High Level Introduction to HDF5

CONTENTS

Introduction to HDF5

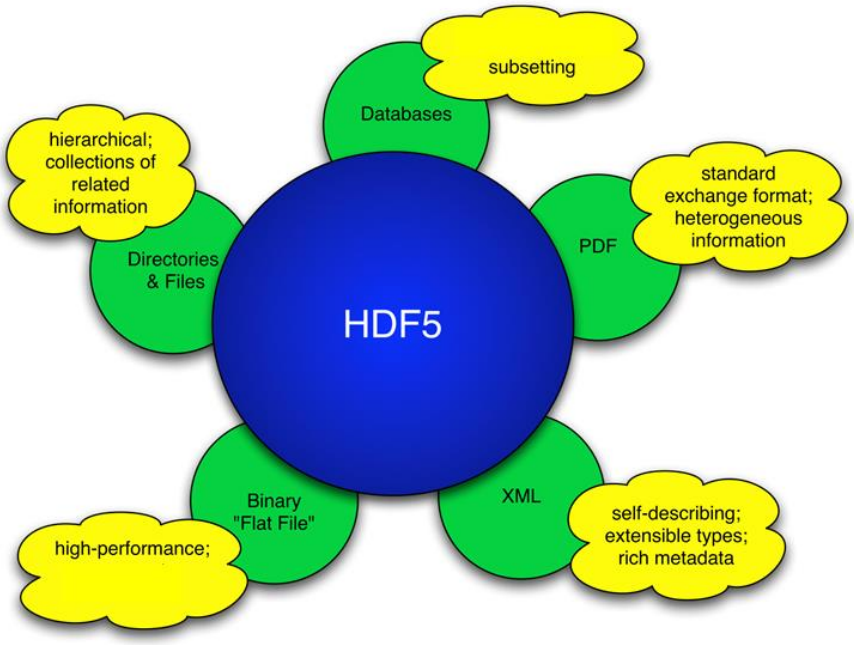
Hierarchical Data Format 5 (HDF5) is a unique *open source* technology suite for managing data collections of all sizes and complexity.

HDF5 was specifically designed:

- For high volume and/or complex data (but can be used for low volume/simple data)
- For every size and type of system (portable)
- For flexible, efficient storage and I/O
- To enable applications to evolve in their use of HDF5 and to accommodate new models
- To be used as a file format tool kit (many formats use HDF5 under the hood)

HDF5 has features of other formats but it can do much more. HDF5 is similar to XML in that HDF5 files are self-describing and allow users to specify complex data relationships and dependencies. In contrast to XML documents, HDF5 files can contain binary data (in many representations) and allow direct access to parts of the file without first parsing the entire contents.

HDF5 also allows hierarchical data objects to be expressed in a natural manner (similar to directories and files), in contrast to the tables in a relational database. Whereas relational databases support tables, HDF5 supports n-dimensional datasets and each element in the dataset may itself be a complex object. Relational databases offer excellent support for queries based on field matching, but are not well-suited for sequentially processing all records in the database or for selecting a subset of the data based on coordinate-style lookup.



HDF5 Description

HDF5 consists of:

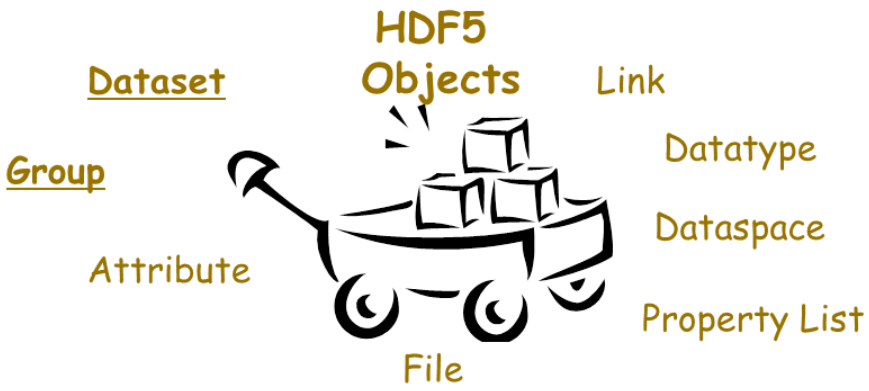
- A *File Format* for storing HDF5 data.
- A *Data Model* for logically organizing and accessing HDF5 data from an application.
- The *Software* (libraries, language interfaces, and tools) for working with this format.

File Format

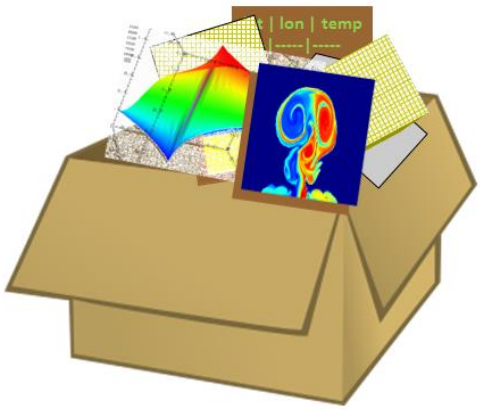
The HDF5 File Format is defined by and adheres to the HDF5 File Format Specification, which specifies the bit-level organization of an HDF5 file on storage media. *In general users do not need to know details about it*.

Data Model

The HDF5 Data Model, also known as the HDF5 Abstract (or Logical) Data Model consists of the building blocks for data organization and specification in HDF5.



An HDF5 file (an object in itself) can be thought of as a container (or group) that holds a variety of heterogeneous data objects (or datasets). The datasets can be most anything: images, tables, graphs, or even documents, such as PDF or Excel:



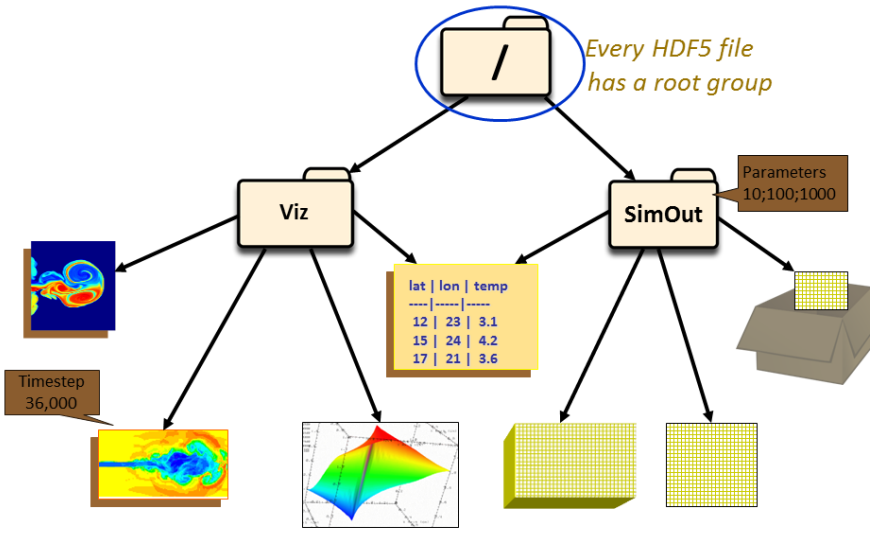
The two primary objects in the HDF5 Data Model are *groups* and *datasets*:

> *group:* a grouping structure containing instances of zero or more groups or datasets, together with supporting metadata.
>
> *dataset:* a multidimensional array of data elements, together with supporting metadata.

There are also a variety of other objects in the HDF5 Data Model that support groups and datasets, including *datatypes*, *dataspaces*, *properties* and *attributes*.

Groups

HDF5 groups (and links) organize data objects. Every HDF5 file contains a root group that can contain other groups or be linked to objects in other files.

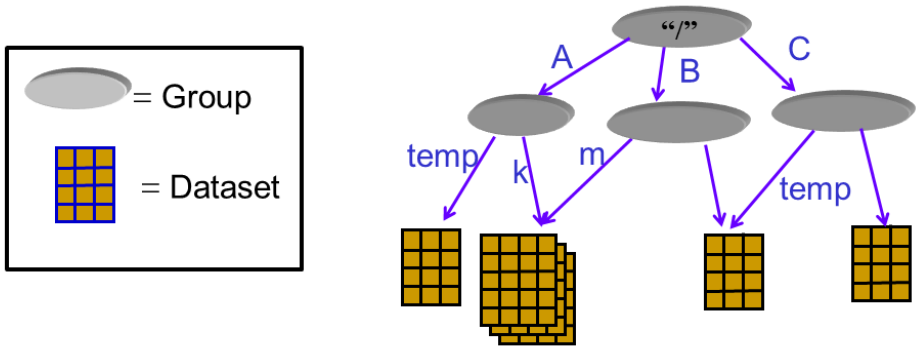


There are two groups in the HDF5 file depicted above: Vis and SimOut. Under the Viz group are a variety of images and a table that is shared with the SimOut group. The SimOut group contains a 3-dimensional array, a 2-dimensional array and a link to a 2-dimensional array in another HDF5 file.

Working with groups and group members is similar in many ways to working with directories and files in UNIX. As with UNIX directories and files, objects in an HDF5 file are often described by giving their full (or absolute) path names.

`/` signifies the root group.
`/foo` signifies a member of the root group called `foo`.
`/foo/zoo` signifies a member of the group `foo`, which in turn is a member of the root group.

An object such as a dataset in a group is defined by its group path:



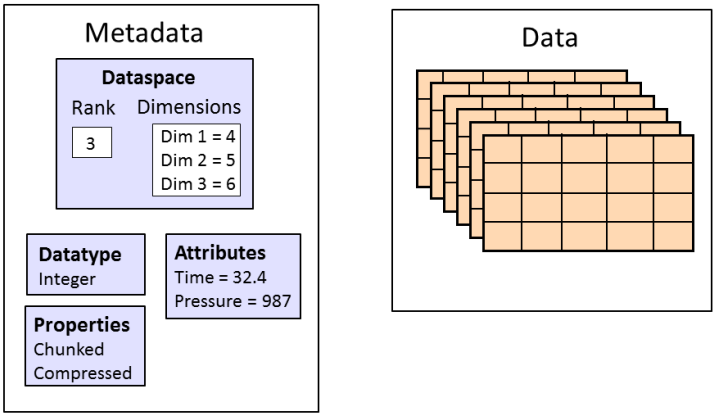
The dataset */C/temp* is a different dataset than */A/temp*.

Also, objects can be shared, so there can be multiple paths to the same objects. In the picture above `/A/k` and `/B/m` point to the same object.

For information on groups see the Groups chapter of the HDF5 User's Guide.

Datasets

HDF5 datasets organize and contain the "raw" data values. A dataset consists of metadata that describes the data, in addition to the data itself:



In the picture above, the data is stored as a three dimensional dataset of size 4 x 5 x 6 with an integer datatype. It contains attributes, *Time* and *Pressure*, and the dataset is chunked and compressed.

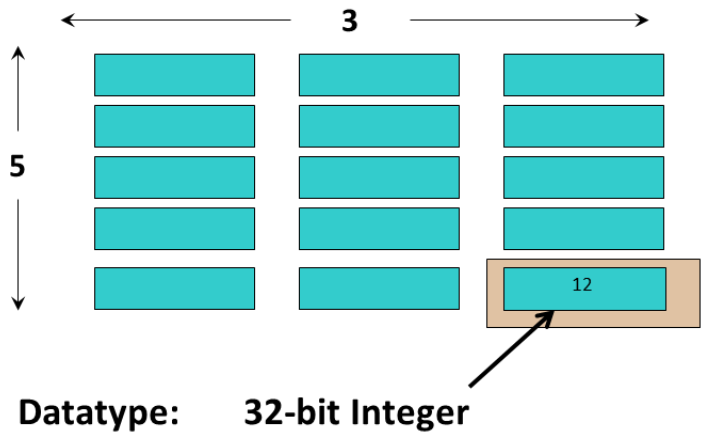
Datatypes, dataspaces, properties and (optional) attributes are HDF5 objects that describe a dataset. The datatype describes the individual data elements.

For information on HDF5 datasets see the Datasets chapter in the HDF5 User's Guide.

Datatypes, Dataspaces, Properties and Attributes

Datatypes

The datatype describes the individual data elements in a dataset. It provides complete information for data conversion to or from that datatype.



In the dataset depicted above each element of the dataset is a 32-bit integer.

Datatypes in HDF5 can be grouped into:

Pre-Defined Datatypes: These are datatypes that are created by HDF5. They are actually opened (and closed) by HDF5 and can have different values from one HDF5 session to the next. There are two types of pre-defined datatypes:

- Standard datatypes are the same on all platforms and are what you see in an HDF5 file.
- Native datatypes are used to simplify memory operations (reading, writing) and are NOT the same on different platforms.

Derived Datatypes: These are datatypes that are created or derived from the pre-defined datatypes. An example of a commonly used derived datatype is a *string* of more than one character. Nested compound datatypes are also derived types.

Pre-defined datatypes have standard symbolic names of the form H5T_ARCH_BASE where ARCH is an architecture name and BASE is a programming type name:

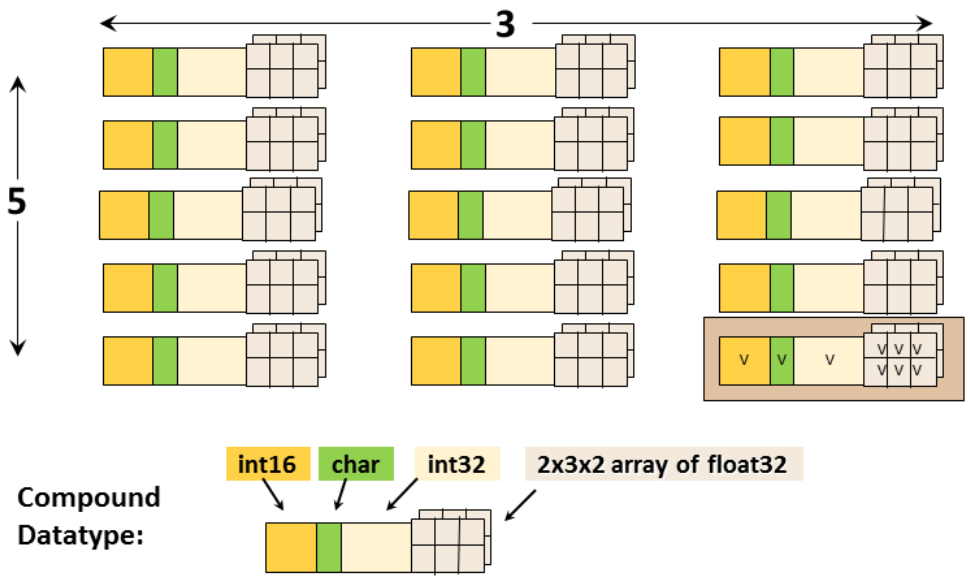
`H5T_IEEE_F32BE IEEE` indicates standard floating point types.

	`F32BE` signifies 32-bit Big Endian floating point.
`H5T_STD_I8LE`	`STD` indicates semi-standard datatypes. `I8LE` indicates 8-bit Little Endian Integer.
`H5T_C_S1`	`C` indicates a type specific to the C programming language. `S1` signifies a one character string.

HDF5 supports a wide variety of datatypes:

- Integer – twos complement integers
- Float – floating point numbers
- Character – array of 1-byte character encoding
- Variable-length sequence types
- Reference – a reference to another object or dataset region within the HDF5 file
- Enumeration – a list of discrete values with symbolic names
- Opaque – uninterpreted (by HDF5)
- Compound (similar to C structs) – a datatype of a sequence of datatypes
- User-defined (eg, 13-bit integer or fixed/variable length strings)
- Nested types

A datatype can be stored as a separate object in an HDF5 file by *commiting* it. A commited datatype can be shared by datasets or attributes.



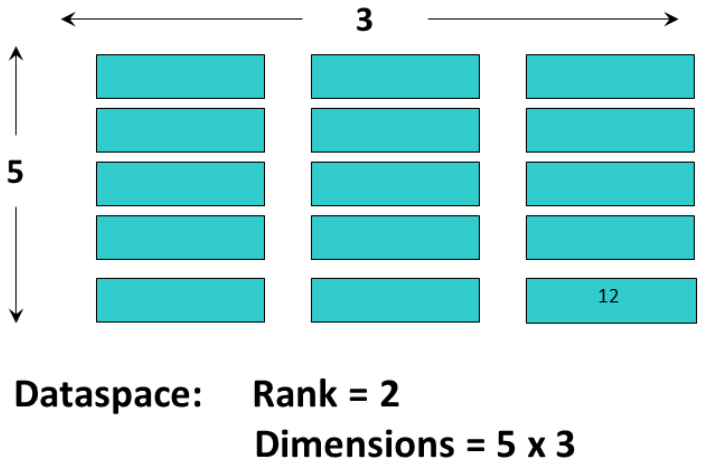
This is an example of a dataset with a compound datatype. Each element in the dataset consists of a 16-bit integer, a character, a 32-bit integer, and a 2x3x2 array of 32-bit floats (the datatype). It is a 2-dimensional 5 x 3 array (the dataspace). The datatype should not be confused with the dataspace.

A compound datatype can be used to create a simple *table*. See the HDF5 Table (H5TB) interface for working with tables. A compound datatype can also be nested, in which it includes one more other compound datatypes.

For complete details regarding datatypes, see the Datatypes chapter in the HDF5 User's Guide.

Dataspaces

A dataspace describes the layout of a dataset's data elements. It can consist of no elements (NULL), a single element (scalar), or a simple array.



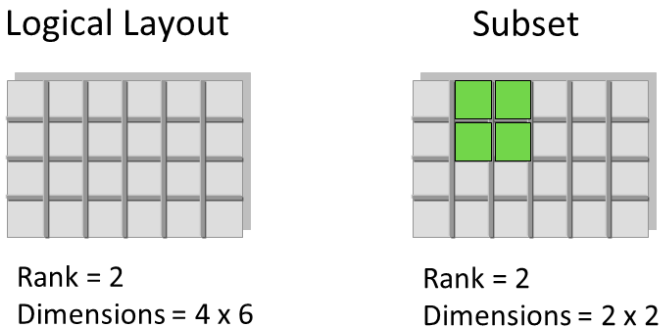
This image illustrates a dataspace that is an array with dimensions of 5 x 3 and a rank (number of dimensions) of 2.

A dataspace can have dimensions that are fixed (unchanging) or *unlimited*, which means they can grow in size (i.e. they are extendible).

There are two roles of a dataspace:

- It contains the spatial information (logical layout) of a dataset stored in a file. This includes the rank and dimensions of a dataset, which are a permanent part of the dataset definition.

- It describes an application's data buffers and data elements participating in I/O. In other words, it can be used to select a portion or subset of a dataset.



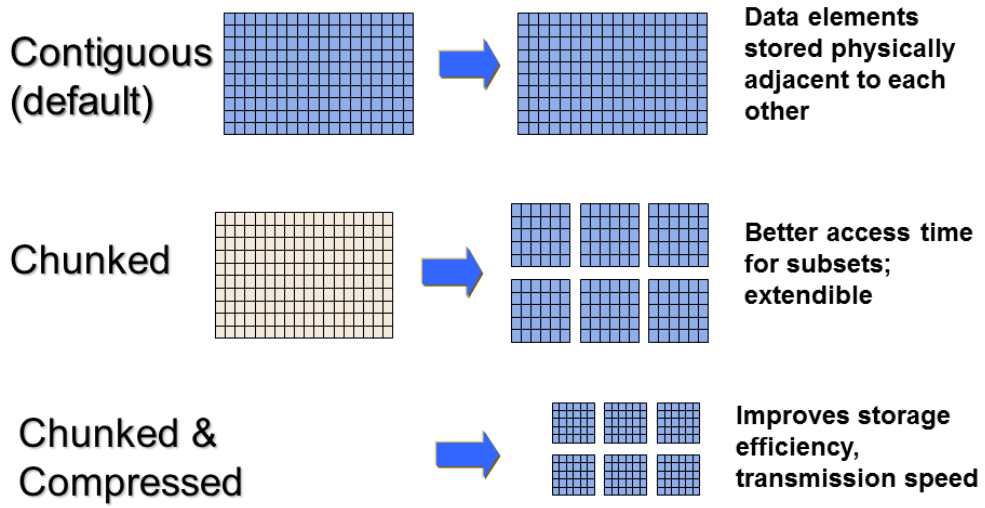
The dataspace is used to describe both the logical layout of a dataset and a subset of a dataset.

For information on dataspaces and partial I/O see the Dataspaces chapter in the HDF5 User's Guide.

Properties

A property is a characteristic or feature of an HDF5 object. There are default properties which handle the most common needs. These default properties can be modified using the HDF5 Property List API to take advantage of more powerful or unusual features of HDF5 objects.

For example, the data storage layout property of a dataset is contiguous by default. For better performance, the layout can be modified to be *chunked* or *chunked and compressed*:



For information on properties see the Properties and Property Lists chapter in the HDF5 User's Guide.

Attributes

Attributes can optionally be associated with HDF5 objects. They have two parts: a name and a value. Attributes are accessed by opening the object that they are attached so are not independent objects. Typically an attribute is small in size and contains user metadata about the object that it is attached to.

Attributes look similar to HDF5 datasets in that they have a datatype and dataspace. However, they do not support partial I/O operations, and they cannot be compressed or extended.

For information on attributes see the Attributes chapter in the HDF5 User's Guide.

HDF5 Software

The HDF5 software is written in C and includes optional wrappers for C++, FORTRAN (90 and F2003), and Java. The HDF5 binary distribution consists of the HDF5 libraries, include files, command-line utilities, scripts for compiling applications, and example programs.

HDF5 runs on a range of computational platforms, from laptops to massively parallel systems. It can be obtained from the HDF5 home page.

HDF5 APIs and Libraries

There are APIs for each type of object in HDF5. For example, all C routines in the HDF5 library begin with a prefix of the form `H5*`, where * is one or two uppercase letters indicating the type of object on which the function operates:

H5A	**A**ttribute Interface
H5D	**D**ataset Interface
H5F	**F**ile Interface
H5G	**G**roup Interface
H5L	**L**ink Interface
H5O	**O**bject Interface
H5P	**P**roperty List Interface
H5S	Data**S**pace Interface
H5T	Data**T**ype Interface

Similarly the FORTRAN wrappers come in the form of subroutines that begin with `h5` and end with `_f`.

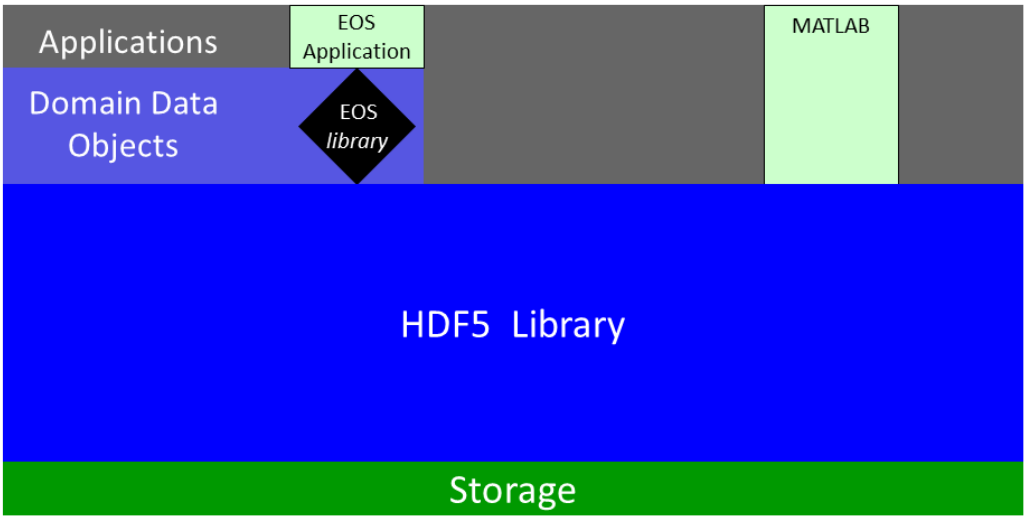
The HDF5 High Level APIs simplify many of the steps required to create and access objects, as well as providing templates for storing objects. Following is a list of the High Level APIs:

- HDF5 Lite (H5LT) – simplifies steps in creating datasets and attributes
- HDF5 Image (H5IM) – defines a standard for storing images in HDF5
- HDF5 Table (H5TB) – condenses the steps required to create tables
- HDF5 Dimension Scales (H5DS) – provides a standard for dimension scale storage
- HDF5 Packet Table (H5PT) – provides a standard for storing packet data

Third Party Software

HDF5 users and enthusiasts have created and are maintaining a variety of add-ons, high-level libraries, plugins, language bindings, and applications. This long list includes tools such as PyTables and h5py, and plugins for applications such as ParaView and VisIt. Examples of community standards with HDF-based representations are NASA's HDF-EOS5, NeXus, CGNS, Energistics' RESQML, and SEG's H5EM-TS. Unidata's netCDF-4 software uses HDF5 as the data management layer.

Commercial off-the-shelf tools supporting HDF5 include IDL, Matlab, Mathematica, Intel Array Visualizer and Array Viewer, EnSight, and Tecplot.

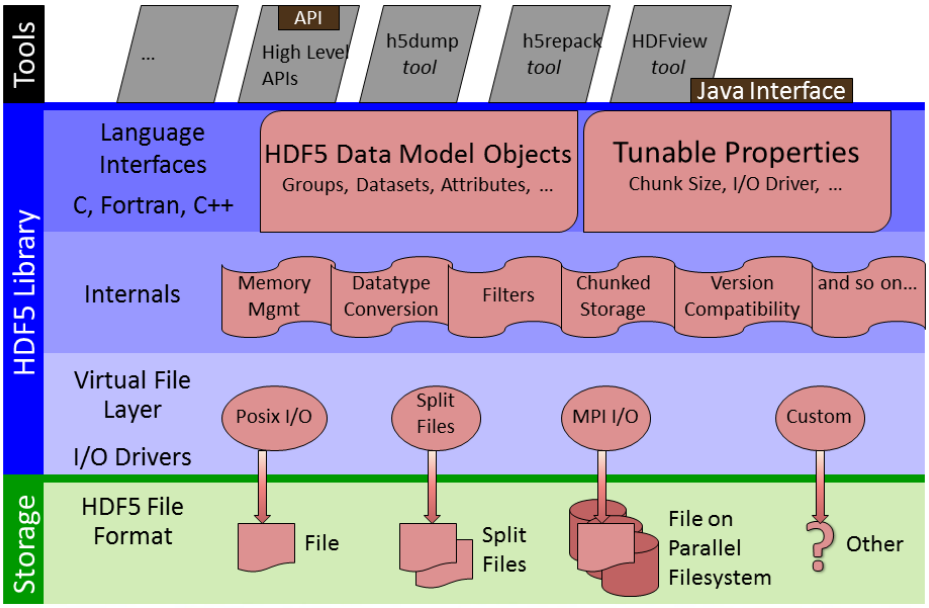


A variety of applications, libraries, and commerical tools use HDF5 to store their data.

Layers in HDF5

At its highest level HDF5 provides tools and APIs for compiling applications with C, FORTRAN, C++, and Java. Users can modify internal behavior by use of tunable properties. At its lowest

level HDF5 consists of the virtual file layer which includes I/O drivers for interfacing with different file types. In between these layers are the HDF5 internals. The most commonly used (and default) I/O driver interfaces with a single file. However, there are other I/O drivers that are available. The MPI I/O driver allows different processors to interface with a given file in a parallel filesystem. The Split File I/O driver enables HDF5 metadata to be stored in one file and the data in another. Users can even create their own custom I/O drivers with HDF5.



Tools

Useful tools for working with HDF5 files include:

h5dump:	A utility to dump or display the contents of an HDF5 File
h5cc, h5c++, h5fc:	Unix scripts for compiling applications
HDFView:	A java browser to view HDF (HDF4 and HDF5) files

For a complete list of the tools that are available with HDF5 see the HDF5 Tools page.

h5dump:

The `h5dump` utility displays the contents of an HDF5 file in Data Description Language (DDL).

Below is an example of `h5dump` output for an HDF5 file that contains no objects:

```
$ h5dump file.h5
HDF5 "file.h5" {
GROUP "/" {
}
}
```

With large files and datasets the output from `h5dump` can be overwhelming. There are options that can be used to examine specific parts of an HDF5 file. Some useful `h5dump` options are included below:

`-H, --header`	Display header information only (no data)
`-d <name>`	Display a dataset with a specified path and name
`-p`	Display properties
`-n`	Display the contents of the file

See the HDF5 Command-line Tools tutorial for information on `h5dump` and other HDF5 tools.

h5cc, h5fc, h5c++:

The built HDF5 binaries include the `h5cc`, `h5fc`, `h5c++` compile scripts for compiling applications. When using these scripts there is no need to specify the HDF5 libraries and include files. Compiler options can be passed to the scripts:

```
h5cc myprog.c
./a.out
```

HDFView:

The `HDFView` tool allows browsing of data in HDF (HDF4 and HDF5) files. For more information on `HDFView` and the Java Products see the HDF-Java home page. With the HDF5-1.10 release the HDF-Java wrappers are included within HDF5. Prior to that they were provided separately.

To learn more about `HDFView` see the HDFView tutorial topic. It demonstrates how to create and access HDF5 files and datasets without the need for programming experience.

Introduction to the HDF5 Programming Model and APIs

The HDF5 Application Programming Interface is extensive (containing hundreds of functions). However a few functions do most of the work. Users can start with the basic functions and build upon their knowledge as needed.

To introduce the programming model, examples in Python, C, and FORTRAN are included below. The HDF5 Python APIs (h5py) are included for their simplicity of use. However, they were not created and are not maintained by The HDF Group.

The general paradigm for working with objects in HDF5 is to:

- Open the object.
- Access the object.
- Close the object.

The library imposes an order on the operations by argument dependencies. For example, a file must be opened before a dataset because the dataset open call requires a file handle as an argument. Objects can be closed in any order. However, once an object is closed it no longer can be accessed.

Keep the following in mind when looking at the example programs included in this section:

- C routines begin with the prefix "`H5*`" where * is a single letter indicating the object on which the operation is to be performed. FORTRAN routines are similar; they begin with "`h5*`" and end with "`_f`". For example:

  File Interface: `H5Fopen` (C) and `h5fopen_f` (FORTRAN)
  Dataset Interface: `H5Dopen` (C) and `h5fdopen_f` (FORTRAN)
  Dataspace interface: `H5Sclose` (C) and `h5sclose_f` (FORTRAN)

  The HDF5 Python APIs use methods associated with specific objects.

- For portability, the HDF5 library has its own defined types. Some common types that you will see in the example code are:

  `hid_t` is used for object handles
  `hsize_t` is used for dimensions
  `herr_t` is used for many return values

- Language specific files must be included in applications:

Python:	Add "`import h5py`" / "`import numpy`"
C:	Add "`#include hdf5.h`"
FORTRAN:	Add "`USE HDF5`" Call `h5open_f` and `h5close_f` to initialize and close the HDF5 FORTRAN interface

These examples are based on or are from the Learning the Basics topic in the HDF5 Tutorial.

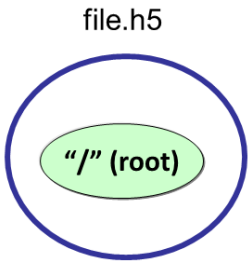
Steps to create a file:

To create an HDF5 file you must:

1. Specify property lists (or use the defaults).
2. Create the file.
3. Close the file (and property lists if needed).

Example:

This example creates a file, `file.h5`, and then closes it:



Following are examples to create a file `file.h5` using Python, C, and FORTRAN.

Python:

Calling `h5py.File` with `'w'` for the file access flag will create a new HDF5 file and overwrite an existing file with the same name. "`file`" is the file handle returned from opening the file. When finished with the file, it must be closed. When not specifying property lists, the default property lists are used:

```
import h5py
file = h5py.File ('file.h5', 'w')
file.close ()
```

C:

The H5Fcreate function creates an HDF5 file. H5F_ACC_TRUNC is the file access flag to create a new file and overwrite an existing file with the same name, and H5P_DEFAULT is the value specified to use a default property list.

```
#include "hdf5.h"

int main() {

    hid_t       file_id;
    herr_t      status;

    file_id = H5Fcreate ("file.h5", H5F_ACC_TRUNC,
                         H5P_DEFAULT, H5P_DEFAULT);
    status = H5Fclose (file_id);

}
```

FORTRAN:

Creating an HDF5 file using FORTRAN is similar to C. The h5fcreate_f call creates the HDF5 file and h5fclose_f closes the file. However, the HDF5 FORTRAN interface must also be initialized (h5open_f) and closed (h5close_f).

```
PROGRAM FILEEXAMPLE

USE HDF5
IMPLICIT NONE

CHARACTER(LEN=8), PARAMETER :: filename = "filef.h5" ! File name
INTEGER(HID_T) :: file_id
INTEGER     ::   error

CALL h5open_f (error)
CALL h5fcreate_f (filename, H5F_ACC_TRUNC_F, file_id, error)
CALL h5fclose_f (file_id, error)
CALL h5close_f (error)

END PROGRAM FILEEXAMPLE
```

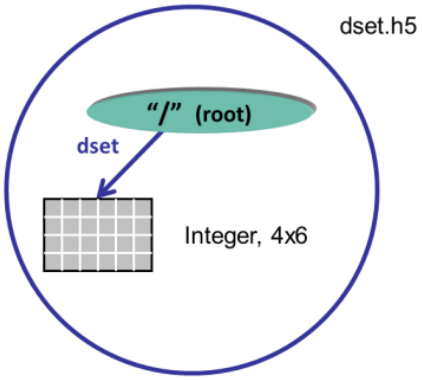
Steps to create a dataset:

As described previously, an HDF5 dataset consists of the raw data, as well as the metadata that describes the data (datatype, spatial information, and properties). To create a dataset you must:

1. Define the dataset characteristics:
   a) Datatype
   b) Dataspace
   c) Properties (or use default)
2. Decide which group to attach the dataset to.
3. Create the dataset.
4. Close the dataset handle from step 3.

Example:

The code excerpts below show the calls that need to be made to create a 4 x 6 integer dataset "`dset`" in a file "`dset.h5`":



Python:

With Python, the creation of the dataspace is included as a parameter in the dataset creation method. Just one call will create a 4 x 6 integer dataset "`dset`". A pre-defined Big Endian 32-bit integer datatype is specified. The `create_dataset` method creates the dataset in the root group (the `file` object). The dataset is close by the Python interface.

```
dataset = file.create_dataset("dset",(4, 6), h5py.h5t.STD_I32BE)
```

C:

To create the same dataset in C, you must specify the dataspace with the `H5Screate_simple` function, create the dataset by calling `H5Dcreate`, and then close the dataspace and dataset with calls to `H5Dclose` and `H5Sclose`. `H5P_DEFAULT` is specified to use a default property list. Note that the file identifier (`file_id`) is passed in as the first parameter to `H5Dcreate`, which creates the dataset in the root group.

```
/* Create the dataspace for the dataset. */
dims[0] = 4;
dims[1] = 6;
dataspace_id = H5Screate_simple(2, dims, NULL);

/* Create the dataset. */
dataset_id = H5Dcreate (file_id, "/dset", H5T_STD_I32BE,
                  dataspace_id, H5P_DEFAULT, H5P_DEFAULT,
                  H5P_DEFAULT);

/* Close the dataset and dataspace */
status = H5Dclose(dataset_id);
status = H5Sclose(dataspace_id);
```

FORTRAN:

Similar to C, in FORTRAN you must call `h5Screate_simple_f` to create the dataspace, `h5dcreate_f` to create the dataset, and then close the dataspace and dataset with `h5dclose_f` and `h5sclose_f`:

```
!
! Create the dataspace.
!
CALL h5screate_simple_f (rank, dims, dspace_id, error)

!
! Create the dataset with default properties.
!
CALL h5dcreate_f(file_id,dsetname,H5T_NATIVE_INTEGER,dspace_id, &
     dset_id, error)

!
! Close the dataset and dataspace
!
CALL h5dclose_f (dset_id, error)
CALL h5sclose_f (dspace_id, error)
```

Writing to or reading from a dataset:

Once you have created or opened a dataset you can write to it:

Python:

```
data = np.zeros((4,6))
for i in range(4):
  for j in range(6):
     data[i][j]= i*6+j+1
dataset[...] = data        ← Write data to dataset
data_read = dataset[...]   ← Read data from dataset
```

C:

`H5S_ALL` is passed in for the memory and file dataspace parameters to indicate that the entire dataspace of the dataset is specified. These two parameters can be modified to allow subsetting of a dataset. The native predefined datatype, `H5T_NATIVE_INT`, is used for reading and writing so that HDF5 will do any necessary integer conversions:

```
status = H5Dwrite (dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
             H5P_DEFAULT, dset_data);

status = H5Dread(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
             H5P_DEFAULT, dset_data);
```

FORTRAN:

Specifying `H5S_ALL` for the memory and file dataspace is not necessary in FORTRAN as it is assumed as the default:

```
!
! Write the dataset.
!
  data_dims(1) = 4
  data_dims(2) = 6
  CALL h5dwrite_f (dset_id, H5T_NATIVE_INTEGER, dset_data, data_dims,
                error)

  !
  ! Read the dataset.
  !
  CALL h5dread_f (dset_id, H5T_NATIVE_INTEGER, data_out, data_dims,
                error)
```

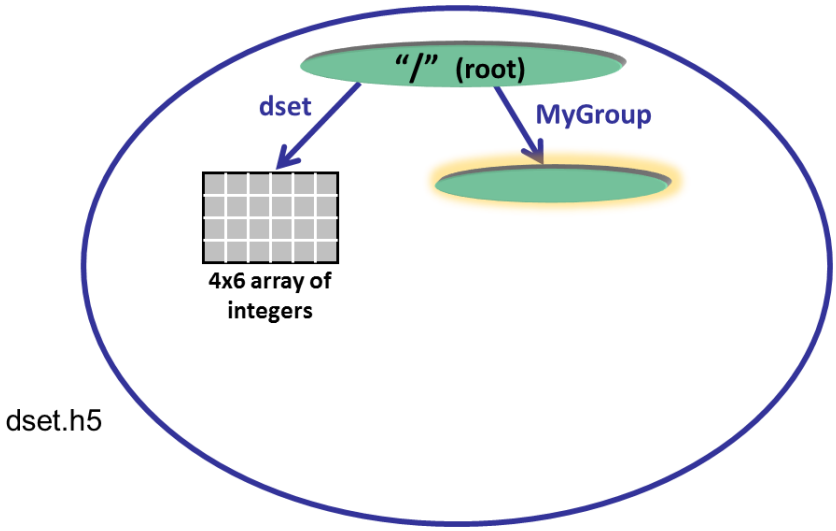
Steps to create a group:

An HDF5 group is a structure containing zero or more HDF5 objects. Before you can create a group you must obtain the location identifier of where the group is to be created. Following are the steps that are required:

1. Decide where to put the group – in the "root group" (or file identifier) or in another group. Open the group if it is not already open.
2. Define properties or use the default.
3. Create the group.
4. Close the group.

Example:

This example illustrates how to create a group `MyGroup` that is attached to the root group. If the file identifier is specified for the location of the group it will be created in the root group.



Python:

The code below opens the dataset `dset.h5` with read/write permission and creates a group `MyGroup` in the root group. Properties are not specified so the defaults are used:

```
import h5py
file = h5py.File('dset.h5', 'r+')
group = file.create_group ('MyGroup')
file.close()
```

C:

To create the group `MyGroup` in the root group, you must call `H5Gcreate`, passing in the file identifier returned from opening or creating the file. The default property lists are specified with `H5P_DEFAULT`. The group is then closed:

```
group_id = H5Gcreate (file_id, "MyGroup", H5P_DEFAULT, H5P_DEFAULT,
                      H5P_DEFAULT);
status = H5Gclose (group_id);
```

FORTRAN:

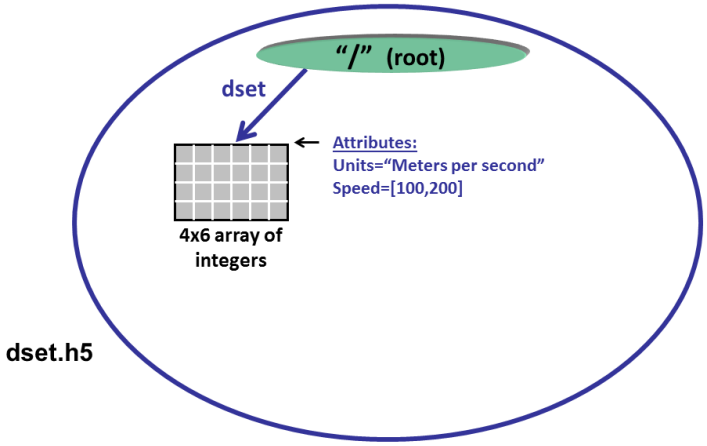
The FORTRAN code looks similar to the C code. Notice that if the properties are not specified, then the default property lists are used:

```
CALL h5gcreate_f (loc_id, name, group_id, error)
CALL h5gclose_f (group_id, error)
```

Steps to create and write to an attribute:

To create an attribute you must open the object that you wish to attach the attribute to. Then you can create, access, and close the attribute as needed:

1. Open the object that you wish to add an attribute to.
2. Create the attribute.
3. Write to the attribute.
4. Close the attribute and the object it is attached to.



Python:

The dataspace, datatype, and data are specified in the call to create an attribute in Python:

```
dataset.attrs["Units"] = "Meters per second"
```
← **Create string**

```
attr_data = np.zeros((2,))
attr_data[0] = 100
attr_data[1] = 200
dataset.attrs.create("Speed", attr_data, (2,), "i")
```
← **Create Integer**

C:

To create an integer attribute in C, you must create the dataspace, create the attribute, write to it and then close it in separate steps:

```
/* Create the data space for the attribute. */
dims = 2;
dataspace_id = H5Screate_simple(1, &dims, NULL);

/* Create a dataset attribute. */
attribute_id = H5Acreate2 (dataset_id, "Units", H5T_STD_I32BE,
                  dataspace_id, H5P_DEFAULT, H5P_DEFAULT);

/* Write the attribute data. */
status = H5Awrite(attribute_id, H5T_NATIVE_INT, attr_data);

/* Close the attribute. */
status = H5Aclose(attribute_id);
```

FORTRAN:

The following code excerpt creates a string attribute in FORTRAN. Note that a derived type is used to create a string of characters:

```
attr_data(1) = "Dataset character attribute"
attr_data(2) = "Some other string here     "
attrlen = 80

CALL h5screate_simple_f(arank, adims, aspace_id, error)
!
! Create datatype for the attribute.
!
CALL h5tcopy_f (H5T_NATIVE_CHARACTER, atype_id, error)
CALL h5tset_size_f (atype_id, attrlen, error)
!
! Create dataset attribute.
!
CALL h5acreate_f (dset_id, aname, atype_id, aspace_id, attr_id,
```

```
                  error)
!
! Write the attribute data.
!
data_dims(1) = 2
CALL h5awrite_f (attr_id, atype_id, attr_data, data_dims, error)
!
! Close the attribute.
!
CALL h5aclose_f (attr_id, error)
```

Subsetting

If specifying `H5S_ALL` for the memory and file dataspace parameters when reading to or writing from a dataset, the entire dataset is selected. To select a subset of a dataset you would modify these two parameters when calling `H5read` and `H5Dwrite`. Subsetting is not covered in this document, but is discussed in detail in the HDF5 Tutorial. See the Reading From or Writing To a Subset of a Dataset tutorial topic for how to select a subset in an HDF5 file.

Compression

If specifying `H5P_DEFAULT` the default properties are used when creating a dataset. By default a dataset is contiguous when created and is uncompressed. To use compression in HDF5 you must modify the dataset creation property list to create a chunked dataset and to use a specified compression method. See the Compressed Datasets tutorial topic.

Discovering the contents of an HDF5 file

The Object (H5O) and Link (H5L) APIs include functions for discovering the contents of an HDF5 file. See this tutorial topic which discusses them.

In the Groups section of the Examples by API pages (accessible from the Examples page), examples are included that use `H5Literate`, `H5OVisit`, and `H5Lvisit`.

Specifically look at: `h5ex_g_iterate.c`, `h5ex_g_traverse.c` and `h5ex_g_visit.c`

References

The Learning the Basics topic in the HDF5 Tutorial covers the information discussed in this document, as well as more in-depth topics such as subsetting and compression.

The example programs used in the tutorial can be accessed separately.

More complex example programs can be found on the Examples page.

The HDF5 Command-line Tools tutorial contains information on `h5dump` and other HDF5 tools.

The `HDFView` tutorial demonstrates how to use `HDFView` to create and access HDF5 files and datasets (without the need to write an application).

There is an HDF Helpdesk where questions can be sent.

There is also an HDF-Forum mailing list that users can join and send questions to. To join the HDF-Forum and view HDF-Forum messages see the HDF Community Support page.

The news mailing list is used by The HDF Group to disseminate information to the HDF community. See the Mailing Lists page for information on the HDF-Forum and news mailing lists.