ESE-RFC XXXX
Category: Draft Community Standard
Updates/Obsoletes : None

Mike Folk, Elena Pourmal

October 2005
HDF5 1.6 standard

HDF5 Data Model, File Format and Library – HDF5 1.6

1 Status of this Memo

This is a description of a Draft ESE Community Standard.

Distribution of this Draft ESE Community Standard is unlimited.

2 Change Explanation

This is a first draft. There are no changes at this point.

3 Copyright Notice

4 Abstract

This document defines HDF5, a data model, file format and I/O library designed for storing, exchanging, managing and archiving complex data including scientific, engineering, and remote sensing data.

Table of Contents

5 Introduction

HDF5 was created to address the data management needs of scientists and engineers working in high performance, data intensive computing environments.

Scientific and engineering applications must deal with increasingly large datasets, a variety of different data types and structures, and many different forms of metadata. This data must be available on a broad range of operating systems and computing environments. It must be easy to move data from place to place, and to share the data among widely differing applications.

Applications and tools must be available for creating and accessing the data, and the basic software for storing and accessing the data must work in as many different computing environments as possible.

Because the amount of scientific data is increasing rapidly, efficient storage is critical. Finally, many applications require extremely efficient and flexible I/O in order to deal with the volume of the data being processed, the frequent need to access small subsets from large data sets, and the variety of architectures and file systems in use.

5.1 What is HDF5?

HDF5 has three components: (1) a general purpose data model, (2) an I/O library and (3) a file format.

The HDF5 *data model* provides structures and operations to allow creation, storage, and access to almost any kind of scientific data structure or collection of structures. In addition the HDF5 file object, the data model includes two primary objects (datasets and groups), a number of supporting objects (e.g. attributes, datatypes), and metadata describing how HDF5 files and objects are to be organized and accessed.

The HDF5 *file format* describes how the HDF5 data structures are represented in storage, in memory, or on other media. Because HDF5 is designed for managing large data objects and complex heterogeneous collections easily and efficiently, the HDF5 format allows for alternate representations of many objects. The format is self-describing in the sense that the structures of HDF5 objects are describe within the file.

The HDF5 *I/O library* implements the data model in a number of programming languages, including C, Fortran, C++, and Java. These API's are designed for flexibility – they give applications full access to available HDF5 storage structures, and provide features for tuning applications for particular platforms, storage requirements, or I/O access patterns.

Complementing these three technical components are the approaches of the HDF5 project to intellectual property and community standards.

The HDF5 library, which is owned by the University of Illinois, is open source, and the HDF5 copyright[1] allows it to be used at no cost by all applications, including commercial applications. The HDF5 project and its sponsors work closely with vendors, as well as non-commercial applications, to enable their products to support HDF5 effectively and to make sure that HDF5 meets the demands of these products for quality and performance.

As for community standards, the HDF5 project and its sponsors dedicate significant resources to developing, supporting, and enforcing standard uses of HDF5. Adherence to standards makes it possible to share data easily, and also to build and share tools for accessing and analyzing data stored in HDF5. Standardization activities include establishing conventions for the use of HDF5 for particular applications. For example, HDF-EOS defines a data model built for earth science data, and the HDF-EOS API implements that data model. As another example, the HDF5 project defines standard ways to store raster images, tables, and other complex objects in HDF5, and provides APIs to encourage adherence to these standards.

5.2 Motivation for Proposing Standardization

HDF5 is the underlying format for HDF-EOS 5. HDF-EOS is the standard format and I/O library for the Earth Observing System (EOS) Data and Information System (EOSDIS). EOSDIS is the data system supporting a coordinated series of polar-orbiting and low inclination satellites for long-term global observations of the land surface, biosphere, solid Earth, atmosphere, and oceans. HDF-EOS 5 is the standard for the Aura mission, which is the final EOS mission.

HDF5 is also to be the distribution format for the National Polar Orbiting Environmental satellite system (NPOESS), a satellite system to be used to monitor global environmental conditions, and collect and disseminate data related to weather, atmosphere, oceans, land and near-space environment.

EOS data stored in HDF-EOS 5 and NPOESS data to be stored in HDF5 are of fundamental importance to current and future research on global climate change, and scores of other applications of national and international importance.

ESE standardization of HDF5 will help to accelerate its adoption among the EOS and NPOESS communities, and many others as well, both through an increase in the number of developers writing to the specification and using the I/O library, and through an increase in the number of those providing their data in HDF5.

ESE standardization will also validate HDF5 to vendors of software that is important to users of EOS and NPOESS data, increasing the likelihood that these vendors will support HDF5 in their products.

ESE standardization will also validate HDF5 to government agencies and other organizations that have considered standardizing on HDF5 but have been reluctant to do so because of the lack

[1] See Appendix E for HDF5 copyright.

of standardization. This will increase the likelihood of adoption of HDF5 by these organizations, in turn broadening the support for HDF5. Greater support and adoption of HDF5 would very likely result in improved usability of HDF5, more software for working with data in HDF5, and more useful data stored in HDF5.

Finally, ESE standardization will pave the way for broader standardization of HDF5, both at the national (ANSI) and international (ISO) levels. If this happens, the results of standardization described above will extend even more broadly, across a broad range of science and engineering domains. Many of these results will return dividends for the ESE community as well.

6 HDF5 Data Model

6.1 Introduction

The HDF5 data model describes the organization and structure of data stored in a computer system, and operations that can be performed on data.

An application program that uses HDF5 for storing and organizing data, maps its data structures to the HDF5 objects defined by the data model. The mapping is independent of storage medium, computational system and computational environment.

A potential complexity of an Application Program's objects, and the richness of data structures and data types used to describe the objects, may present a challenge for any data model. The HDF5 Data Model has proved to be adequate for describing data objects used in *scientific* and *engineering* applications. It uses *arrays of structures* to describe *object's data* and *metadata*, and *grouping mechanism* to describe *relationships* between the objects.

This chapter defines HDF5 objects, their structural metadata and operations on the HDF5 objects.

6.2 Overview of HDF5 Objects

The HDF5 Data Model defines seven objects: *File*, *Group, Link*, *Dataset, Datatype, Dataspace*, and *Attribute*.

Group and *Dataset* objects are stored in a *File.*

Dataspace and *Datatype* objects are parts of a *Dataset* definition.

A Datatype can be also stored in a File as a "standalone" object shared by many *Datasets.*

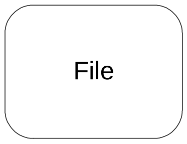
The HDF5 objects in a *File* are called *persistent* objects. Each *persistent* object has one or more *Link* objects associated with it. *Links* provide a mechanism for locating *persistent* objects in a File and for sharing *persistent* objects between *Group*s.

An Attribute is a means of attaching *user-defined* metadata to a *persistent* object.

Sections 6.3-6.9 define each of HDF5 objects.

6.3 File Object

6.3.1 Definition, organization and storage layouts.



The HDF5 Data Model defines a *File* as a container for the HDF5 *persistent* objects.

Objects in a *File* are organized as a directed graph with a designated entry node. Every node is represented by an HDF5 *persistent* object. Node that has outgoing edges is always a *Group*. Entry node is called a *Root Group*. Node that has incoming edges is any of the HDF5 *persistent* objects.

Each edge is represented by a *Link* object.

An HDF5 *persistent* object in a *File* can be identified by any of the paths that connect the object with its ancestor nodes. The path that connects *a persistent* object with a *Root Group* is called an *absolute path*. The path that connects *a persistent* object with any other parent *Group* is called a *local path*.

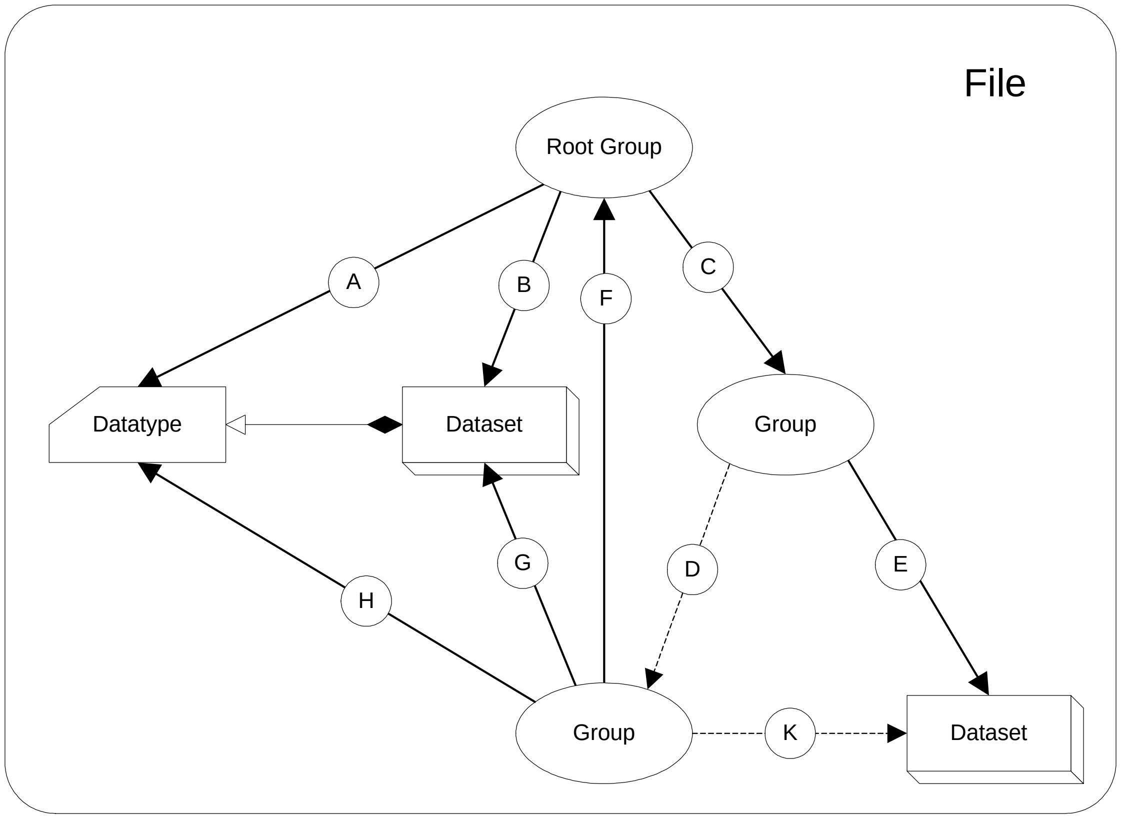


Figure 1: The HDF5 File serves as a container for HDF5 objects.

The HDF5 Data model does not define *File*'s physical storage layout on a computer system. The HDF5 file format specification defines the types of physical storage layout that are valid. The format allows users to define alternate forms of physical storage beyond those described in the specification. For instance, a *File* can be stored as a single file or as a set of files. It may also exist in Application Program's memory only. HDF5 1.6 has the following predefined storage layouts for a *File* object:

Storage layout	***Description***
Single file	A single file on a file system
Family of files	The logical space of HDF5 file is partitioned among a set of the single files that have the same size. This storage layout makes it possible to store HDF5 files larger than 2GB on a file system that doesn't support large files.
Multi file	A set of single files that represent an HDF5 file. Each of the files stores one or more predefined types of internal structures (global and local heaps, B-trees, objects headers, raw data) used to describe persistent objects in an HDF5 file.
Core (memory) file	An HDF5 file stored in memory of an Application Program.

6.3.2 Structural metadata

File structural metadata is described in **Appendix A**,"HDF5 File Format Specification', section II "File Metadata".

Structural metadata sets *File* object's properties and cannot be modified after *File* is created.

Structural metadata contains version information and parameters of internal data structures used for storing HDF5 objects. It also contains information about how a *File* is accessed for different File's storage layouts on a computer system (drivers' information). For more information on *File* access drivers see section **8.3.3** Virtual File Driver.

HDF5 1.6 defines the following elements of *File* structural metadata that can be defined by an Application Program:

Name	***Description***
Size of offsets	Number of bytes used to store addresses in an HDF5 file
Size of lengths	Number of bytes used to store the size of an HDF5 object
Group leaf node K	Each leaf node of a Group B-tree will have at least K entries but not more than 2K entries. Single leaf node may have fewer entries.

Name	***Description***
Group internal node K	Each internal node of a Group B-tree will have at least K entries but not more than 2K entries. Single internal node may have fewer entries.
Index Storage Internal Node K	Each internal node of an indexed storage. B-tree will have at least K entries but not more than 2K entries. Single internal node may have fewer entries.
Driver information	Information about a file driver needed to reopen an HDF5 file.

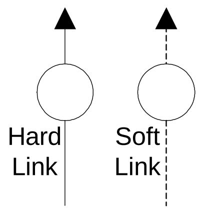
6.3.3 Operations on File

The following operation can be performed on *File*

Name	***Description***
Create	Creates a *File* with defined properties
Open	Opens an existing *File*
Reopen	Reopens an already open *File*
Close	Closes a *File*
Mount/Unmount	Includes/removes *File* graph structure in another *File's* graph structure
Flush	Flushes the contents of a *File*
Query if HDF	Determines whether a file is an *HDF5 File*.

6.4 Link Object

6.4.1 Definition



A Link represents an edge in a *File* graph structure and has a (*Name*, *Value*) pair associated with it. *Name* is defined as ASCII null terminated string and is stored in a *Group* structural metadata (see section 6.5.2) *Name* is used as key for the object search in a *File*. *Value* is an address of an object pointed by *Link*.

There are two types of *Link* objects in the HDF5 Data Model: a *Soft Link* and a *Hard Link. Hard Link* always points to a *persistent* object. *Soft Link* can point to no object. For *Soft Link* a *Value* may be missing. Any *persistent* object in a *File* has at least one *Hard Link* pointing to it.

Link is created when a member is added to a Group and removed when a member is removed from a Group.

6.4.2 Operations on Links

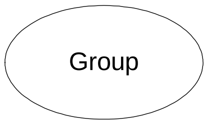
The following operations can be performed on a *Link* object:

Name	***Description***
Insert	Inserts *Link* to a *Group* (adds new member to a *Group*)
Remove	Removes *Link* from a *Group* (removes a member from a *Group*)

Note: When an object is created in a *Group*, *Hard Link* is inserted to the Group's structural metadata, therefore there is no explicit *create* operation for *Hard Links*.

6.5 Group Object

6.5.1 Definition



Group object is a container within a *File* that has zero or more HDF5 *persistent* objects including the *Group* itself. Each *persistent* object is a member of at least one *Group* in a *File* with exception of the *Root Group* that may not be a member of any *Group*. An HDF5 *persistent* object may be included in multiple *Groups.* Such object is called a *shared* object.

Group membership is implemented via the *Link* object. A *Hard Link* object is added to a *Group* every time a *persistent* object is created. *Hard* or *Soft Link* object can also be added to a *Group* to include already existing *persistent* object. When *Link* is removed from a *Group*, the corresponding *persistent* object pointed by *Link* is excluded from the *Group* members.

6.5.2 Group structural metadata

Group structural metadata is described in Appendix A "HDF5 File Format Specification", section II "Level 1- File Infrastructure, Level 1A, 1B".

The following structural metadata can be defined by an Application Program:

Name	***Description***
Data Segment Size of Local Heap	Total amount of space allocated for *Local Heap* data

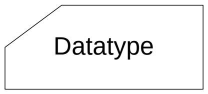
6.5.3 Operations on Group

The following operation can be performed on *Group*:

Name	***Description***
Create	Creates a *Group* with defined properties
Open	Opens an existing *Group*
Close	Closes a *Group*
Iterate	Iterates through group members
Get information about members	Gets number of members, member's object type and name of a Link pointed to a member

6.6 Datatype Object

6.6.1 Definition and types of Datatype Objects



A Datatype object describes an individual data element of a *Dataset* or an *Attribute*.

The structural metadata of a *Datatype* object defines a layout of a single data element and provides complete information for data conversion to and from a *Datatype*.

The HDF5 Data Model defines two types of the *Datatype* objects: *persistent Datatype* and *transient Datatype*.

A Persistent Datatype is stored in a File as a separate object. It always has a Hard Link associated with it. *Persistent Datatype* can be used to define multiple *Attributes* and/or *Datasets.*

A Transient Datatype is used to define a *Dataset* or an *Attribute*. When a *Dataset* or an *Attribute* object is created, a *transient Datatype's* structural metadata becomes a part of the *Dataset's* or *Attribute's* structural metadata. It can be accessed only by accessing the structural metadata of that object.

Both *persistent* and *transient Datatypes* fall into two categories: *atomic* and *composite*. Those categories describe if the element of a particular *Datatype* is treated as an atomic unit during I/O operations on a *Dataset* or an *Attribute*, or if it is composed of other atomic units that can be accessed independently.

The Current HDF5 Data Model defines only one composite *Datatype – Compound Datatype*. *Compound Datatypes* are similar to C structures or Fortran 95 derived datatypes and may have multiple members of any types.

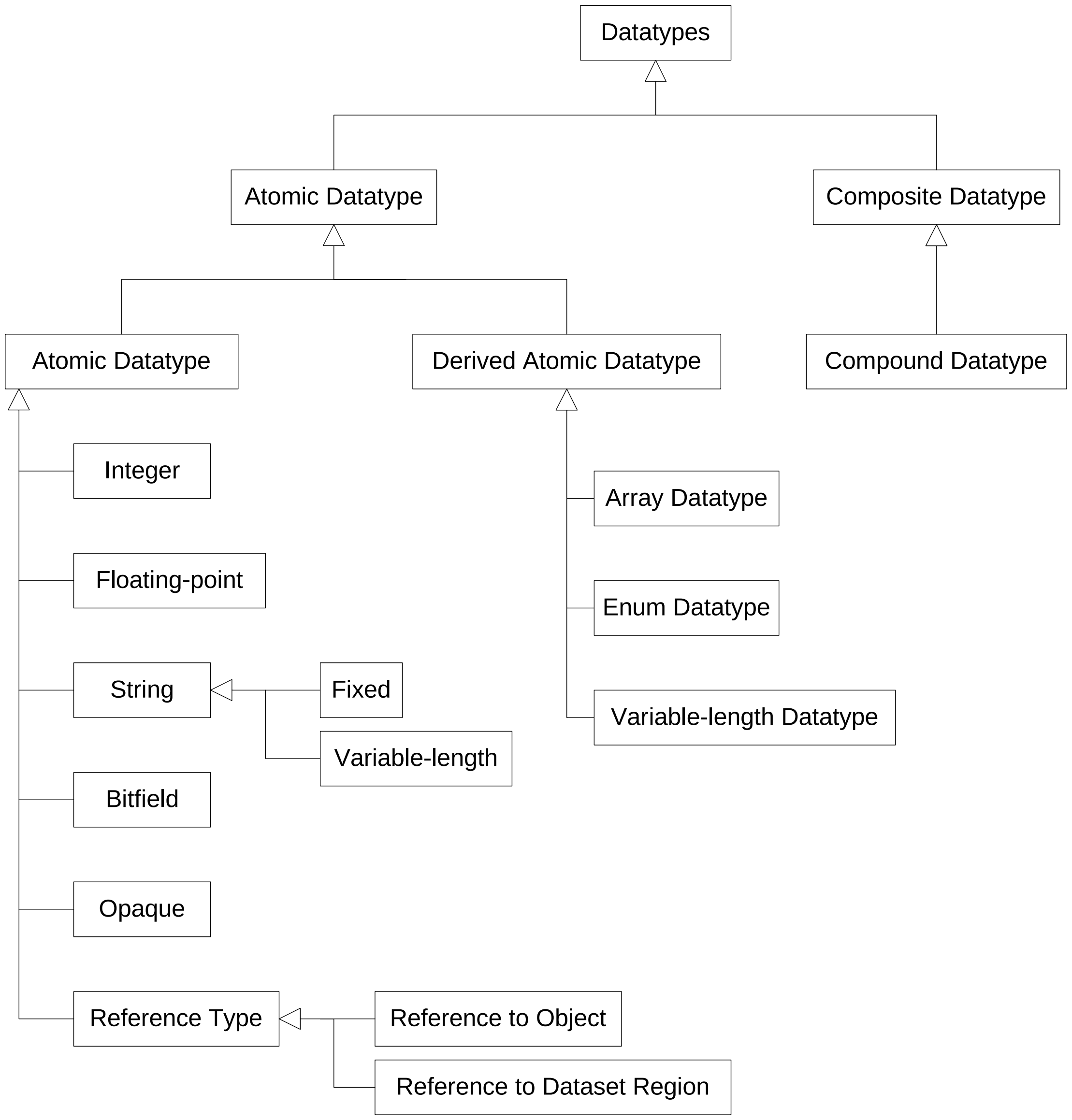


Figure 2: The hierarchy of atomic and composite HDF5 datatypes.

Member of a *Compound Datatype* can be any of *atomic Datatypes* described in the section 6.9.4 or of a *Compound Datatype*. There is no restriction on a number of members or depth of the structure.

Both *persistent* and *transient atomic Datatype* objects are created from the pre-defined *atomic Datatypes* by copying and modifying them. *Persisten*t and *transient composite Datatype* objects are created using *atomic* or *composite Datatypes*. Pre-defined *Datatypes* of each class are described in the section 6.9.4

Datatype structural metadata and operations on a *Datatype* depend on whether a *Datatype* is *persistent* or *transient*, and if it is *atomic* or *composite*. The next sections describe structural metatdata of the HDF5 *Datatypes* and operations on them.

6.6.2 Structural metadata

Structural metadata of Datatype object is described in Appendix A "HDF5 File Format Specification", section IV "Disk Format Level 2- Data Objects". It defines properties of a *Datatype*. Since properties depend on the type of *Datatype*, they will be discussed under the corresponding subsections of 6.6.4

6.6.3 Common operations on Datatype object

The following operations can be performed on a *Datatype* object; operations marked with * are the operations that are not allowed on pre-defined *atomic* types:

Name	***Description***
*Create**	Creates a *Dadatype* of particular class
*Open**	Opens a *persistent Datatype*
Copy	Copies a *Datatype*
*Close**	Closes a *Datatype*
Commit	Stores a *Datatype* in a *File*
Query if committed	Finds if a *Datatype* is stored in a *File*
Compare	Finds if two *Datatypes* are the same
Get/set size	Get/sets the size of a Datatype
Get class	Finds a class to which *Datatype* belongs

6.6.4 Atomic Datatypes

Atomic Datatypes describe the smallest possible raw data element of *Dataset* or *Attribute* that I/O operations can be performed on. The HDF5 Data Model defines a rich collection of *atomic Datatypes* that include integer, floating-point, string, enumeration, bitfield datatypes along with the HDF5 specific datatypes such as array, reference, opaque, and variable length datatypes. The next sections talk about *basic* and *derived atomic Datatypes*.

6.6.4.1 Basic Atomic Datatypes

Basic atomic Datatypes are used in definitions of *derived* and *compound Datatypes*. *Basic Datatypes* include integer, floating-point, string, opaque, bitfield, and reference datatypes.

6.6.4.1.1 Integer type

Integer Datatype describes integer number formats. It is created by copying pre-defined integer type and setting up size, sign, and order (endianess) properties.

Operations on *Integer Datatype* include:

Name	***Description***
See section 6.6.3	All operations described in 6.6.3 can be performed
Set/get sign	Defines if integer is signed or not
Set/get order	Defines byte order in which integer is stored
Set/get precision	Identifies a number of significant bits
Set/get offset	Identifies a location of significant bits
Conversion	Converts *Integer Datatype* to *Integer* or *Floating-point Datatype*

Pre-defined *Integer Datatypes* describe most commonly used integer type formats. There are two types of pre-defined integer datatypes: standard and native.

Description of *pre-defined standard Integer Datatype* is computer system independent and is used in the definitions of the persistent objects in the file.

Description of *pre-defined native Integer Datatype* depends on a computer system and on a computational environment (compiler settings). It is provided for the convenience of Application Program to avoid discovery of integer types' properties needed for describing *Integer Datatype* in a File. Each *pre-defined native Integer Dataype* has a corresponding *pre-defined standard Integer Datatype* that is used for such description.

Pre-defined standard Integer Datatypes have names associated with them of the form H5T_STD_<base>, where <base> is a letter "I" for signed integers, and "U" for unsigned integers, followed by one of the numbers 8,16,32, or 64 that correspond to the number of bits representing the integer type, followed by strings "LE" or "BE" for little-endian or big-endian byte order.

Examples:

H5T_SDT_I32LE represents 32-bit, little-endian, signed two's complement integer

H5T_SDT_U64BE represents 64-bit, big-endian, unsigned integer.

Pre-defined native Integer Datatypes have names associated with them of the forms H5T_NATIVE_<language_type>, where <language_type> is a string that corresponds to a computer language specific datatype definition.

Examples:

H5T_NATIVE_INT corresponds to C “int” type

H5T_NATIVE_ULLONG corresponds to C “unsigned long long” type

H5T_NATIVE_INTEGER corresponds to Fortran “INTEGER” type

For complete list of *pre-defined Integer Datatypes* see Appendix **B**, RM, Predefined datatypes)

6.6.4.1.2 Floating-point type

Floating-point Datatype describes the floating-point type formats that satisfy the following conditions:

- Bits of the exponent are contiguous and stored as biased positive number
- Bits of the mantissa are contiguous and stored as a positive magnitude
- Sign bit exists which is set for negative values

Floating-point Datatype is created by copying pre-defined atomic *Floating-point Datatype* and modifying its properties.

Operations on Floating-point Datatype include:

Name	***Description***
See section 6.6.3	All operations described in 6.6.3 can be performed
Set/get sign, exponent, and mantissa	Defines position of sign, exponent and mantissa, and their sizes
Set/get bias	Defines exponent bias
Set/get normalization methods for mantissa	Determines the normalization method of the mantissa
Set/get padding	Identifies padding for unused bits in datum
Set/get order	Defines byte order in which floating-point number is stored
Conversion	Converts *Floating-point Datatype* to *Integer* or *Floating-point Datatype*

Pre-defined *Floating-point Datatypes* describe most commonly used floating-point type formats. There are two types of pre-defined *Floating-point Datatypes*: standard and native.

Description of pre-defined standard *Floating-point Datatype* is computer system independent and is used in the definitions of persistent objects in the file.

Description of pre-defined native *Floating-point Datatype* depends on a computer system and on a computational environment (compiler settings). It is provided for the convenience of Application Program to avoid discovery of floating-point types' properties needed for describing a *Floating-point Datatype* in a File. Each pre-defined native *Floating-point Dataype* has a corresponding pre-defined standard *Floating-point Datatype* that is used for such description.

Pre-defined standard *Floating-point Datatypes* have names associated with them of the form H5T_IEEE_<base>, where <base> is a letter "F" indicating floating-point datatype, followed by one of the numbers 8,16,32, or 64 that correspond to the number of bits representing the floating-point datatype, followed by strings "LE" or "BE" for little-endian or big-endian byte order.

Examples:

H5T_IEEE_F32LE represents 32-bit, little-endian, IEEE floating-point

H5T_IEEE_F64BE represents 64-bit, big-endian, IEEE floating-point

Pre-defined native *Floating-point Datatypes* have names associated with them of the forms H5T_NATIVE_<language_type>, where <language_type> is a string that corresponds to a computer language specific datatype definition.

Examples:

H5T_NATIVE_FLOAT corresponds to C "float" type

H5T_NATIVE_DOUBLE corresponds to C "double" type

H5T_NATIVE_REAL corresponds to Fortran "REAL" type

For complete list of pre-defined *Floating-point Datatypes* see Appendix B, RM, Predefined datatypes.

6.6.4.1.3 String type

The HDF5 Data Model provides *String Datatype* object. *String Datatype* is a sequence of characters that is interpreted as an ASCII text. It is created by copying pre-defined *String Datatype* object and setting its size to a fixed length or to a variable length.

Pre-defined *String Datatype* has a language specific name associated with it: H5T_C_S1 (C String objects corresponds to C null terminated string) and H5T_FORTARN_S1 (Fortran space padded string).

Operations on *String Datatype* include:

Name	***Description***
See section 6.6.3	All operations described in 6.6.3 can be performed
Set/get a type of string storage	Sets/gets a method of string storage to accommodate a language specific storage options such as padding and termination character.

6.6.4.1.4 Bitfield type

The HDF5 Data Model defines *Bitfield Datatype* as a sequence of bits packed in one of the *Integer Datatypes*. *Bitfield Datatype* is created by copying a pre-defined *Bitfield Datatype* and setting precision, offset, and padding.

Pre-defined *Bitfield Datatypes* have a name associated with them of the form H5T_<arch>_B<8,16,32,64>, where <arch> is “NATIVE” or “SDT” string.

Example:

- H5T_NATIVE_B64 – native 8-byte bit field
- H5T_SDT_B16 – standard 2-byte bit field.

Operations on *Bitfield Datatype* include:

Name	***Description***
See section 6.6.3	All operations described in 6.6.3 can be performed
Set/get padding	Identifies padding for unused bits in datum
Set/get precision	Identifies a number of significant bits
Set/get offset	Identifies a location of significant bits
Conversion	Converts *Bitfield Datatype* to another *Bitfiled Datatype*

6.6.4.1.5 Opaque type

The HDF5 Data Model provides *Opaque Datatype* for describing data that cannot be otherwise described by HDF5. Element of an *Opaque Datatype* is treaded as a blob and is not interpreted by the HDF5 library.

Opaque Datatype is identified by its size and a tag (ASCII string). Operations on the Opaque datatype include:

Name	***Description***
See section 6.6.3	All operations described in 6.6.3 can be performed
Set/get tag	Defines the tag (ASCII string) for *Opaque Datatype* identification

6.6.4.1.6 Reference type

The HDF5 Data Model defines two types of HDF5 “pointers” called “*Object Reference*” and “*Dataset Region Reference*”.

6.6.4.1.6.1 Object reference type

Element of “Object Reference” datatype points to a *persistent* HDF5 Object.

“*Object Reference*” Datatype is created by copying a pre-defined “*Object Reference*” Datatype that has a name of the form H5T_STD_REF_OBJ.

Operations on “Object Reference” datatype include all operations listed in 6.6.3

6.6.4.1.6.2 Datset region reference type

Element of “*Datset Region Reference*” datatype points to a selected region of a *Dataset*.

“*Dataset Region Reference*” Datatype is created by copying a pre-defined *“Dataset Region Reference” Datatype* that has a name of the form H5T_STD_REF_DSETREG.

Operations on “*Object Reference*” *Datatype* include all operations listed in 6.8.3

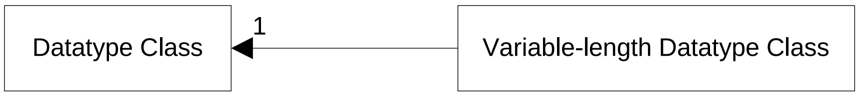
6.6.4.2 Derived atomic types

Derived atomic Datatypes are built from *basic atomic Datatypes* or *composite atomic Datatypes*.

Element of *derived atomic Datatype* is treated as one unit during I/O operations. The HDF5 Data Model defines three *derived atomic Datatypes*: *Variable length Datatype, Enumeration Datatype* and *Array Datatype* that are described in the following sections.

6.6.4.2.1 Variable length type

The HDF5 Data Model defines *Variable length Datatype* for describing data elements that are one dimensional arrays of some base type element. Size of arrays may differ from element to element. *Datatype* of a base type element can be of any HDF5 atomic or composite types including *Variable length Datatypes*.



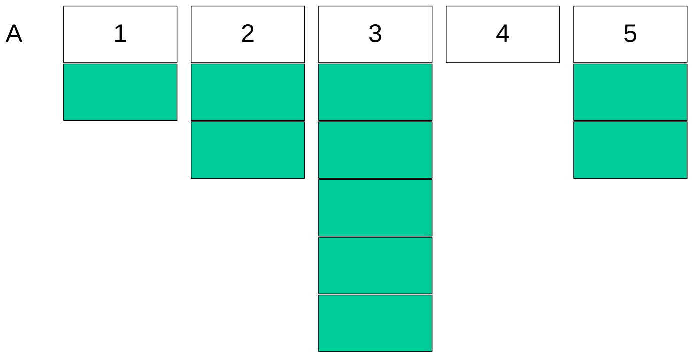


Figure 3: Data element of a dataset with a variable-length datatype can be of a different size.

Operations on *Variable length Datatype* include:

Name	***Description***
See section 6.6.3	All operations described in 6.6.3 can be performed
Get datatype of base element	Discovers a Datatype of a base element

6.6.4.2.2 Enumeration type

The HDF5 Data Model defines *Enumeration Datatype* as a one-to-one mapping between a set of *Symbols* (ASCII character strings) and a set of *Values* that have *Integer Datatype*, i.e. as a set of (*Symbol*, *Value*) pairs.

Since *Enumeration Datatype* is derived from *Integer Datatype*, operations on *Enumeration Datatype* include:

Name	***Description***
See section 6.6.4.1.1	All operations described in 6.6.4.1.1 can be performed
Get number of mapping pairs	Gets the numbers of member pairs in a *Enumeration Datatype*
Get Value of a Symbol	Gets numeric value for a symbol name

Name	***Description***
Get Symbol for a Value	Gets a symbol name for a numeric value
Conversion	Converts *Enumaeration Datatype* to another *Enumeration Datatype*

6.6.4.2.3 Array type

The HDF5 Datatype Model provides *Array Datatype* for describing elements that are homogeneous multi-dimensional arrays of the same base datatype. Base datatype may be of any HDF5 datatypes including *Array Datatype*.

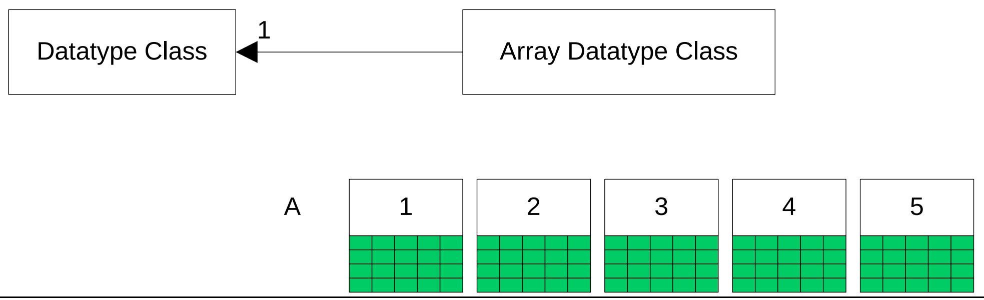


Figure 4: In an HDF5 dataset with an array datatype, each data element is itself an array.

The following operations are allowed on Array datatype:

Name	***Description***
See section 6.6.3	All operations described in 6.6.3 can be performed
Get datatype of base element	Discovers a Datatype of a base element
Get number of dimensions	Gets the number of array dimensions
Get sizes of dimensions	Gets sizes of array dimensions
Conversion	Converts *Array Datatype* to another *Array* Datatype if Datatype of base elemenst allows conversion

6.6.5 Composite datatype

Current version of the HDF5 Data Model defines only one type of composite datatype – *Compound Datatype*.

6.6.5.1 Compound datatype

A Compound Datatype is a datatype that combines one or more datatypes, called *fields*, into a more complex datatype. A *Compound Datatype* is similar to a C structure or Fortran derived datatype. The *Datatype* of each field can be of any HDF5 *Datatype*, including a *Compound Datatype*. Each field of a *Compound Datatype* has a name (an ASCII string) and byte offset indicating its position within the *Compound Datatype*.

A Compound Datatype is created by defining the total size of the *Datatype* and then adding its members.

I/O operations can be performed on a selected member of each element of a *Compound Datatype*. This property distinguishes *Compound Datatypes* from any subtypes of *atomic Datatype*.

Operations on Compound Datatypes include:

Name	***Description***
See section 6.6.3	All operations described in 6.6.3 can be performed
Get number of members	Gets number of members of a *Compound Datatype*
Get member's info	Gets offset, Datatype and a name of a *Compound Datatype* member
Conversion	Converts *Compound Datatype* to another *Compound Datatyp*e if *Datatype* of members allows conversion

6.7 Dataspace Object

6.7.1 Definition

A Dataspace object describes the logical spatial layout of raw data associated with a *Dataset* or *Attribute* object.

HDF5 1.6 supports *Simple Dataspace* objects: a point and a multi-dimensional array

Application Program uses *Dataspace* object to create *Datasets* and/or *Attributes,* and to define the elements that participate in I/O operation when data is moved between Application Program's memory and *File*.

6.7.2 Dataspace structural metadata

Structural metadata of *Simple Dataspace* is described in Appendix A, "HDF5 File Format Specification", section IV, "Disk Format Level 2- Data Objects". The following structural metadata information is provided by an Application Program:

Name	***Description***
Rank N	Number of dimensions of a *Dataset*
Sizes of dimensions 0 to N-1	Current and maximum sizes for each of N dimensions

Current dimension sizes define an *extent* of *Dataset.*

6.7.3 Operations on Dataspace

The following operations can be performed on *Dataspace*:

Name	***Description***
Create	Creates a Dataspace
Set dimensions	Sets sizes of current and maximum dimensions
Close	Close a *Dataspace*
Copy	Creates a copy of a *Dataspace*
Get rank and dimensions	Gets *Dataspace* rank and sizes of current and maximum dimensions
Set/get selection	Defines points in a Dataspace for partial I/O (see section 8.3.1)

6.8 Dataset Object

6.8.1 Definition

The HDF5 Data Model defines *Dataset* as a multidimensional array of elements along with the structural metadata that includes the descriptions of elements' *Datatype*, spatial information about an array (*Dataspace*) and storage properties (how actual raw data is stored in a *File*).

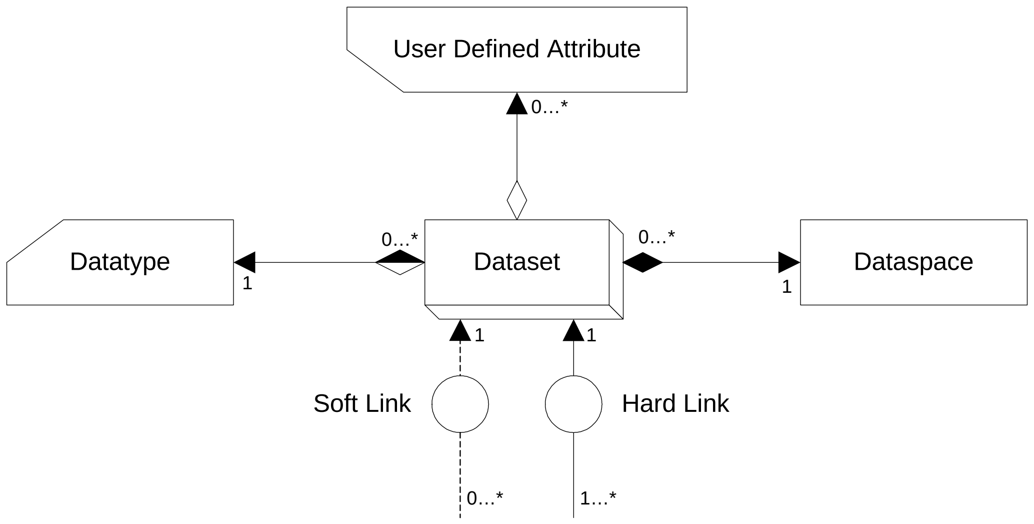


Figure 5: A dataset has a datatype and dataspace, optional attributes, and can be linked into the file hierarchy with both soft and hard links.

6.8.2 Structural metadata

Dataset's structural metadata is described Appendix Appendix A, "HDF5 File Format Specification", section IV "Disk Format Level 2- Data Objects". Structural metadata that is defined by and Application Program includes:

Name	***Description***
Dataspace	Describes spatial information about a *Dataset*
Datatype	*Datatype* of of a *Datatset* element
Fill value	Value returned to an Application Program for uninitialized data
Data storage layout	How raw data is stored in a *File*
o *Compact*	Raw data is stored in an object header
o *Contiguous*	Raw data is stored as a contiguous blob

o *Chunked* o *External*	Raw data is stored in chunks Raw data is stored in an external file
Filter Pipeline	Description of transformations to be applied to the data stream during I/O operation
User-defined metadata	Atributes attached to a *Dataset*

6.8.3 User-defined metadata (Attribute)

Dataset may have Application defined metadata (*Attribute*). *Attribute* is a part of the Dataset structural metadata. For more information on the *Attribute* object see section 6.9

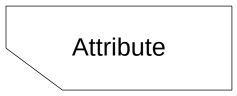
6.8.4 Operations on Dataset Object

Operation s on the Dataset object includes:

Name	***Description***
Create	Creates a *Dataset* with defined properties
Open	Opens an existing *Dataset*
Close	Closes a *Dataset*
Iterate	Iterate over elements of a *Dataset*
Read/Write	Reads/writes raw data to a *File*
Extend	Increases current dimensions' sizes
Query structural metadata	Gets *Datatypes*, *Dataaspace*, storage layout and filter pipeline
Fill	Fills an initialized data with a fill value

6.9 Attribute Object

6.9.1 Definition



The HDF5 Data Model defines *Attribute* as a user-defined metadata. It comes in the form of name-value pairs, where a name is an ASCII string, and value is a scalar or a multi-dimensional array of elements.

Attribute is a part of a *Group*, *Dataset* or *persistent Datatype* definition.

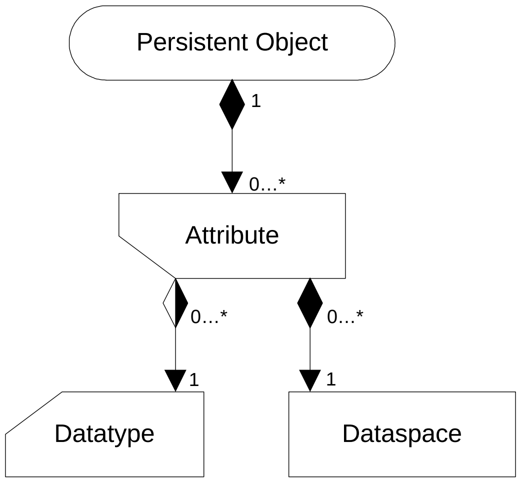


Figure 6: An attribute is attached to a persistent object.

6.9.2 Structural metadata

Attribute structural metadata is described in Appendix Appendix A, “HDF5 File Format Specification”, section IV “Disk Format Level 2- Data Objects”. Structural metadata defined by an Application Program includes:

Name	***Description***
Name	*Attribute’s* name (an ASCII string)
Datatype	*Datatype* of *Attribute’s* data element
Dataspace	*Dataspace* of *Attribute’s* data

6.9.3 Operations on Attribute Object

The following operations are defined on Attribute:

Name	***Description***
Create	Creates an *Attribute* with defined properties and attaches it to a persistent object
Open	Opens an existing *Attribute*
Close	Closes an *Attribute*
Iterate	Iterate over elements of an *Attribute*
Delete	Deletes an *Attribute*
Query structural metadata	Finds name, *Datatype*, and *Dataspace of* an *Attribute*
Read/Write	Reads/writes raw data to a *File*

7 HDF5 File Format

This section gives a brief introduction to the HDF5 File Format. The complete description can be found in Appendix A, "HDF5 File Format Specification."

7.1 HDF5 File Format internal structures

The HDF5 File Format defines the low-level objects in terms of a sequence of bytes. The HDF5 persistent objects are described in terms of the low-level objects, thus creating a mapping from the HDF5 data model to the set of byte sequences (the HDF5 logical space).

For each HDF5 File, the layout of the byte sequences on the storage media depends on the File driver used to create the File, and on the structural metadata of the persistent objects stored in the File. In other words, the HDF5 File Format doesn't define how HDF5 logical space is mapped to the physical storage; this is done by the HDF5 Library described in section 8.

The table below defines the low-level objects. For a full description of the byte sequences corresponding to each object see Appendix A, "HDF5 File Format Specification."

Name	***Description***
Super block	Contains the structural metadata about the File; for byte sequence see section II
B-tree node	Implements B-link trees used to store persistent objects that can grow; for byte sequence see section III
Global Heap	Stores information about an object that cannot be stored in the fixed size object header message (see below). The information is usually shared between several objects in the file. For byte sequence see section III.
Local Heap	Stores information about an object that cannot be stored in the object header fixed size message (see below); for byte sequence see section III.
Object header	Information needed to identify an object and to interpret its raw data and metadata. Each object header is a set of object header messages. Byte sequence corresponding to each header message is given in section IV.

7.2 Mapping between HDF5 Objects and File Format low-level objects

Sections III and IV of the “HDF5 File Format Specification” describe how HDF5 persistent objects, groups, datasets and datatypes are represented by B-trees, global and local heaps, and object headers.

A group object is represented by an object header that contains a message that points to a local heap and to a B-tree which points to the object headers of the group members.

A dataset is represented by an object header that contains messages that describe the datatype, dataspace, layout and other structural metadata, and a message that points to either a raw data chunk, or to a B-tree that points to raw data chunks.

A datatype is represented by an object header that contains a datatype message.

While object headers, heaps, and B-tree nodes are represented by contiguous byte sequences, the complete description of a persistent object may not be a contiguous set of byte sequences. Therefore, in general, there is no way to discover an object in an HDF5 file by specifying a file offset and a size of the object; a special library is needed.

Section 8 describes the HDF5 Library and the set of application programming interfaces that provide access to an HDF5 file and objects stored in the file.

8 HDF5 I/O Library

The HDF5 Library implements the HDF5 Data Model and storage model according to the specifications given in sections 6 and 7 and in Appendix A. The next sections introduce the HDF5 programming model and the HDF5 application programming interfaces (APIs). For a complete set of the HDF5 APIs, see Appendix B "HDF5 Reference Manual."

8.1 Introduction to the HDF5 Programming Model

The HDF5 programming model defines how operations are performed on HDF5 objects. It consists of five steps as described in the table below. Steps marked with an asterisk (*) are optional.

Step	***Description***	***Example***
1*	Properties of an HDF5 object are defined.	Properties are set to create an extendible dataset in step 2.
2	The HDF5 object is accessed (opened or created).	Dataset is created; operations can be performed on the dataset.
3*	Properties of operations on the HDF5 objects are defined.	Size of type conversion buffer is set to tune performance.
4	Operations on the HDF5 object are performed.	Dataset is written to the file.
5	Access to the HDF5 object is terminated.	Access to the dataset is terminated; any further operation on it will fail.

The order of the steps is important. Step 1 has to be done before steps 2 – 5 (i.e. immutable structural metadata has to be defined before an object is created and accessed); obviously step 3 has to be performed before step 4; step 5 can be performed only if step 2 is complete; step 4 cannot be performed if access to an object was terminated (step 5).

The HDF5 Library refers to an object via an object identifier. An identifier is created when the object is opened or created. The identifier is used to invoke subsequent operations on the object. The identifier is also used to specify dependencies between objects. For example, a dataset is always created under some group; the creation operation on the dataset therefore uses the group identifier to specify the location of the dataset in the File hierarchy.

8.2 HDF5 APIs

The HDF5 APIs enforce the HDF5 Data Model and the HDF5 programming model. An application program uses HDF5 APIs to map its data structures to HDF5 objects and to perform operations on those objects.

The complete set of the HDF5 APIs is implemented in the C language; a subset of the APIs is also implemented in the Fortran 90, C++ and Java languages. This proposal describes the HDF5 C APIs only.

The names of the C HDF5 APIs follow the following convention: H5<A,D,E,F,G,I,P,R,S,T,Z><string>, where <string> is a sequence of lower case characters "a – z" and "_"; <string> always starts with a character. Each of the capital letters corresponds to the set of APIs that implements operations on a particular HDF5 object or on an HDF5 Library object. The table below summarizes the HDF5 APIs.

Prefix	***Description (examples)***
H5A	Operations on an Attribute object (H5Acreate, H5Awrite, H5Aclose)
H5D	Operations on a Dataset object (H5Dopen, H5Diterate)
H5E	Error handling operations (H5Eprint, H5Eclear)
H5F	Operations on a File object (H5Fis_hdf5, H5Fmount)
H5G	Operations on Group objects (H5Gcreate, H5Gunlink)
H5I	Operations on object identifiers (H5Iget_name, H5Iget_type)
H5P	Operations to set and modify object properties and properties of operations on objects (H5Pcreate, H5Pset_fapl_family, H5Pset_chunk)
H5R	Operations on References to Objects and References to Dataset Regions (H5Rcreate, H5Rdereference, H5Rget_obj_type)
H5S	Operations on Dataspace objects (H5Screate_simple, H5Sselect_hyperslab)
H5T	Operations on Datatype objects (H5Tcopy, H5Tget_size)
H5Z	Operations on user-defined filters

Appendix B includes lists of all HDF5 C and Fortran90 APIs and complete descriptions of each.

8.3 The HDF5 Library I/O features

The next sections introduce the main three features of the HDF5 I/O library: spatial transformation of raw data during I/O operations (partial I/O), data transformation operations on each element of the raw data during I/O operations (HDF5 filters), and special-purpose I/O mechanisms to specify physical storage layouts (Virtual File Layer or VFL).

8.3.1 Partial I/O

The HDF5 Library provides a mechanism for spatial transformation.

Spatial transformations define complex selections of the dataset elements on which the I/O operation will be performed.

HDF5 defines two types of selections:

Selection type	***Description***
Point selection	One element of a dataset identified by array indices
Simple hyperslab selection	Contiguous sub-array identified by an offset from the beginning of the array in each dimension, and the size in each dimension

All selections are built from one of those two types using set operations:

- union
- difference
- intersection
- complement

Set operations on hyperslab or point selections allow the definition of very complex, irregularly shaped subsets of an N-dimensional array. During I/O operations, the shape and the dimensionality may not be preserved, i.e. a 2-dimensional sub-array from a dataset in a file (source) may be scattered to an irregularly shaped subset of a 3-dimensional array in memory (destination) during the read operation, as long as the number of elements in both selections is the same.

Elements in each selection are ordered according to the C ordering rule (row-column). The i-th element in the source selection is transformed (read/written) to the i-th element in the destination selection.

8.3.2 Filters

The HDF5 Library defines transformation (filters) on each element of a dataset that participates in I/O operation. Filters may be combined together to create a filter pipeline. When an element goes through the filter pipeline, the result of a transformation becomes an input to the transformation next in the pipeline.

Permanent transformations on elements are defined when a dataset is created and cannot be removed after that. Transient transformations are defined on every I/O operations. The HDF5 Library standard defines only permanent transformations given in the table below. Transformations marked with an asterisk (*) may be performed on the elements of chunked datasets only.

Transformation type	***Description***
Datatype conversion	Conversion of the numeric values of the same class (integer to integer, or floating-point to floating-point)
Raw data shuffling*	Rearranging bits in each element to achieve better compression ratio
Raw data compression*	Raw data is stored in compressed form; two types of compressions are supported: o GNU zlib compression o NASA szip compression
Raw data error detection*	Error-detecting codes (EDC) provide a way to identify data that has been corrupted during storage or transmission; HDF5 supports Fletcher32 algorithm

8.3.3 Virtual File Layer

The HDF5 file format describes how HDF5 data structures and dataset raw data are mapped to a linear format address space (HDF5 logical file). The HDF5 Library implements the mapping in terms of an API. However, the HDF5 format specifications do *not* indicate how the format address space is mapped onto storage and how the data in the storage is accessed. The Virtual File Layer allows an application to design and implement its own mapping between the HDF5 logical file and storage. It also allows defining the type of access. The proposed standard defines four different storage layouts (see 6.3.1) and corresponding access methods:

Storage layout	***Access method***
Single file on a storage media	POSIX unbuffered I/O POSIX buffered I/O MPI I/O
Multi files	POSIX unbuffered I/O
Family of files	POSIX unbuffered I/O
Core (memory) file	Memory access

9 Authors' Address

Mike Folk, Elena Pourmal , The HDF Group 605 E. Springfield Ave, Champaign, IL 6182 USA
Tel: 217-333-0238 fax 217-244-1987, email: mfolk@hdfgroup.org , epourmal@hdfgroup.org

10 Appendix A – HDF5 File Format Specification

11 Appendix B – HDF5 API Reference Manual

12 Appendix C – Glossary of acronyms

Acronym	Description
HDF5	Hierarchical Data Format, version 5
CCSDS:	Consultative Committee for Space Data Systems
ECS:	EOSDIS Core System
EOSDIS:	Earth Observing System Data and Information System

13 Appendix D – Errata

There are no errata for the document.

14 Appendix E – HDF5 copyright