is missing,
the create operation will fail.

<!-- ??????? REDACTED ??????? -->

<!-- ??????? as not necessarily helpful ???????

<!-- ??????? and potentially confusing. ???????

</p>

<p>

Note that this property is an exception to the general rule.

It is a creation property not because it is immutable
but because it is used only in the creation process.

??????? END REDACTION ??????? -->

Whether an HDF5 file is a single file

or a set of tightly related files that form a virtual HDF5 file

</p>

<p>

Certain file creation properties enable the application to select one of several file layouts.

Examples of the available layouts include

a standard <small>POSIX</small>-compliant layout

(`H5Pset_fapl_sec2`),

a family of files (`H5Pset_fapl_family`),

and a split file layout that separates raw data and metadata into separate files (`H5Pset_fapl_split`).

These and other file layout options are discussed in the

“Storage Layouts and Drivers” section of the

“HDF5

File” chapter of this <cite>HDF5 User’s

Guide</cite>.

To enable error detection when creating a dataset

<p>

In settings where data integrity is vulnerable,

it may be desirable to set checksumming when datasets are created

(`H5Pset_fletcher32`). A subsequent application will

then have a means to verify data integrity when reading the dataset.

</p>

<p>

<i>Access properties</i> control <i>transient</i> object characteristics.

These characteristics may change with the circumstances under which an object is accessed.

</p>

<p>

Examples of access properties include:

The driver used to open a file

<p>

For example, a file might be created with the MPI I/O driver (`H5Pset_fapl_mpio`) during high-speed data acquisition in a parallel computing environment.

The same file might later be analyzed in a serial computing environment with I/O access handled through the serial POSIX driver (`H5Pset_fapl_sec2`).

Optimization settings in specialized environments

<p>

Optimizations differ across computing environments and according to the needs of the task being performed, so are transient by nature.

</p>

<p>

<i>Transfer properties</i> apply only to datasets and control <i>transient</i> aspects of data I/O.

These characteristics may change with the circumstances under which data is accessed.

</p>

<p>

Examples of dataset transfer properties include:

To enable error detection when reading a dataset

<p>

If checksumming has been set on a dataset

(with `H5Pset_fletcher32`,

in the dataset creation property list),

an application reading that dataset can choose whether

check for data integrity (`H5Pset_edc_check`).

Various properties to optimize chunked data I/O on
parallel computing systems

<p>

HDF5 provides several properties for tuning I/O of

chunked datasets in a parallel computing environment

(`H5Pset_dxpl_mpio_chunk_opt`,

`H5Pset_dxpl_mpio_chunk_opt_num`,

`H5Pset_dxpl_mpio_chunk_opt_ratio`, and

`H5Pget_mpio_actual_chunk_opt_mode`).

<p>

Optimal settings differ

due to the characteristics of a computing environment

and due to an application's data access patterns;

even when working with the same file, these settings might

change for every application and every platform.

</p>

<p>

<!-- ??????? REDACTED FOR REVIEW & PUBLICATION ???????

??????? Questions for QK: ???????

??????? Only 2 DAPL calls? (1 get, 1 set) ???????

??????? All others are labelled DXPL. ???????

??????? Yet opaque to me why there are both types. ???????

<i>Dataset transfer properties:</i>

<i>

Similar discussion of DXPLs.

</i>

</p>

<p>

??????? END REDACTION ??????? ??????? -->

<!-- NEW PAGE -->

<h3 class="pagebefore">10.3. Programming Model</h3>

<p>

The programming model for HDF5 property lists is actually quite simple:

Create a property list.

Modify the property list, if required.

Use the property list.

Close the property list.

There are nuances, of course, but that is the basic process.

</p>

<p>

In some cases, you will not have to define property lists at all.

If the default property settings are sufficient for your application, you can tell HDF5 to use the default property list.

</p>

<p>

The following sections

first discuss the use of default property lists,

then each step of the programming model,

and finally a few less frequently used property list operations.

<!-- ??????? REDACTED FOR REVIEW &/OR PUBLICATION ???????

??????? Restart on Introduction ???????

<p>

This section describes the programming model for creating, populating, and using HDF5 property lists.

</p>

<h4>10.3.1. Creating a New Property List</h4>

<p>

The programming model for creating, populating, and using a new HDF5 property list can be summarized as follows:

Create the property list

Populate the property list with the required properties

Use the property list when creating or accessing
the relevant kinds of objects

</p>

<p>

Consider the simple case of creating _ _ _

See the example below.</p>

<i>

<p>

<u>From notes:</u>

How do property lists work?

How are property lists used? Include sample code.

Diagrams can also be included.

How are property lists configured? How are they set up?

How are they changed?

When can a property list be changed and when is it immutable?

Which property lists are recorded in the file and which are transient?

What property lists are available, and what does each do?

See the “Available Property Lists” section below for more information.

What are the most used property lists?

What property lists seem to make the most difference when they are used?

</i>

<h4>10.3.2. Adding Properties to a Property List</h4>

<h4>10.3.3. Using Property Lists</h4>

??????? END REDACTION ?????? ?????? -->

</i>

<h4 class="pagebefore">10.3.1. Using Default Property Lists</h4>

Default property lists can simplify many routine HDF5 tasks because you do not always have to create every property list you use.

</p>

<p>

An application that would be well-served by HDF5’s default property settings can use the default property lists simply by substituting the value <code>H5P_DEFAULT</code>

and the most common access patterns, both of which must be taken into account in setting up the size and shape of chunks.

</p>

<h4 class="pagebefore">10.3.2. Basic Steps of the Programming Model</h4>

The steps of the property list programming model are described in the sub-sections below.

<h5 class="pagebefore">10.3.2.1. Create a Property List</h5>

A new property list can be created either as an instance of a property list class or by copying an existing property list.

Consider the following examples.

A new dataset creation property list is first created

“from scratch” with `H5Pcreate`.

A second dataset creation property list is then created

by copying the first one with `H5Pcopy`.

<dir>

<dl>

<dt><code>dcplA_id = H5Pcreate (H5P_DATASET_CREATE)</code>;

</dt>

<dd>

The new dataset creation property list is created as an instance

of the property list class `H5P_DATASET_CREATE`.

<p>

The new dataset creation property list's identifier is returned in `dcplA_id` and the property list is initialized with default dataset creation property values.

<p>

A list of valid classes appears above in

“[Table 1: Property List Classes](#PListClassesTable) in HDF5”.

</dd>

<dt>`dcplB_id = H5Pcopy (dcplA_id)`;

</dt>

<dd>

A new dataset creation property list, `dcplB_id`, is created as a copy of `dcplA_id` and is initialized with dataset creation property values currently in `dcplA_id`.

</dd>

</dl>

</div>

At this point, `dcplA_id` and `dcplB_id` are identical; they will both contain any modified property values that were changed in `dcplA_id` before `dcplB_id` was created. They may, however, diverge as additional property values are reset in each.

</p>

<p>

While we are creating property lists, let's

create a link creation property list;
we will need this property list when the new dataset
is linked into the file below:

```
<dir>
<dl>
  <dt><code>lcplAB_id = H5Pcreate (H5P_LINK_CREATE)</code>;
  </dt>
</dl>
</dir>
</p>
```


<h5 class="pagebefore">10.3.2.2. Change Property Values</h5>

This section describes how to set property values.

</p>

<p>

Later in this section, the dataset creation property lists

<code>dcplA_id</code> and <code>dcplB_id</code> created in the
section above will be used respectively to create
chunked and contiguous datasets.

To set this up, we must set the layout property in each property list.

The following example sets

<code>dcplA_id</code> for chunked datasets and
<code>dcplB_id</code> for contiguous datasets:

```
<dir>
<dl>
  <dt><code>error = H5Pset_layout (dcplA_id, H5D_CHUNKED)</code>;
  </dt>
```

```
<dt><code>error = H5Pset_layout (dcplB_id, H5D_CONTIGUOUS)</code>;
```

```
</dt>
```

```
</dl>
```

```
</dir>
```

```
<p>
```

Since `dcplA_id` specifies a chunked layout,
we must also set the number of dimensions and the size of the chunks.

The example below specifies that datasets created with

`dcplA_id` will be 3-dimensional and

that the chunk size will be 100 in each dimension:

```
<dir>
```

```
<dl>
```

```
<dt><code>error = H5Pset_chunk (dcplA_id, 3, [100,100,100])</code>;
```

```
</dt>
```

```
</dl>
```

```
</dir>
```

```
</p>
```

```
<p>
```

These datasets will be created with UTF-8 encoded names.

To accomplish that, the following example sets the character encoding

property in the link creation property list to create link names

with UTF-8 encoding:

```
<dir>
```

```
<dl>
```

```
<dt><code>error = H5Pset_char_encoding (lcplAB_id, H5T_CSET_UTF8)</code>;
```

```
</dt>
```

```
</dl>
```

```
</dir>
```

`dcplA_id` can now be used to create chunked datasets

and `dcplB_id` to create contiguous datasets.

And with the use of `lcplAB_id`, they will be created with UTF-8 encoded names.

[UsePropertyList](#)

10.3.2.3. Use the Property List

Once the required property lists have been created, they can be used to control various HDF5 processes.

For illustration, consider dataset creation.

Assume that the datatype `dtypeAB` and the dataspace `dspaceA` and `dspaceB` have been defined

and that the location identifier `locAB_id` specifies the group `AB` in the current HDF5 file.

We have already created the required link creation and dataset creation property lists. For the sake of illustration, we assume that the default dataset access property list meets our application requirements.

The following calls would create the datasets `dsetA` and `dsetB` in the group `AB`.

The raw data in `dsetA` will be contiguous while `dsetB` raw data will be chunked;

both datasets will have UTF-8 encoded link names:

```
<pre>
```

```
dsetA_id = H5Dcreate2( locAB_id, dsetA, dtypeAB, dspaceA_id,
                        lcplAB_id, dcplA_id, H5P_DEFAULT );
dsetB_id = H5Dcreate2( locAB_id, dsetB, dtypeAB, dspaceB_id,
```

```
lcplAB_id, dcplB_id, H5P_DEFAULT );
```

```
</pre>
```

```
<a name="ClosePList">
```

```
<h5 class="pagebefore">10.3.2.4. Close the Property List</h5>
```

```
</a>
```

Generally, creating or opening anything in an HDF5 file results in an HDF5 identifier. These identifiers are of HDF5 type `hid_t` and include things like file identifiers, often expressed as `<i>file_id</i>`; dataset identifiers, `<i>dset_id</i>`; and property list identifiers, `<i>plist_id</i>`.

To reduce the risk of memory leaks, all of these identifiers must be closed once they are no longer needed.

```
</p>
```

```
<p>
```

Property list identifiers are no exception to this rule, and `H5Pclose` is used for this purpose.

The calls immediately following would close the property lists created and used in the examples above.

```
<div>
```

```
<dl>
```

```
<dt><code>error = H5Pclose (dcplA_id)</code>;
```

```
</dt>
```

```
<dt><code>error = H5Pclose (dcplB_id)</code>;
```

```
</dt>
```

```
<dt><code>error = H5Pclose (lcplAB_id)</code>;
```

```
</dt>
```

```
</dl>
```

```
</dir>
```

```
</p>
```

```
<a name="AdditionalPListOps">
```

```
<h4 class="pagebefore">10.3.3. Additional Property List Operations</h4>
```

```
</a>
```

A few property list operations fall outside of the programming model described above. This section describes those operations.

```
<a name="QueryPListClass">
```

```
<h5 class="pagebefore">10.3.3.1. Query the Class of an Existing  
Property List</h5>
```

```
</a>
```

Occasionally an application will have a property list but not know the corresponding property list class. A call such as in the following example will retrieve the unknown class of a known property list:

```
<dir>
```

```
<dl>
```

```
<dt><code><em>PList_Class</em> = H5Pget_class (dcplA_id)</code>;
```

```
</dt>
```

```
</dl>
```

```
</dir>
```

Upon this function's return, `PList_Class` will contain the value `H5P_DATASET_CREATE` indicating that `dcplA_id` is a dataset creation property list.

<h5 class="pagebefore">10.3.3.2. Determine Current Creation Property List

Settings in an Existing Object</h5>

After a file has been created, another application may work on the file without knowing how the creation properties for the file were set up. Retrieving these property values is often unnecessary; HDF5 can read the data and knows how to deal with any properties it encounters.

</p>

<p>

But sometimes an application must do something that requires knowing the creation property settings. HDF5 makes the acquisition of this information fairly straight-forward;

for each property setting call, `H5Pset_*`,

there is a corresponding `H5Pget_*` call

to retrieve the property's current setting.

</p>

<p>

Consider the following examples which illustrate the determination of dataset layout and chunking settings:

<div>

<div>

<div>The application must first identify the creation property list with the appropriate *get creation property list* call.

There is one such call for each kind of object.</div>

</div>

<div><code>H5Dget_create_plist</code> will return a property list

identifier for the creation property list that was used

to create the dataset. Call it `DCPL1_id`.

`H5Pset_layout` sets a dataset's layout to be

compact, contiguous, or chunked.

`H5Pget_layout` called with `DCPL1_id`

will return the dataset's layout, either

`H5D_COMPACT`, `H5D_CONTIGUOUS`,

or `H5D_CHUNKED`.

`H5Pset_chunk` sets the rank of a dataset,

that is the number of dimensions it will have,

and the maximum size of each dimension.

`H5Pget_chunk`, also called with `DCPL1_id`,

will return the rank of the dataset

and the maximum size of each dimension.

If a creation property value has not been explicitly set,

these `H5Pget_` calls will return the property's

default value.

`REDACTED FOR REVIEW &/OR PUBLICATION`

`REMOVE H5Piterate DISCUSSION; GENERIC PROP`

`FUNCTIONS BEYOND SCOPE`

A more comprehensive approach is to perform a

systematic analysis of the properties in a property list.

This approach could be based on `H5Piterate`, which enables an application discover all the properties in a property list (or all the property lists in a property list class).

The application can then assess each property (or property list) that is of interest. Further discussion of this function is beyond the current scope of this document.

?????? END REDACTION ?????? ?????? -->

</p>

<h5 class="pagebefore">10.3.3.3. Determine Access Property Settings</h5>

Access property settings are quite different from creation properties.

Since access property settings are not retained in an HDF5 file or object, there is normally no knowledge of the settings that were used in the past.

On the other hand, since access properties do not affect characteristics of the file or object, this is not normally an issue.

(See [footnote 2](#APL,CPLexceptions) for exceptions.)

</p>

<p>

One circumstance under which an application might need to determine access property settings might be when a file or object is already open but the application does not know the property list settings.

In that case, the application can use the appropriate

get access property list call to retrieve a property list

identifier. For example, if the dataset `dsetA` from the earlier examples is still open, the following call would return an identifier for the dataset access property list in use:

```
<pre>
```

```
dsetA_dacpl_id = H5Dget_access_plist( dsetA_id );
```

```
</pre>
```

The application could then use the returned property list identifier to analyze the property settings.

```
<!-- ??????? REDACTED FOR REVIEW & PUBLICATION          ???????
```

```
<font color="red">
```

```
<i>
```

```
</p>
```

```
<p>
```

Before putting more time into this subsection, under what circumstance might an application need to do this?

If never or exceedingly rare, this can be dropped.

```
</p>
```

```
</i>
```

```
</font>
```

```
??????? END REDACTION ??????? ??????? -->
```

```
<!-- ??????? REDACTED FOR REVIEW &/OR PUBLICATION ???????
```

```
??????? This is a 1.10 feature.          ???????
```

```
<a name="EncodeDecodePList">
```

```
<h5 class="pagebefore">10.3.3.4. Encode and Decode Property Lists</h5>
```


<i>

A discussion of property list encoding and decoding,
<code>H5Pencode</code> and <code>H5Pdecode</code>,
will be added when these functions appear in a public release,
anticipated to be HDF5 Release 1.10.

</p>

</i>

??????? END REDACTION ?????? ?????? -->

<!-- ?????? REDACTED FOR REVIEW &/OR PUBLICATION ??????

<h3 class="pagebefore">10.4. General Property List Operations</h3>

In many ways, property lists are handled much like objects in HDF5.
Property lists can be created and modified,
and information for many creation properties is retained in the file.
On the other hand, while access properties can be created, changed
as needed, and discarded, access property information is not generally
retained in the file or object.

<sup size="-1">1</sup>

</p>

<p>

The following sections provide general instructions for creating and using property lists.

</p>

<h4 class="pagebefore">10.4.1. Set Object and Attribute Name

Character Encoding</h4>

Object and attribute names can be created with either ASCII or UTF-8 character encoding.

The relevant HDF5 property functions are:

<div>

<dl>

<dt><code>H5Pset_char_encoding</code></dt>

<dt><code>H5Pget_char_encoding</code></dt>

</dl>

</div>

</p>

<p>

ASCII and UTF-8 encodings and the use of these functions are discussed in the “Object and Attribute Names” section of “Using UTF-8 Encoding in HDF5 Applications”.

??????? END REDACTION ??????? ??????? -->

<!-- ??????? REDACTED FOR REVIEW &/OR PUBLICATION ???????

??????? Section isn't quite right. ???????

??????? ???????

??????? Is there an actual OCPL? ???????

??????? Is it always a DCPL or GCPL? ???????

??????? and possibly TCPL? ???????

<h3 class="pagebefore">10.4. Object Properties and Property Lists</h3>

The HDF5 *object creation properties* govern certain aspects of object creation and can be used with several types of objects.

Object creation properties include setting the recording of times associated with a file (`H5Pset_obj_track_times`) and setting the tracking of attribute creation order and indexing an object's attributes on that creation order (`H5Pset_attr_creation_order`).

</p>

<p>

Object creation properties are set in an object creation property list, often represented by `<i>ocpl_id</i>`, and can be used in any create or open call that accepts an object creation property list.

</p>

<p>

Object copy properties govern aspects of copying objects and, like object creation properties, can be used with several types of objects.

Object copying properties include setting various aspects of copying

an object (`H5Pset_copy_object`) and managing the duplication of copied committed datatypes (`H5Padd_merge_committed_dtype_path` and `H5Pfree_merge_committed_dtype_paths`) to ensure that committed datatypes used commonly by several data objects before a copy operation remain shared after the copy operation. whether to copy recursively when copying a group.

Object copy properties are set in an object copy property list, often represented by `ocpypl_id`, and can be used in any call that accepts an object copy property list.

??????? END REDACTION ?????? ?????? -->

<h3 class="pagebefore">10.4. Generic Properties Interface and User-defined Properties</h3>

HDF5's generic property interface provides tools for managing the entire property hierarchy and for the creation and management of user-defined property lists and properties. This interface also makes it possible for an application or a driver to create, modify, and manage custom properties, property lists, and property list classes.

A comprehensive list of functions for this interface appears under [Generic Property Operations \(Advanced\)](http://www.hdfgroup.org/hdf/5.0/RM/RM_H5P.html#GenericPropFuncs) in the

“H5P: Property List Interface” section of the
<cite>HDF5 Reference Manual</cite>.

<p>

</p>

Further discussion of HDF5’s generic property interface and user-defined properties and property lists is beyond the scope of this document.

<!-- ??????? REDACTED FOR REVIEW &/OR PUBLICATION ???????
 ?????? BEGIN MORE SERIOUS Generic Properties TEXT ???????
 ?????? But it's beyond scope for this version ???????

<p>

</p>

Property names beginning with ‘<code>H5</code>’ are reserved for HDF5 Library use and should not be used for user-defined properties.

</p>

<p>

HDF5’s generic property functions provide a general foundation for managing the entire property hierarchy.

The generic property functions enable an application or a driver to create, modify, and manage custom properties, property lists, and property list classes.

In fact, the HDF5 Library itself uses private versions of these functions to some extent when initializing the basic properties, property lists, and property list classes described elsewhere.

</p>

<p>

Applications and drivers (the virtual file layer (VFL) and datatype conversion, for example) can use the generic property functions to create properties that control features they add in an HDF5 environment.

The ability of a driver to create properties when installed at run-time enables new features to be easily created and controlled while localizing changes to the code being added or modified.

This facilitates software maintenance and the evolution of both the application or driver's properties and HDF5's properties ecosystem.

</p>

<p>

The generic property functions also enable users to create and set properties which are temporary in nature and do not need to be stored longer than the application is active. This would allow users to set and query application-specific properties during an application's execution without impacting another application or leaving unnecessary "residue" in the file.

</p>

<h4 class="pagebefore">10.541. Using Generic Property List Functions</h4>

Generic property functions can be used to do anything that the more focussed property list functions do. For example, functionality of the function `H5Pset_fapl_mpio` could be completely implemented with generic function calls. Of course, these tasks can be more simply performed with the

existing functions.

</p>

<p>

The value of the generic property functions comes into play when an application requires a property not included with HDF5.

For exmaple,

<i>

< example needed >

</i>

.

</p>

<p>

A new property list class can be derived from an existing class by calling `H5Pcreate_class` with the `parent_class` parameter set to the existing class that the new class is to inherit from.

Properties that differentiate the new class can then be added and set with `H5Pregister` and, if necessary, `H5Pset`.

</p>

<p>

If the new property list class is unlike any existing class and is being created from scratch, the new class is created with `H5Pcreate_class` with the `parent_class` parameter set to `NULL`.

At this point, an empty property list class exists.

The new empty property list class can then be populated with a set of properties using `H5Pregister` and, if necessary, `H5Pset`.

<i>

< Correct? >

</i>

</p>

<p>

Additional new property list classes can be derived from any existing property list class, including a new *empty* property list class, other user-derived property list classes, or a property list class defined by the HDF5 Library.

User-derived property list classes which are derived from the HDF5 Library-defined classes may be passed to HDF5 functions that expect library-defined property lists and the functions will traverse the inherited classes to find the correct class to retrieve information.

</p>

<p>

New properties can be added to a property list class with `H5Pregister`.

The property will become a permanent part of the property list and will be present in new property lists subsequently created with that class.

Registered properties can have default values for each new property list created for that class.

</p>

<p>

New properties can be added to a property list with `H5Pinsert`.

Such a property can be temporary and will no longer exist once the application quits.

Properties inserted into a property list will not affect property lists created with the parent property list class.

<i>

< Can inserted properties be temporary such that they will no longer exist once the application quits?

This was proposed in the design, but was it implemented that way? >

</i>

</p>

<p>

Property names beginning with '`H5`' are reserved for HDF5 Library use and should not be used by third-party applications or libraries.

</p>

<p>

The names and sizes of property values for each property are local to each property list; changing them in a property list class does not affect existing property lists.

</p>

<p>

A comprehensive list of generic property functions appears below in the [generic property functions](#GenericPropFunctions) table.

<i>

</p>

<p>

This section will continue with discussions of
selected generic property functions to give the user
a better sense of how they can be used.

</i>

??????? END "beyond scope" Generic Properties discussion ?????? ??????

??????? END REDACTION ??????? ??????? -->

</p>

<!-- ??????? REDACTED FOR REVIEW &/OR PUBLICATION ???????

??????? MASK UNUSED TEMPLATES ???????

<h3 class="pagebefore">10.6. Code Snippets</h3>

<i>

This section simply provides templates; none of the text is apropos.

</i>

<p>

Consider the the short example below and the longer one immediately following.

</p>

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
file_id = H5Fcreate ("SampleFile.h5", H5F_ACC_EXCL,
  H5P_DEFAULT, H5P_DEFAULT)</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      <b>Example 1. Creating an HDF5 file using property list defaults</b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />
```

<p>

And the more complex case.

<p>

In this more complex case,

we define file creation and access property lists (though we do not assign any properties), specify that `H5Fcreate` should fail if `SampleFile.h5` already exists, and create a

new file named `SampleFile.h5`. The example does not specify a driver, so the default driver, `H5FD_SEC2`, will be used.

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
fcplist_id = H5Pcreate (H5P_FILE_CREATE)
  &lt;...<em>set desired file creation properties</em>...&gt;
faplist_id = H5Pcreate (H5P_FILE_ACCESS)
  &lt;...<em>set desired file access properties</em>...&gt;
file_id = H5Fcreate ("SampleFile.h5", H5F_ACC_EXCL, fcplist_id, faplist_id)</pre></td>
    </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left">
      <b>Example 2. Creating an HDF5 file using property lists</b>
      <hr color="green" size="3"/>
    </tr>
  <tr valign="top" align="left">
    <td>
      <font size="-1">
Notes:
<br>
A root group is automatically created in a file when the file
is first created.
<br>
```

File property lists, once defined, can be reused when another file is created within the same application.

</td>

</tr>

</table>

<dir>

<p>Notes:

A root group is automatically created in a file when the file is first created.

File property lists, once defined, can be reused when another file is created within the same application.

</p>

</dir>

??????? END UNUSED TEMPLATES MASK ????????

??????? END REDACTION ???????? -->

<!-- NEW PAGE -->

<h3 class="pagebefore">10.5. Property List Function Summaries</h3>

<p>

General property functions,

generic property functions and macros,

property functions that are used with multiple types of objects,

and object and link property functions


```
<code>H5Pcreate</code>
```

```
<br />
```

```
<code>h5pcreate_f<code>
```

```
</td>
```

```
<td>&nbsp;</td>
```

```
<td>
```

Creates a new property list as an instance of a specified parent property list class.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5copy</code>
```

```
<br />
```

```
<code>h5copy_f</code>
```

```
</td>
```

```
<td>&nbsp;</td>
```

```
<td>
```

Creates a new property list by copying the specified existing property list.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pget_class</code>
```

```
<br />
```

```
<code>h5pget_class_f</code>
```

```
</td>
```

```
<td>&nbsp;</td>
```


<td>

Retrieves the parent property list class of the specified property list.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pclose</code>

<code>h5pclose_f</code>

</td>

<td> </td>

<td>

Closes the specified property list.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

</table>

</p>

<p>

Object property functions can be used with several kinds of objects.

</p>

<p>

```
<table width="600" cellspacing="0" align="center" cellpadding="0">
```

```
<tr valign="bottom">
```

```
<th colspan="3" align="left" valign="bottom">
```

```
Function Listing 2. Object property functions (H5P)
```

```
</th>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="3" /></td></tr>
```

```
<tr valign="top">
```

```
<th align="left">
```

```
C&nbsp;Function
```

```
<br />
```

```
Fortran&nbsp;Subroutine&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</b>
```

```
</th>
```

```
<th>&nbsp;</th>
```

```
<th align="left">
```

```
Purpose</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr><th colspan="3" align="left">
```

```
Object Creation Properties
```

```
</th></tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pget_attr_creation_order</code>
```

```
<br />
```

```
<code>h5pget_attr_creation_order_f</code>
```

```
</td>
```

```
<td>&nbsp;</td>
```

```
<td>
```

```
Retrieves tracking and indexing settings for attribute creation order.
```

```

</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pget_attr_phase_change</code>
    <br />
    <code>h5pget_attr_phase_change_f</code>
  </td>
  <td>&nbsp;</td>
  <td>
    Retrieves attribute storage phase change thresholds.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pget_obj_track_times</code>
    <br />
    <code>h5pget_obj_track_times_f</code>
  </td>
  <td>&nbsp;</td>
  <td>
    Determines whether times associated with an object are being recorded.
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset_attr_creation_order</code>
    <br />

```

```

<code>h5pset_attr_creation_order_f</code>
</td>
<td>&nbsp;</td>
<td>
Sets tracking and indexing of attribute creation order.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5Pset_attr_phase_change</code>
<br />
<code>h5pset_attr_phase_change_f</code>
</td>
<td>&nbsp;</td>
<td>
Sets attribute storage phase change thresholds.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5Pset_obj_track_times</code>
<br />
<code>h5pset_obj_track_times_f</code>
</td>
<td>&nbsp;</td>
<td>
Sets the recording of times associated with an object.
</td>
</tr>

```

```

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr><th colspan="3" align="left">
    Object Copy Properties
</th></tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Padd_merge_committed_dtype_path</code>
        <br />
        <code>(none)</code>
    </td>
    <td>&nbsp;</td>
    <td>
        Adds a path to the list of paths that will be searched in
        the destination file for a matching committed datatype.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
    <td>
        <code>H5Pfree_merge_committed_dtype_paths&nbsp;&nbsp;&nbsp;</code>
        <br />
        <code>(none)</code>
    </td>
    <td>&nbsp;</td>
    <td>
        Clears the list of paths stored in an object copy property list.
    </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">

```

```

<td>
<code>H5Pget_copy_object</code>
<br />
<code>h5pget_copy_object_f</code>
</td>
<td>&nbsp;</td>
<td>
Retrieves the properties to be used when an object is copied.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5Pget_mcdt_search_cb</code>
<br />
<code>(none)</code>
</td>
<td>&nbsp;</td>
<td>
Retrieves the callback function from the
specified object copy property list.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5Pset_copy_object</code>
<br />
<code>h5pset_copy_object_f</code>
</td>
<td>&nbsp;</td>

```

|

Sets the properties to be used when an object is copied.

</td>

| --- | | |

```
<tr valign="top">
```

 |

```
<code>H5Pset_mcdt_search_cb</code>
```



```
<code>(none)</code>
```

</td>

 | |

Sets the callback function that H5Ocopy will invoke

before searching the entire destination file

for a matching committed datatype.

</td>

| --- | | |

<p>

[LinkCreationTable](#)

The following table lists link creation properties.

Since the creation of a link is almost always a step in the creation of an object, these properties may also be set in group creation property lists,

dataset creation property lists,

datatype creation property lists,


```

    </tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pget_char_encoding</code>
    <br />
    <code>h5pget_char_encoding_f&nbsp;&nbsp;&nbsp;</code>
  </td>
  <td>&nbsp;</td>
  <td>
    Queries the character encoding used to encode link or attribute names.
    <br>
    <font size="-1"><i>
      Any link, object, dataset, datatype, group, or attribute
      creation property list
    </i></font>
  </td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td>
    <code>H5Pset_char_encoding</code>
    <br />
    <code>h5pset_char_encoding_f&nbsp;&nbsp;&nbsp;</code>
  </td>
  <td>&nbsp;</td>
  <td>
    Sets the character encoding used to encode link and attribute names.
    <br>
    <font size="-1"><i>
      Any link, object, dataset, datatype, group, or attribute

```

creation property list

</i>

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pget_create_intermediate_group</code>

<code>h5pget_create_intermediate_group_f </code>

</td>

<td> </td>

<td>

Queries setting for creation of intermediate groups.

<i>

Link creation property list, which in turn can be used in the

create call for any dataset, datatype, or group

</i>

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pset_create_intermediate_group</code>

<code>h5pset_create_intermediate_group_f </code>

</td>

<td> </td>

<td>

Specifies whether to create intermediate groups

when they do not already exist.

<i>

Link creation property list, which in turn can be used in the

create call for any dataset, datatype, or group

</i>

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

</table>

</p>

<p>

<!-- ??????? GENERIC PROPERTY OPERATIONS TABLE REMOVED FROM UG ?????? -->

<!-- ??????? Property Lists CHAPTER (UG/14_PropertyLists.html) ?????? -->

<!-- ??????? AND CORRESPONDING TEXT REDUCED IN FAVOR OF ?????? -->

<!-- ??????? EVENTUAL (AND AS YET UNSCHEDULED) DOCUMENT ?????? -->

<!-- ??????? SEPARATELY DOCUMENTING GENERIC PROPERTY OPS. ?????? -->

<!-- ??????? THIS WILL PROBABLY BE AN "ADVANCED TOPIC". ?????? -->

<!-- ??????? 9 July 2014 ?????? -->

<!-- ??????? SEE hdf5doc/trunk/sandbox/GenericProperties/. ??????

Generic property functions allow an application to create

properties, property lists, and property list classes

beyond those provided by HDF5.

Beyond this function listing and the

generic and user-defined properties

section above, discussions of HDF5's generic property interface

and user-defined properties and property lists are beyond


```
<td>&nbsp;</td>
```

```
<td>
```

Closes an existing property list class.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pcopy_prop</code>
```

```
<br />
```

```
<code>h5pcopy_prop_f</code>
```

```
</td>
```

```
<td>&nbsp;</td>
```

```
<td>
```

Copies a property from one property list or property list class to another.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pcreate_class</code>
```

```
<br />
```

```
<code>h5pcreate_class_f</code>
```

```
</td>
```

```
<td>&nbsp;</td>
```

```
<td>
```

Creates a new property list class.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
  <td>
    <code>H5Pequal</code>
    <br />
    <code>h5pequal_f</code>
  </td>
  <td>&nbsp;</td>
  <td>
    Compares two property lists or property list classes for equality.
  </td>
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
  <td>
    <code>H5Pexist</code>
    <br />
    <code>h5pexist_f</code>
  </td>
  <td>&nbsp;</td>
  <td>
    Queries whether a property name exists in a property list
    or property list class.
  </td>
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
  <td>
    <code>H5Pget</code>
    <br />
    <code>h5pget_f</code>
  </td>
```

```
<td>&nbsp;</td>
```

```
<td>
```

Queries the value of a property.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pget_class_name</code>
```

```
<br />
```

```
<code>h5pget_class_name_f</code>
```

```
</td>
```

```
<td>&nbsp;</td>
```

```
<td>
```

Retrieves the name of a property list class.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

```
<td>
```

```
<code>H5Pget_class_parent</code>
```

```
<br />
```

```
<code>h5pget_class_parent_f&nbsp;&nbsp;&nbsp;</code>
```

```
</td>
```

```
<td>&nbsp;</td>
```

```
<td>
```

Retrieves the parent class of a property list class.

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
```

```
<tr valign="top">
```

| | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| <code><td></code>
<code><code>H5Pget_nprops</code></code>
<code>
</code>
<code><code>h5pget_nprops_f</code></code>
<code></td></code> | <code><td>&nbsp;</td></code> |
| <code><td></code>
Queries the number of properties in a property list or
property list class.
<code></td></code> | |
| <code></tr></code> | |
| <code><tr><td colspan="3"><hr color="green" size="1" /></td></tr></code> | |
| <code><tr valign="top"></code> | |
| <code><td></code>
<code><code>H5Pget_size</code></code>
<code>
</code>
<code><code>h5pget_size_f</code></code>
<code></td></code> | <code><td>&nbsp;</td></code> |
| <code><td></code>
Queries the size of a property value in bytes.
<code></td></code> | |
| <code></tr></code> | |
| <code><tr><td colspan="3"><hr color="green" size="1" /></td></tr></code> | |
| <code><tr valign="top"></code> | |
| <code><td></code>
<code><code>H5Pinsert</code></code>
<code>
</code>
<code><code>h5pinsert_f</code></code>
<code></td></code> | <code><td>&nbsp;</td></code> |

<td>

Registers a temporary property with a property list.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Pisa_class</code>

<code>h5pisa_class_f</code>

</td>

<td> </td>

<td>

Determines whether a property list is a member of a class.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Piterate</code>

<code>h5piterate_f</code>

</td>

<td> </td>

<td>

Iterates over the properties in a property list or
property list class.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

```

<td>
<code>H5Pregister</code>
<br />
<code>h5pregister_f</code>
</td>
<td>&nbsp;</td>
<td>
Registers a permanent property with a property list class.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5Premove</code>
<br />
<code>h5premove_f</code>
</td>
<td>&nbsp;</td>
<td>
Removes a property from a property list.
</td>
</tr>
<tr><td colspan="3"><hr color="green" size="1" /></td></tr>
<tr valign="top">
<td>
<code>H5Pset</code>
<br />
<code>h5pset_f</code>
</td>
<td>&nbsp;</td>
<td>

```

Sets a property list value.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="1" /></td></tr>

<tr valign="top">

<td>

<code>H5Punregister</code>

<code>h5punregister_f</code>

</td>

<td> </td>

<td>

Removes a property from a property list class.

</td>

</tr>

<tr><td colspan="3"><hr color="green" size="3" /></td></tr>

</table>

</p>

-->

<!-- ??????? END OF REMOVED GENERIC PROPERTY OPERATIONS TABLE. ??????? -->

<h3 class="pagebefore">10.6. Additional Property List Resources</h3>

Property lists are ubiquitous in an HDF5 environment and are therefore discussed in many places in HDF5 documentation.

The following sections and listings in the

<cite>HDF5 User's Guide</cite>

are of particular interest:

<dir>

<dl>

<dt>In the “HDF5

Data Model and File Structure” chapter,

see the “Property List” section.

<dt>In the “HDF5

File” chapter, see the following sections and listings:

<dd>File Creation and File Access Properties

<dd>File Property Lists and its sub-sections

<dd>Example with the File Creation Property List

<dd>Example with File Access Property List

<dd>File creation property list functions

<dd>File access property list functions

<dd>File driver functions

<dt>In the “HDF5

Attributes” chapter, see the “Attribute creation

property list functions” listing.

<dt>In the “HDF5

Groups” chapter, see the “Group creation

property list functions” listing.

<dt>Property lists are discussed throughout the

“<a href="UG_frame10Datasets.html"

target="_top">HDF5 Datasets” chapter.

</dl>

</dir>

</p>

<p>

All property list functions are described in the

“H5P: Property List

Interface” section of the

<cite>HDF5 Reference Manual</cite>.

The function index at the top of the page provides a categorized listing

grouped by property list class. Those classes are listed below:

File creation properties

File access properties

Group creation properties

Dataset creation properties

Dataset access properties

Dataset transfer properties

Link creation properties

Link access properties

Object creation properties

Object copy properties

String creation property

Additional categories not related to the class structure

are as follows:

General property list operations

Generic property list functions

The general property functions can be used with any property list;

the generic property functions constitute an advanced feature.

</p>

<p>

The in-memory file image feature of HDF5 uses property lists

in a manner that differs substantially from their use elsewhere in HDF5.

Those who plan to use in-memory file images must study “<a href=

"../Advanced/FileImageOperations/HDF5FileImageOperations.pdf">File

Image Operations” (PDF) in the

<cite>Advanced Topics in HDF5</cite>

collection.

</p>

<hr>

<h3 class="pagebefore">Footnotes</h3>

<dl>

<dt>1. File mount properties

</dt>

<dd>While the file mount property list class

<code>H5P_FILE_MOUNT</code> is a valid HDF5 property list class,

no file mount properties are defined by the HDF5 Library.

References to a file mount property list should always be

expressed as <code>H5P_DEFAULT</code>, meaning the

default file mount property list.

</dl>

</p>

<p>

<dl>

<dt>2. Access and creation property exceptions

</dt>

<dd>There are a small number of exceptions to the rule that
creation properties are always retained in a file or object and
access properties are never retained.

</p>

<p>

The following properties are file access properties

but they are not transient;

they have permanent and different effects on a file.

They could be validly classified as file creation properties

as they must be set at creation time to properly create the file.

But they are access properties because they must also be set

when a file is reopened to properly access the file.

<table>

<tr align="left" valign="top">

<th><code> </code></th>

<th>Property</th>

<th><code> </code></th>

<th>Related function</th>

</tr>

<tr align="left" valign="top">

<td><code> </code></td>

<td>Family file driver</td>

```

<td><code>&nbsp;</code></td>
<td><code>H5Pset_fapl_family</code></td>
</tr>
<tr align="left" valign="top">
  <td><code>&nbsp;&nbsp;&nbsp;&nbsp;</code></td>
  <td>Split file driver</td>
  <td><code>&nbsp;</code></td>
  <td><code>H5Pset_fapl_split</code></td>
</tr>
<tr align="left" valign="top">
  <td><code>&nbsp;&nbsp;&nbsp;&nbsp;</code></td>
  <td>Core file driver</td>
  <td><code>&nbsp;</code></td>
</p>
<p>
</td>
<td><code>H5Pset_fapl_core</code></td>
</tr>

<tr align="left" valign="top">
  <td colspan="4">
    The following is a link creation property, but
    it is not relevant after an object has been created
    and is not retained in the file or object.
  </td>
</tr>
<tr align="left" valign="top">
  <th><code>&nbsp;&nbsp;&nbsp;&nbsp;</code></th>
  <th>Property</th>
  <th><code>&nbsp;</code></th>
  <th>Related function</th>
</tr>
<tr align="left" valign="top">

```


Create missing intermediate groups
H5Pset_create_intermediate_groups

<!-- ??????? REDACTED FOR REVIEW &/OR PUBLICATION ???????
 ??????? NOT RELEVANT TO THE USER PUBLICATION ???????

<p>

Additional writing resources:

<p>

The `Topic-7.7-File Image Operations.pptx` file has a section titled “Digression: Property Lists” that includes slides 8-9. This might be a resource for the property lists chapter in the UG.

This PowerPoint file is located in `C:\1 MEE\Working\hdf5_trunk\projects\PSI\Day 2`.

<i>

This info is incorporated.

</i>

<p>

See slide 48 of `Topic-6-HDF-Parallel.pptx` in `C:\1 MEE\Working\hdf5_trunk\projects\PSI\` Day 1. Parallel HDF5 programming model.

Comment on slide: "HDF5 uses access template object (property list) to control the file access mechanism."

<i>

May be useful if a Parallel I/O example is included.

</i>

</p>

??????? END REDACTION ?????? ?????? -->

</body>

</html>

I

```

<!doctype HTML public "-//W3C//DTD HTML 4.0 Frameset//EN">
<html>
<head>
<title>Chapter 10: Additional Resources</title>

<!--(Meta)=====-->

<!--(Links)=====-->

<link href="ed_styles/NewUGelect.css" rel="stylesheet" type="text/css">

<!--( Begin styles definition )=====-->
<!--   Replaced with external stylesheet 'styles_NewUG.css'.   -->
<!--( End styles definition )=====-->

</head>

<body>

<!-- #BeginLibraryItem "/ed_libs/Copyright.lbi" -->
<!--
*****

* Copyright by The HDF Group.                *
* Copyright by the Board of Trustees of the University of Illinois.    *
* All rights reserved.                        *
*                                           *

* This file is part of HDF5. The full HDF5 copyright notice, including  *
* terms governing use, modification, and redistribution, is contained in  *
* the files COPYING and Copyright.html. COPYING can be found at the root  *
* of the source code distribution tree; Copyright.html can be found at the *

```

* root level of an installed copy of the electronic HDF5 document set and *
* is linked from the top-level documents page. It can also be found at *
* <http://www.hdfgroup.org/HDF5/doc/Copyright.html>. If you do not have *
* access to either file, you may request a copy from help@hdfgroup.org. *

-->

<!-- #EndLibraryItem -->

<!-- HEADER LEFT "HDF5 User's Guide" -->

<!-- HEADER RIGHT "Additional Resources" -->

<div align="center">

12. Additional Resources

```
</div>
```

```
<!-- FOR USE WITH ELECTRONIC VERSION ----->
```

```
<center>
```

```
<table border="0" width="80%">
```

```
<tr>
```

```
<td valign="top" colspan="3">
```

```
<p>These documents supplement the <cite>HDF5 User's Guide</cite>
```

```
and provide additional detailed information for the use and tuning
```

```
of specific HDF5 features.</p></td>
```

```
</tr>
```

```
<tr>
```

```
<td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>
```

```
<td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>
```

```
<td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>
```

```
</tr>
```

```
<tr>
```

```
<td valign="top">
```

```
<a href="http://www.hdfgroup.org/HDF5/examples/">
```

```
HDF5 Examples</a></td>
```

```
<td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>
```

```
<td valign="top">Code examples by API.
```

```
<!-- 11.2.10, keep code examples at the top of the list of links -->
```

</td>

</tr>

<tr><td colspan="3"> </td></tr>

<tr>

<td valign="top">

Chunking in HDF5</td>

<td> </td>

<td valign="top">Structuring the use of chunking and tuning it for
performance.</td>

</tr>

<tr><td colspan="3"> </td></tr>

<tr>

<td valign="top">

Using the Direct Chunk Write Function</td>

<td> </td>

<td valign="top">Describes another way that chunks can be written
to datasets.</td>

</tr>

<tr><td colspan="3"> </td></tr>

<tr>

<td valign="top">

Copying Committed Datatypes with H5Ocopy</td>

<td> </td>


```
<td valign="top">
```

```
Describes how to copy to another file a dataset that uses a committed
datatype or an object with an attribute that uses a committed datatype
so that the committed datatype in the destination file can be used by
multiple objects.</td>
```

```
</tr>
```

```
<tr><td colspan="3">&nbsp;</td></tr>
```

```
<tr>
```

```
<td valign="top"><a href="../Advanced/MetadataCache/index.html">
Metadata Caching in HDF5</a></td>
```

```
<td>&nbsp;&nbsp;&nbsp;&nbsp;</td>
```

```
<td valign="top">Managing the HDF5 metadata cache and tuning it for
performance.</td>
```

```
</tr>
```

```
<tr><td colspan="3">&nbsp;</td></tr>
```

```
<tr><td valign="top">
```

```
<a href="../Advanced/DynamicallyLoadedFilters/HDF5DynamicallyLoadedFilters.pdf">
HDF5 Dynamically Loaded Filters</a>
```

```
<td>&nbsp;</td>
```

```
<td valign="top">
```

```
Describes how an HDF5 application can apply a filter that is not
registered with the HDF5 Library.
```

```
</td>
```

```
</tr>
```

```
<tr><td colspan="3">&nbsp;</td></tr>
```

```
<tr><td valign="top">
  <a href="../Advanced/FileImageOperations/HDF5FileImageOperations.pdf">
    HDF5 File Image Operations</a></td>
<td>&nbsp;</td>
<td valign="top">
  Describes how to work with HDF5 files in memory. Disk I/O is not
  required when file images are opened, created, read from, or
  written to.</td>
</tr>
```

```
<tr><td colspan="3">&nbsp;</td></tr>
```

```
<tr><td valign="top">
  <a href="../Advanced/ModifiedRegionWrites/ModifiedRegionWrites.pdf">
    Modified Region Writes</a></td>
<td>&nbsp;</td>
<td valign="top">
  Describes how to set write operations for in-memory files so that
  only modified regions are written to storage. Available when
  the Core (Memory) VFD is used.
</td>
</tr>
```

```
<tr><td colspan="3">&nbsp;</td></tr>
```

```
<tr><td valign="top"><a href="../Advanced/UsingIdentifiers/index.html">
  Using Identifiers</a></td>
<td>&nbsp;&nbsp;&nbsp;</td>
<td valign="top">
  Describes how identifiers behave and how they should be treated.</td>
</tr>
```

```
<tr><td colspan="3">&nbsp;</td></tr>
```

```
<tr><td valign="top"><a href="../Advanced/UsingUnicode/index.html">
```

```
Using UTF-8 Encoding in </a>
```

```
<br />
```

```
<a href="../Advanced/UsingUnicode/index.html">HDF5 Applications</a>
```

```
</td>
```

```
<td>&nbsp;&nbsp;&nbsp;</td>
```

```
<td valign="top">
```

```
Describes the use of UTF-8 Unicode character encodings in HDF5  
applications.</td>
```

```
</tr>
```

```
<tr><td colspan="3">&nbsp;</td></tr>
```

```
<tr><td valign="top"><a href="../Advanced/FreeingMemory/FreeingMemoryAllocatedByTheHdf5Li-  
brary.pdf">
```

```
Freeing Memory Allocated</a>
```

```
<br />
```

```
<a href="../Advanced/FreeingMemory/FreeingMemoryAllocatedByTheHdf5Library.pdf">by the HDF5  
Library</a>
```

```
</td>
```

```
<td>&nbsp;&nbsp;&nbsp;</td>
```

```
<td valign="top">
```

```
Describes how inconsistent memory management can cause heap  
corruption or resource leaks and possible solutions.</td>
```

```
</tr>
```

```
<tr><td colspan="3">&nbsp;</td></tr>
```

```
<tr>
  <td valign="top"><a href="../Glossary.html">
    HDF5 Glossary</a></td>
  <td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>
  <td valign="top">A glossary of terms.
  <!-- 11.2.10, keep Glossary at the bottom of the list of links --></td>
</tr>

<tr><td colspan="3">&nbsp;&nbsp;&nbsp;</td></tr>
```

```
</table>
</center>
```

```
<!-- FOR USE WITH ELECTRONIC VERSION ----->
```

```
<!-- PRINT VERSION INCLUDES THE TEXT TO THE END OF THE FILE -->
```

```
<!-- FOR USE WITH PRINT VERSION -----
```

<p> This chapter provides supplemental material for the
 <cite>HDF5 User’s Guide</cite>. </p>

<p> To see code examples by API, go to the <cite>HDF5 Examples</cite>
 page at this address:

```
<pre>  http://www.hdfgroup.org/HDF5/examples/  
 </pre></p>
```

<p> For more information on how to manage the metadata cache and how
 to configure it for better performance, go to the <cite>Metadata
 Caching in HDF5</cite> page at this address:

```
<pre>  
 http://www.hdfgroup.org/hdf5/doc/Advanced/MetadataCache/index.html  
 </pre></p>
```

<p> A number of functions are macros. For more information on how
 to use the macros, see the <cite>API Compatibility Macros in HDF5</cite>
 page at this address:

```
<pre>  
 http://www.hdfgroup.org/HDF5/doc/RM/APICompatMacros.html  
 </pre>
```

<p>The following sections are included in this chapter:</p>

-
 <i>Using Identifiers</i> - describes how identifiers behave
 and how they should be treated
 - <i>Chunking in HDF5</i> - describes chunking storage and
 how it can be used to improve performance
 - <i>HDF5 Glossary and Terms</i>
-

<!-- NEW PAGE -->

<!-- PRINT VERSION CONTINUED --

<h2>11.1. Using Identifiers</h2>

<p> The purpose of this section is to describe how identifiers behave and how they should be treated by application programs.</p>

<p> When an application program uses the HDF5 library to create or open an item, a unique identifier is returned. The items that return a unique identifier when they are created or opened include the following: dataset, group, datatype, dataspace, file, attribute, property list, referenced object, error stack, and error message.</p>

<p> An application may open one of the items listed above more than once at the same time. For example, an application might open a group twice, receiving two identifiers. Information from one dataset in the group could be handled through one identifier, and the information from another dataset in the group is handled by a different identifier.</p>

<p> An application program should track every identifier it receives as a result of creating or opening one of the items listed above. In order for an application to close properly, it must release every identifier it has opened. If an application opened a group twice for example, it would need to issue two `H5Gclose` commands, one for each identifier. Not releasing identifiers causes resource leaks. Until an identifier is released, the item associated with the identifier is still open.</p>

<p> The library considers a file open until all of the identifiers associated with the file and with the file's various items have been released. The identifiers associated with these open items must be

released separately. This means that an application can close a file and still work with one or more portions of the file. Suppose an application opened a file, a group within the file, and two datasets within the group. If the application closed the file with `H5Fclose`, then the file would be considered closed to the application, but the group and two datasets would still be open.

There are several exceptions to the above file closing rule. One is when the `H5close` function is used instead of `H5Fclose`. `H5close` causes a general shutdown of the library: all data is written to disk, all identifiers are closed, and all memory used by the library is cleaned up. Another exception occurs on parallel processing systems. Suppose on a parallel system an application has opened a file, a group in the file, and two datasets in the group. If the application uses the `H5Fclose` function to close the file, the call will fail with an error. The open group and datasets must be closed before the file can be closed. A third exception is when the file access property list includes the property `H5F_CLOSE_STRONG`. This property causes the closing of all of the file's open items when the file is closed with `H5Fclose`.

For more information about `H5close`, `H5Fclose`, and `H5Pset_fclose_degree`, see the [HDF5 Reference Manual](http://hdf5.org/doc/1.8.12/RM/RM_H5Front.html).

<!--

-----1-----2-----3-----4-----5-----6-----7-----8

-->

<!-- PRINT VERSION CONTINUED --

<h3>Functions that Return Identifiers</h3>

<p>Some of the functions that return identifiers are listed below.</p>

<code>H5Acreate</code>

<code>H5Acreate_by_name</code>

<code>H5Aget_type</code>

<code>H5Aopen</code>

<code>H5Aopen_by_idx</code>

<code>H5Aopen_by_name</code>

<code>H5Dcreate</code>

<code>H5Dcreate_anon</code>

<code>H5Dget_access_plist</code>

<code>H5Dget_create_plist</code>

<code>H5Dget_space</code>

<code>H5Dget_type</code>

<code>H5Dopen</code>

<code>H5Ecreate_msg</code>

<code>H5Ecreate_stack</code>

<code>H5Fcreate</code>

<code>H5Fopen</code>

<code>H5Freopen</code>

<code>H5Gcreate</code>

<code>H5Gcreate_anon</code>

<code>H5Gopen</code>

<code>H5Oopen</code>

<code>H5Oopen_by_addr</code>

<code>H5Oopen_by_idx</code>

- <code>H5Pcreate</code>
- <code>H5Rdereference</code>
- <code>H5Rget_region</code>
- <code>H5Screate</code>
- <code>H5Screate_simple</code>
- <code>H5Tcopy</code>
- <code>H5Tcreate</code>
- <code>H5Tdecode</code>
- <code>H5Tget_member_type</code>
- <code>H5Tget_super</code>
- <code>H5Topen</code>

<!-- NEW PAGE -->

<!-- PRINT VERSION CONTINUED --

<h2>11.2. Chunking in HDF5</h2>

<p> Datasets in HDF5 not only provide a convenient, structured, and self-describing way to store data, but are also designed to do so with good performance. In order to maximize performance, the HDF5 library provides ways to specify how the data is stored on disk, how it is accessed, and how it should be held in memory.</p>

<!--

-----1-----2-----3-----4-----5-----6-----7-----8

-->

<!-- PRINT VERSION CONTINUED --

<h3>11.2.1. What are Chunks?</h3>

<p> Datasets in HDF5 can represent arrays with any number of dimensions (up to 32). However, in the file this dataset must be stored as part of the 1-dimensional stream of data that is the low-level file.

The way in which the multidimensional dataset is mapped to the serial file is called the layout. The most obvious way to accomplish this is to simply flatten the dataset in a way similar to how arrays are stored in memory, serializing the entire dataset into a monolithic block on disk, which maps directly to a memory buffer the size of the dataset. This is called a contiguous layout.</p>

<p> An alternative to the contiguous layout is the chunked layout.

Whereas contiguous datasets are stored in a single block in the file, chunked datasets are split into multiple chunks which are all stored separately in the file. The chunks can be stored in any order and any position within the HDF5 file. Chunks can then be read and written individually, improving performance when operating on a subset of the dataset. </p>

<p> The API functions used to read and write chunked datasets are exactly the same functions used to read and write contiguous datasets. The only difference is a single call to set up the layout on a property list before the dataset is created. In this way, a program can switch between using chunked and contiguous datasets by simply altering that call. Example 1, below, creates a dataset with a size of 12x12 and a chunk size of 4x4. The example could be change to create a contiguous dataset instead by simply commenting out the call to <code>H5Pset_chunk</code>.</p>

<!-- NEW PAGE -->

<!-- PRINT VERSION CONTINUED --

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
#include <hdf5.h>
int main(void) {
    hid_t   file_id, dset_id, space_id, dcpl_id;
    hsize_t chunk_dims[2] = {4, 4};
    hsize_t dset_dims[2] = {12, 12};
    int     buffer[12][12];

    /* Create the file */
    file_id = H5Fcreate(file.h5, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

    /* Create a dataset creation property list and set it to use chunking
    */
    dcpl_id = H5Pcreate(H5P_DATASET_CREATE);
    H5Pset_chunk(dcpl_id, 2, chunk_dims);

    /* Create the dataspace and the chunked dataset */
    space_id = H5Screate_simple(2, dset_dims, NULL);
    dset_id = H5Dcreate(file, dataset, H5T_NATIVE_INT, space_id, dcpl_id, H5P_DEFAULT);

    /* Write to the dataset */
    buffer = <initialize buffer>
    H5Dwrite(dset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT, buffer);

    /* Close */
    H5Dclose(dset_id);
```

```

H5Sclose(space_id);
H5Pclose(dcpl_id);
H5Fclose(file_id);
return 0;
}
</pre></td>
</tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left">
    <b>Example 1. Creating a chunked dataset</b>
    <hr color="green" size="3" /></td>
  </tr>
</table>
<br />

```

The chunks of a chunked dataset are split along logical boundaries in the dataset's representation as an array, not along boundaries in the serialized form. Suppose a dataset has a chunk size of 2x2. In this case, the first chunk would go from (0,0) to (2,2), the second from (0,2) to (2,4), and so on. By selecting the chunk size carefully, it is possible to fine tune I/O to maximize performance for any access pattern. Chunking is also required to use advanced features such as compression and dataset resizing.

<!-- NEW Page -->

<!-- PRINT VERSION CONTINUED --

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <br />
      
      <br /><br />
    </td></tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 1. Contiguous dataset</b>
      <hr color="green" size="3"/></td></tr>
</table>
<br />

```

```

<!-- NEW Page -->

```

```

<!-- PRINT VERSION CONTINUED --

```

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <br />
      
      <br /><br />
    </td></tr>
  <tr><td><hr color="green" size="1" /></td></tr>

```

```
<tr valign="top">
  <td align="left" >
    <b>Figure 2. Chunked dataset</b>
    <hr color="green" size="3"/></td></tr>
</table>
<br />
```

```
<!-- NEW PAGE -->
<!-- PRINT VERSION CONTINUED --
<h3>11.2.2. Data Storage Order</h3>
```

<p> To understand the effects of chunking on I/O performance it is necessary to understand the order in which data is actually stored on disk. When using the C interface, data elements are stored in "row-major" order, meaning that, for a 2-dimensional dataset, rows of data are stored in-order on the disk. This is equivalent to the storage order of C arrays in memory.</p>

<p> Suppose we have a 10x10 contiguous dataset B. The first element stored on disk is B[0][0], the second B[0][1], the eleventh B[1][0], and so on. If we want to read the elements from B[2][3] to B[2][7], we have to read the elements in the 24th, 25th, 26th, 27th, and 28th positions. Since all of these positions are contiguous, or next to each other, this can be done in a single read operation: read 5 elements starting at the 24th position. This operation is illustrated in figure 3: the pink cells represent elements to be read and the solid line represents a read operation. Now suppose we want to read the elements in the column from B[3][2] to B[7][2]. In this case

we must read the elements in the 33rd, 43rd, 53rd, 63rd, and 73rd positions. Since these positions are not contiguous, this must be done in 5 separate read operations. This operation is illustrated in figure 4: the solid lines again represent read operations, and the dotted lines represent seek operations. An alternative would be to perform a single large read operation, in this case 41 elements starting at the 33rd position. This is called a sieve buffer and is supported by HDF5 for contiguous datasets, but not for chunked datasets. By setting the chunk sizes correctly, it is possible to greatly exceed the performance of the sieve buffer scheme.

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <br />
      
      <br /><br />
    </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 3. Reading part of a row from a contiguous dataset</b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />
<br />
```

```

<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <br />
      
      <br /><br />
    </td>
  </tr>
  <tr><td><hr color="green" size="1" /></td></tr>
  <tr valign="top">
    <td align="left" >
      <b>Figure 4. Reading part of a column from a contiguous dataset</b>
      <hr color="green" size="3"/></td>
    </tr>
</table>
<br />

```

Likewise, in higher dimensions, the last dimension specified is the fastest changing on disk. So if we have a four dimensional dataset A, then the first element on disk would be A[0][0][0][0], the second A[0][0][0][1], the third A[0][0][0][2], and so on.</p>

11.2.3. Chunking and Partial I/O</h3>

The issues outlined above regarding data storage order help to

illustrate one of the major benefits of dataset chunking, its ability to improve the performance of partial I/O. Partial I/O is an I/O operation (read or write) which operates on only one part of the dataset. To maximize the performance of partial I/O, the data elements selected for I/O must be contiguous on disk. As we saw above, with a contiguous dataset, this means that the selection must always equal the extent in all but the slowest changing dimension, unless the selection in the slowest changing dimension is a single element. With a 2-d dataset in C, this means that the selection must be as wide as the entire dataset unless only a single row is selected. With a 3-d dataset, this means that the selection must be as wide and as deep as the entire dataset, unless only a single row is selected, in which case it must still be as deep as the entire dataset, unless only a single column is also selected.

Chunking allows the user to modify the conditions for maximum performance by changing the regions in the dataset which are contiguous. For example, reading a 20x20 selection in a contiguous dataset with a width greater than 20 would require 20 separate and non-contiguous read operations. If the same operation were performed on a dataset that was created with a chunk size of 20x20, the operation would require only a single read operation. In general, if your selections are always the same size (or multiples of the same size), and start at multiples of that size, then the chunk size should be set to the selection size, or an integer divisor of it. This recommendation is subject to the guidelines in the pitfalls section; specifically, it should not be too small or too large.

Using this strategy, we can greatly improve the performance of the

operation shown in figure 4. If we create the dataset with a chunk size of 10x1, each column of the dataset will be stored separately and contiguously. The read of a partial column can then be done in a single operation. This is illustrated in figure 5, and the code to implement a similar operation is shown in example 2. For simplicity, example 2 implements writing to this dataset instead of reading from it.

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <br />
      
      <br /><br />
    </td></tr>
    <tr><td><hr color="green" size="1" /></td></tr>
    <tr valign="top">
      <td align="left" >
        <b>Figure 5. Reading part of a column from a chunked dataset</b>
        <hr color="green" size="3"/></td></tr>
  </table>
<br />
<br />
```

```
<!-- NEW PAGE -->
```

```
<!-- PRINT VERSION CONTINUED --
```

```
<table width="600" cellspacing="0" align="center">
  <tr valign="top">
    <td align="left">
      <hr color="green" size="3"/>
      <pre>
#include <hdf5.h>
int main(void) {
    hid_t   file_id, dset_id, fspace_id, mspace_id, dcpl_id;
    hsize_t chunk_dims[2] = {10, 1};
    hsize_t dset_dims[2] = {10, 10};
    hsize_t mem_dims[1] = {5};
    hsize_t start[2] = {3, 2};
    hsize_t count[2] = {5, 1};
    int     buffer[5];

    /* Create the file */
    file_id = H5Fcreate(file.h5</em>, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

    /* Create a dataset creation property list and set it to use chunking
     * with a chunk size of 10x1 */
    dcpl_id = H5Pcreate(H5P_DATASET_CREATE);
    H5Pset_chunk(dcpl_id, 2, chunk_dims);

    /* Create the dataspace and the chunked dataset */
    space_id = H5Screate_simple(2, dset_dims, NULL);
    dset_id = H5Dcreate(file, dataset</em>, H5T_NATIVE_INT, space_id, dcpl_id,
H5P_DEFAULT);

    /* Select the elements from 3, 2 to 7, 2 */
    H5Sselect_hyperslab(fspace_id, H5S_SELECT_SET, start, NULL, count, NULL);
```


<p> Another major feature of the dataset chunking scheme is the chunk cache. As it sounds, this is a cache of the chunks in the dataset. This cache can greatly improve performance whenever the same chunks are read from or written to multiple times, by preventing the library from having to read from and write to disk multiple times. However, the current implementation of the chunk cache does not adjust its parameters automatically, and therefore the parameters must be adjusted manually to achieve optimal performance. In some rare cases it may be best to completely disable the chunk caching scheme. Each open dataset has its own chunk cache, which is separate from the caches for all other open datasets.</p>

<p> When a selection is read from a chunked dataset, the chunks containing the selection are first read into the cache, and then the selected parts of those chunks are copied into the user's buffer. The cached chunks stay in the cache until they are evicted, which typically occurs because more space is needed in the cache for new chunks, but they can also be evicted if hash values collide (more on this later). Once the chunk is evicted it is written to disk if necessary and freed from memory.</p>

<p> This process is illustrated in figures 6 and 7. In figure 6, the application requests a row of values, and the library responds by bringing the chunks containing that row into cache, and retrieving the values from cache. In figure 7, the application requests a different row that is covered by the same chunks, and the library retrieves the values directly from cache without touching the disk.</p>

```
<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
```

```

    <hr color="green" size="3"/>
    <br />
    
    <br /><br />
  </td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 6. Reading a row from a chunked dataset with the
    chunk cache enabled</b>
    <hr color="green" size="3"/></td></tr>
</table>
<br />
<br />

<table width="600" cellpadding="0" align="center">
  <tr valign="top">
    <td align="center">
      <hr color="green" size="3"/>
      <br />
      
      <br /><br />
    </td></tr>
<tr><td><hr color="green" size="1" /></td></tr>
<tr valign="top">
  <td align="left" >
    <b>Figure 7. Reading a row from a chunked dataset with the
    chunks already cached</b>

```

```
<hr color="green" size="3"/></td></tr>
</table>
<br />
<br />
```

In order to allow the chunks to be looked up quickly in cache, each chunk is assigned a unique hash value that is used to look up the chunk. The cache contains a simple array of pointers to chunks, which is called a hash table. A chunk's hash value is simply the index into the hash table of the pointer to that chunk. While the pointer at this location might instead point to a different chunk or to nothing at all, no other locations in the hash table can contain a pointer to the chunk in question. Therefore, the library only has to check this one location in the hash table to tell if a chunk is in cache or not. This also means that if two or more chunks share the same hash value, then only one of those chunks can be in the cache at the same time. When a chunk is brought into cache and another chunk with the same hash value is already in cache, the second chunk must be evicted first. Therefore it is very important to make sure that the size of the hash table, also called the `nslots` parameter in `H5Pset_cache` and `H5Pset_chunk_cache`, is large enough to minimize the number of hash value collisions.

To determine the hash value for a chunk, the chunk is first assigned a unique index that is the linear index into a hypothetical array of the chunks. That is, the upper-left chunk has an index of 0, the one

to the right of that has an index of 1, and so on. This index is then divided by the size of the hash table, `nslots`, and the remainder, or modulus, is the hash value. Because this scheme can result in regularly spaced indices being used frequently, it is important that `nslots` be a prime number to minimize the chance of collisions. In general, `nslots` should probably be set to a number approximately 100 times the number of chunks that can fit in `nbytes` bytes, unless memory is extremely limited. There is of course no advantage in setting `nslots` to a number larger than the total number of chunks in the dataset.

The `w0` parameter affects how the library decides which chunk to evict when it needs room in the cache. If `w0` is set to 0, then the library will always evict the least recently used chunk in cache. If `w0` is set to 1, the library will always evict the least recently used chunk which has been fully read or written, and if none have been fully read or written, it will evict the least recently used chunk. If `w0` is between 0 and 1, the behaviour will be a blend of the two. Therefore, if the application will access the same data more than once, `w0` should be set closer to 0, and if the application does not, `w0` should be set closer to 1.

It is important to remember that chunk caching will only give a benefit when reading or writing the same chunk more than once. If, for example, an application is reading an entire dataset, with only whole chunks selected for each operation, then chunk caching will not help performance, and it may be preferable to completely disable the chunk cache in order to save memory. It may also be advantageous to disable the chunk cache when writing small amounts to many different chunks, if memory is not large enough to hold all those chunks in cache at once.

11.2.5. I/O Filters and Compression

Dataset chunking also enables the use of I/O filters, including compression. The filters are applied to each chunk individually, and the entire chunk is processed at once. The filter must be applied every time the chunk is loaded into cache, and every time the chunk is flushed to disk. These facts all make choosing the proper settings for the chunk cache and chunk size even more critical for the performance of filtered datasets.

Because the entire chunk must be filtered every time disk I/O occurs, it is no longer a viable option to disable the chunk cache when writing small amounts of data to many different chunks. To achieve acceptable performance, it is critical to minimize the chance that a chunk will be flushed from cache before it is completely read or written. This can be done by increasing the size of the chunk cache, adjusting the size of the chunks, or adjusting I/O patterns.

11.2.6. Pitfalls

Inappropriate chunk size and cache settings can dramatically reduce performance. There are a number of ways this can happen. Some of the more common issues include:

-

- Chunks are too small

There is a certain amount of overhead associated with finding chunks. When chunks are made smaller, there are more of them in the dataset. When performing I/O on a dataset, if there

are many chunks in the selection, it will take extra time to look up each chunk. In addition, since the chunks are stored independently, more chunks results in more I/O operations, further compounding the issue. The extra metadata needed to locate the chunks also causes the file size to increase as chunks are made smaller. Making chunks larger results in fewer chunk lookups, smaller file size, and fewer I/O operations in most cases.

Chunks are too large

It may be tempting to simply set the chunk size to be the same as the dataset size in order to enable compression on a contiguous dataset. However, this can have unintended consequences. Because the entire chunk must be read from disk and decompressed before performing any operations, this will impose a great performance penalty when operating on a small subset of the dataset if the cache is not large enough to hold the one-chunk dataset. In addition, if the dataset is large enough, since the entire chunk must be held in memory while compressing and decompressing, the operation could cause the operating system to page memory to disk, slowing down the entire system.

Cache is not big enough

Similarly, if the chunk cache is not set to a large enough size for the chunk size and access pattern, poor performance will result. In general, the chunk cache should be large enough to fit all of the chunks that contain part of a hyperslab selection used to read or write. When the chunk cache is not large enough, all of the chunks in the selection will be read into cache and then written to disk (if writing) and evicted. If the application then revisits the same chunks, they will have to be read and possibly written

again, whereas if the cache were large enough they would only have to be read (and possibly written) once. However, if selections for I/O always coincide with chunk boundaries, this does not matter as much, as there is no wasted I/O and the application is unlikely to revisit the same chunks soon after.

If the total size of the chunks involved in a selection is too big to practically fit into memory, and neither the chunk nor the selection can be resized or reshaped, it may be better to disable the chunk cache. Whether this is better depends on the storage order of the selected elements. It will also make little difference if the dataset is filtered, as entire chunks must be brought into memory anyways in that case. When the chunk cache is disabled and there are no filters, all I/O is done directly to and from the disk. If the selection is mostly along the fastest changing dimension (i.e. rows), then the data will be more contiguous on disk, and direct I/O will be more efficient than reading entire chunks, and hence the cache should be disabled. If however the selection is mostly along the slowest changing dimension (columns), then the data will not be contiguous on disk, and direct I/O will involve a large number of small operations, and it will probably be more efficient to just operate on the entire chunk, therefore the cache should be set large enough to hold at least 1 chunk. To disable the chunk cache, either `nbytes` or `nslots` should be set to 0.

Improper hash table size

Because only one chunk can be present in each slot of the hash table, it is possible for an improperly set hash table size (`nslots`) to severely impact performance. For example, if there are 100 columns of chunks in a dataset, and the hash

table size is set to 100, then all the chunks in each row will have the same hash value. Attempting to access a row of elements will result in each chunk being brought into cache and then evicted to allow the next one to occupy its slot in the hash table, even if the chunk cache is large enough, in terms of nbytes, to hold all of them. Similar situations can arise when nslots is a factor or multiple of the number of rows of chunks, or equivalent situations in higher dimensions.

Luckily, because each slot in the hash table only occupies the size of the pointer for the system, usually 4 or 8 bytes, there is little reason to keep nslots small. Again, a general rule is that nslots should be set to a prime number at least 100 times the number of chunks that can fit in nbytes, or simply set to the number of chunks in the dataset.

11.2.7. For More Information

!--

-----1-----2-----3-----4-----5-----6-----7-----8

-->

!-- the pdf mentioned in the paragraph below is not available to print readers (ie, it's not on the web site)

The slide set "The slide set

[HDF5](../_topic/Chunking/Chunking_Tutorial_EOS13_2009.pdf)

Advanced Topics: Chunking in HDF5" (PDF), a tutorial from HDF and HDF-EOS Workshop XIII (2009) provides additional HDF5 chunking use cases and examples.

-->

<!-- FOR USE WITH ELECTRONIC VERSION ----->

<!-- 11.18.10, the paragraph below is labeled for the electronic version,
but it doesn't seem to work with the electronic page. I'm keeping this
paragraph commented out. --

<p> The page “<a href=

"http://www.hdfgroup.org/HDF5/examples/api18-c.html">

HDF5 Examples by API” lists many code examples that are
regularly tested with the HDF5 Library. Several illustrate the
use of chunking in HDF5, particularly “Read/Write Chunked
Dataset” and any examples demonstrating filters. </p>

<!-- FOR USE WITH ELECTRONIC VERSION ----->

<!-- FOR USE WITH PRINT VERSION ----->

<!-- PRINT VERSION CONTINUED --

<p> The “HDF5 Examples by API” page,

<code>

http://www.hdfgroup.org/ftp/HDF5/examples/examples-by-api/api18-c.html,

</code>

lists many code examples that are regularly tested with the HDF5
Library. Several illustrate the use of chunking in HDF5, particularly
“Read/Write Chunked Dataset” and any examples
demonstrating filters. </p>

<!-- FOR USE WITH PRINT VERSION ----->

<!-- FOR USE WITH ELECTRONIC VERSION ----->

<!-- 11.19.10, section 11.2.8. is labeled for the electronic version,
but doesn't seem to be appropriate for the page. I'm leaving it commented
out. --
<!--
<h3>11.2.8. Directions for Future Development</h3>

As seen above, the HDF5 chunk cache currently requires careful control of the parameters in order to achieve optimal performance. In the future, we plan to improve the chunk cache to be more foolproof in many ways, and deliver acceptable performance in most cases even when no thought is given to the chunking parameters.

<p>

One way to make the chunk cache more user-friendly is to automatically resize the chunk cache as needed for each operation. The cache should be able to detect when the cache should be skipped or when it needs to be enlarged based on the pattern of I/O operations. At a minimum, it should be able to detect when the cache would severely hurt performance for a single operation and disable the cache for that operation. This would of course be optional.

<p>

Another way is to allow chaining of entries in the hash table. This would make the hash table size much less of an issue, as chunks could share the same hash value by making a linked list.

<p>

Finally, it may even be desirable to set some reasonable default chunk size based on the dataset size and possibly some other information on the intended access pattern. This would probably be a high-level routine.

<p>

Other features planned for chunking include new index methods (besides b-trees), disabling filters for chunks that are partially over the edge of a dataset, only storing the used portions of these edge chunks, and allowing multiple reader processes to read the same dataset as a single writer process writes to it.

[illegible]

```
<!-- FOR USE WITH ELECTRONIC VERSION ----->
<!-- PRINT VERSION CONTINUED --
<br />
<!-- NEW PAGE -->
<!-- PRINT VERSION CONTINUED --
<h2>11.3. HDF5 Glossary and Terms</h2>
```

```
<dl>

<dt><strong><a name="Glossary-AtomicDType">atomic datatype</a></strong></dt>

  <dd>A datatype which cannot be decomposed into smaller units at the
    API level. </dd>

<br />

<dt><a name="Glossary-Attribute"><b>attribute</b></a></dt>

<dd>A small dataset that can be used to describe the nature and/or
  the intended usage of the object it is attached to.</dd>
```

<!--

<dt>basic datatypes</dt>

char - 8-bit character (only for ASCII information)

int8 - 8-bit signed integer

uint8 - 8-bit unsigned integer

int16 - 16-bit signed integer

uint16 - 16-bit unsigned integer

int32 - 32-bit signed integer

uint32 - 32-bit unsigned integer

intn - "native" signed integer

uintn - "native" unsigned integer

int64 - 64-bit signed integer (new)

uint64 - 64-bit unsigned integer (new)

float32 - 32-bit IEEE float

float64 - 64-bit IEEE float

-->

<!-- PRINT VERSION CONTINUED --

<dt>chunked layout</dt>

<dd>The storage layout of a chunked dataset.</dd>

<dt>chunking</dt>

<dd>A storage layout where a dataset is partitioned into fixed-size

multi-dimensional chunks. Chunking tends to improve performance

and facilitates dataset extensibility.</dd>

<dt>committed datatype</dt>

<dd>A datatype that is named and stored in a file so that it can be shared.

Committed datatypes can be shared. Committing is permanent; a datatype cannot be changed after being committed. Committed datatypes used to be called named datatypes.</dd>

<dt>compound datatype</dt>

<dd>A collection of one or more atomic types or small arrays of such types.

Similar to a struct in C or a common block in Fortran.</dd>

<!--

<dt>complex datatype</dt>

<dd>A collection of one or more atomic types or small arrays of such types.

hid_t - 32-bit unsigned integer used as ID for memory objects

hoid_t - 32-bit unsigned integer (currently) used as ID for disk-based objects

hbool_t - boolean to indicate true/false/error codes from functions

herr_t - 32-bit integer to indicate succeed/fail codes from functions

</dd>

-->

<!-- PRINT VERSION CONTINUED --

<dt>contiguous layout</dt>

<dd>The storage layout of a dataset that is not chunked, so that the entire data portion of the dataset is stored in a single contiguous block.</dd>

<dt>data transfer property list</dt>

<dd>The data transfer property list is used to control various aspects
of the I/O, such as caching hints or collective I/O information.</dd>

<dt>dataset</dt>

<dd>A multi-dimensional array of data elements, together with
supporting metadata. </dd>

<dt>dataset access property list</dt>

<dd>A property list containing information on how a dataset is to be accessed.</dd>

<dt>dataset creation property list</dt>

<dd>A property list containing information on how
raw data is organized on disk and how the raw data is compressed.</dd>

<!--

The dataset API partitions these terms by layout, compression,
and external storage:

Layout:

H5D_COMPACT: Data is small and can be stored in object header (not
implemented yet). This eliminates disk seek/read requests.

H5D_CONTIGUOUS: (default) The data is large, non-extendible,
non-compressible, non-sparse, and can be stored externally.

H5D_CHUNKED: The data is large and can be extended in any dimension.
It is partitioned into chunks so each chunk is the same logical size.

Compression: (gzip compression)

External Storage Properties: The data must be contiguous to be stored externally. It allows you to store the data in one or more non-HDF5 files.

-->

<!-- PRINT VERSION CONTINUED --

<dt>dataspace</dt>

<dd>An object that describes the dimensionality of the data array.

A dataspace is either a regular N-dimensional array of data points, called a simple dataspace, or a more general collection of data points organized in another manner, called a complex dataspace.</dd>

<!-- NEW PAGE -->

<!-- PRINT VERSION CONTINUED --

<dt>datatype</dt>

<dd>An object that describes the storage format of the individual data points of a data set.

There are two categories of datatypes: atomic and compound datatypes.

An atomic type is a type which cannot be decomposed into smaller units at the API level. A compound datatype is a collection of one or more atomic types or small arrays of such types.</dd>

<!--

<dt>DDL</dt>

<dd>A detailed description of the HDF5 format and objects, written in a Data Description Language using Backus-Naur Form.

-->

<!--

<dt>disk I/O datatypes</dt>

- hoff_t - (64-bit?) offset on disk in bytes

- hlen_t - (64-bit?) length on disk in bytes

-->

<!-- PRINT VERSION CONTINUED --

<dt>enumeration datatype</dt>

<dd>A one-to-one mapping between a set of symbols and a set of

integer values, and an order is imposed on the symbols by their

integer values. The symbols are passed between the application

and library as character strings and all the values for a

particular enumeration datatype are of the same integer type,

which is not necessarily a native type.</dd>

<dt>file</dt>

<dd>A container for storing grouped collections of

multi-dimensional arrays containing scientific data. </dd>

<dt>file access mode</dt>

<dd>Determines whether an existing file will be overwritten,

opened for read-only access, or opened for read/write access.

All newly created files are opened for both reading and

writing. </dd>

```
<!--
```

```
Possible values are:
```

```
<PRE>
```

```
H5F_ACC_RDWR: Allow read and write access to file.
```

```
H5F_ACC_RDONLY: Allow read-only access to file.
```

```
H5F_ACC_TRUNC: Truncate file, if it already exists, erasing all data
                  previously stored in the file.
```

```
H5F_ACC_EXCL: Fail if file already exists.
```

```
H5F_ACC_DEBUG: Print debug information.
```

```
H5P_DEFAULT: Apply default file access and creation properties.
```

```
</PRE>
```

```
-->
```

```
<!-- PRINT VERSION CONTINUED --
```

```
<br />
```

```
<dt><b><a name="Glossary-PListFileAccess">file access property list</a></b></dt>
```

```
<dd>File access property lists are used to control different methods
    of performing I/O on files.</dd>
```

```
<!--
```

```
<ul>
```

```
<b>Unbuffered I/O:</b> Local permanent files can be accessed with the
    functions described in Section 2 of the Posix manual, namely open(),
    lseek(), read(), write(), and close(). <br />
```

```
<b>Buffered I/O:</b> Local permanent files can be accessed with the
    functions declared in the stdio.h header file, namely fopen(),
    fseek(), fread(), fwrite(), and fclose().<br />
```

```
<b>Memory I/O:</b> Local temporary files can be created and accessed
    directly from memory without ever creating permanent storage.
    The library uses malloc() and free() to create storage space for the
    file<br />
```

```
<b>Parallel Files using MPI I/O:</b> This driver allows parallel access
```

to a file through the MPI I/O library. The parameters which can be modified are the MPI communicator, the info object, and the access mode.

The communicator and info object are saved and then passed to MPI_File_open() during file creation or open. The access_mode controls the kind of parallel access the application intends.

Data Alignment: Sometimes file access is faster if certain things are aligned on file blocks. This can be controlled by setting alignment properties of a file access property list with the H5Pset_alignment() function.

-->

<!-- PRINT VERSION CONTINUED --

<dt>file creation property list</dt>

<dd>The property list used to control file metadata. </dd>

<!--

The parameters that can be modified are:

User-Block Size: The "user-block" is a fixed length block of data located at the beginning of the file which is ignored by the HDF5 library and may be used to store any data information found to be useful to applications.

Offset and Length Sizes: The number of bytes used to store the offset and length of objects in the HDF5 file can be controlled with this parameter.

Symbol Table Parameters: The size of symbol table B-trees can be controlled by setting the 1/2 rank and 1/2 node size parameters of the B-tree.

Indexed Storage Parameters: The size of indexed storage

B-trees can be controlled by setting the 1/2 rank and 1/2 node
size parameters of the B-tree.

-->

<!-- PRINT VERSION CONTINUED --

<dt>group</dt>

<dd>A structure containing zero or more HDF5 objects,
together with supporting metadata.

The two primary HDF5 objects are datasets and groups.</dd>

<dt>hard link</dt>

<dd>A direct association between a name and the object where both exist
in a single HDF5 address space.</dd>

<!--

<dt>HDF5</dt>

<dd>HDF5 is an abbreviation for Hierarchical Data Format Version 5.

This file format is intended to make it easy to write and read
scientific data

by including the information needed to understand the data
within the file

by providing a library of C, FORTRAN, and other language

programs that reduce the work required to provide efficient
writing and reading - even with parallel IO

A portion of a dataset. A hyperslab selection can be a
logically contiguous collection of points in a dataspace or
a regular pattern of points or blocks in a dataspace.

A unique entity provided by the HDF5 library and used to access
an HDF5 object such as a file, group, or dataset. In the past,
an identifier might have been called a handle.

An association between a name and the object in an HDF5 file group.

A group or dataset that is in another dataset, *dataset A*,
is a member of *dataset A*.

A slash-separated list of components that uniquely identifies an
element of an HDF5 file. A name begins that begins with a slash

is an absolute name which is accessed beginning with the root group of the file; all other names are relative names and the associated objects are accessed beginning with the current or specified group.</dd>

<dt>opaque datatype</dt>
<dd>A mechanism for describing data which cannot be otherwise described by HDF5. The only properties associated with opaque types are a size in bytes and an ASCII tag.</dd>

<!--
<dt>parallel I/O HDF5</dt>
<dd>The parallel I/O version of HDF5 supports parallel file access using MPI (Message Passing Interface). </dd>

-->

<!-- PRINT VERSION CONTINUED --
<dt>path</dt>
<dd>The slash-separated list of components that forms the name uniquely identifying an element of an HDF5 file.</dd>

<dt>property list</dt>
<dd>A collection of name/value pairs that can be passed to other HDF5 functions to control features that are typically unimportant or whose default values are usually used. </dd>

<dt>root group</dt>
<dd>The group that is the entry point to the group graph in an HDF5 file.

Every HDF5 file has exactly one root group.</dd>

<dt>selection</dt>

<dd>(1) A subset of a dataset or a dataspace, up to the entire dataset or dataspace.

(2) The elements of an array or dataset that are marked for I/O.</dd>

<dt>serialization</dt>

<dd>The flattening of an N-dimensional data object into a 1-dimensional object so that, for example, the data object can be transmitted over the network as a 1-dimensional bitstream.</dd>

<dt>soft link</dt>

<dd>An indirect association between a name and an object in an HDF5 file group.</dd>

<dt>storage layout</dt>

<dd>The manner in which a dataset is stored, either contiguous or chunked, in the HDF5 file.</dd>

<dt>super block</dt>

<dd>A block of data containing the information required to portably access HDF5 files on multiple platforms, followed by information about the groups and datasets in the file.

The super block contains information about the size of offsets, lengths of objects, the number of entries in group tables,

and additional version information for the file. </dd>

<!--

<dt>threadsafe</dt>

<dd>A "thread-safe" version of HDF-5 (TSHDF5) is one that can be called from any thread of a multi-threaded program. Any calls to HDF can be made in any order, and each individual HDF call will perform correctly. A calling program does not have to explicitly lock the HDF library in order to do I/O. Applications programmers may assume that the TSHDF5 guarantees the following:

the HDF-5 library does not create or destroy threads.

the HDF-5 library uses modest amounts of per-thread private memory.

the HDF-5 library only locks/unlocks it's own locks (no locks are passed in or returned from HDF), and the internal locking is guaranteed to be deadlock free.

These properties mean that the TSHDF5 library will not interfere with an application's use of threads. A TSHDF5 library is the same library as regular HDF-5 library, with additional code to synchronize access to the HDF-5 library's internal data structures. </dd>

-->

<!-- PRINT VERSION CONTINUED --

<dt>variable-length datatype</dt>

<dd>A sequence of an existing datatype (atomic, variable-length (VL), or compound) which are not fixed in length from one dataset location to another.</dd>

</dl>

<!-- FOR USE WITH PRINT VERSION ----->

</body>

</html>

I