Date: September 29, 2013 Delivered as part of Milestones 5.6 & 5.7	*HDF5 Data in IOD Containers Layout Specification* **FOR EXTREME-SCALE COMPUTING RESEARCH AND DEVELOPMENT (FAST FORWARD) STORAGE AND I/O**

LLNS Subcontract No.	B599860
Subcontractor Name	Intel Federal LLC
Subcontractor Address	2200 Mission College Blvd. Santa Clara, CA 95052

Table of Contents

Revision History

Date	Revision	History	Author
Mar. 21, 2013	1.0	First Draft	Mohamad Chaarawi, Quincey Koziol – The HDF Group
Mar. 21, 2013	1.1	• More details on file format • Submitted to DOE - M3.3	Mohamad Chaarawi, Quincey Koziol – The HDF Group
June 17, 2013	2.0	• Added layout spec for Named Datatypes. • Added layout spec for Attributes. • Submitted to DOE – M4.2	Mohamad Chaarawi, Quincey Koziol – The HDF Group
September 25, 2013	3.0	• Revamped the entire document • Removed Old picture and created new ones through MW itself. This will make modification to graphics much easier. • Added Maps. • Added VL datatypes.	Mohamad Chaarawi – The HDF Group
September 29, 2013	3.1	• Changed title	Ruth Aydt – The HDF Group

Introduction

The document details how HDF5 File, Group, Dataset, and Attribute objects map to IOD objects. The application will call the HDF5 library while running on the system's compute nodes (CNs). Using the VOL architecture, the IOD VOL plugin will use a function shipper (FS) to forward the VOL calls to a server component running on the I/O nodes (IONs). At that point, the VOL calls are translated into I/O Dispatcher (IOD) API calls and executed at the IONs. The translation includes mapping HDF5 object into IOD objects. The IOD will be responsible for storing the data on distributed storage using DAOS.

Definitions

CN = Compute Node

EFF = Exascale FastForward

FS = Function Shipper

IOD = I/O Dispatcher

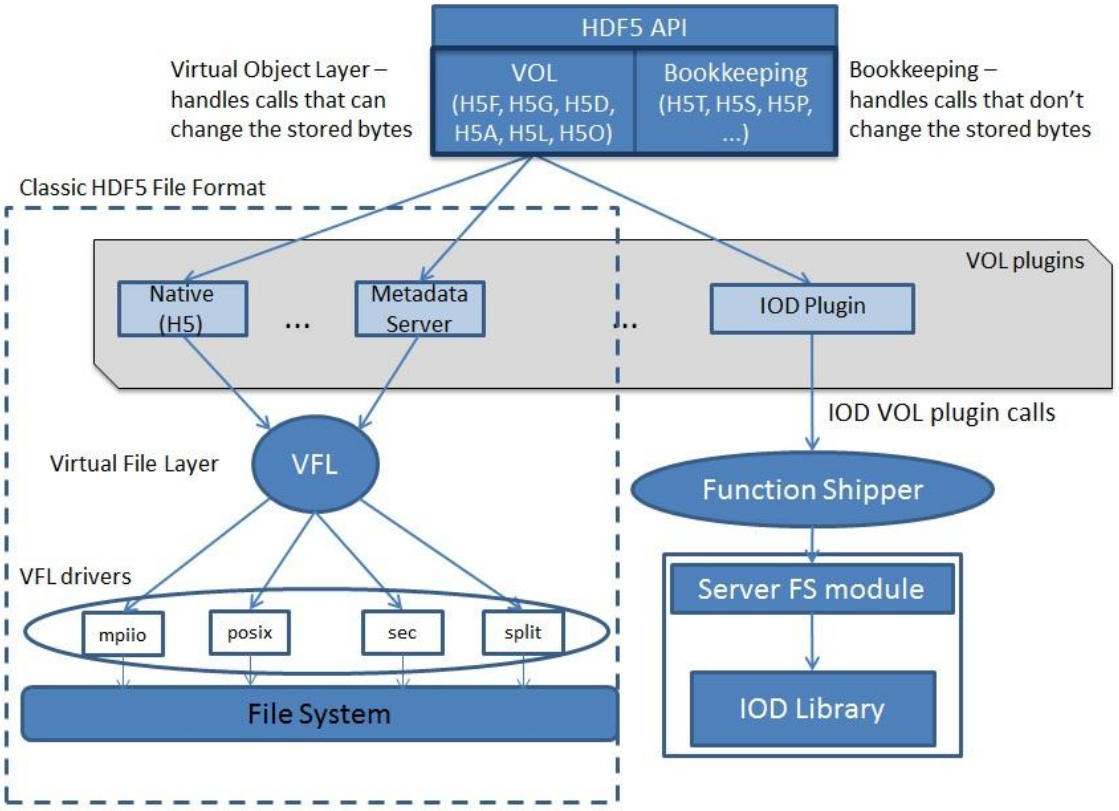
ION = I/O Node

VOL = Virtual Object Layer

Background

The HDF5 VOL intercepts all HDF5 calls that would potentially touch the storage and routes them to an internal or user-developed plugin. This allows for HDF5 objects to be stored in different file formats or storage abstractions that are hidden from the application, allowing the application to continue using the same HDF5 API and data model while benefiting from new storage methods and architectures.

The overall architecture of the HDF5 library with the addition of an IOD VOL plugin looks like this:



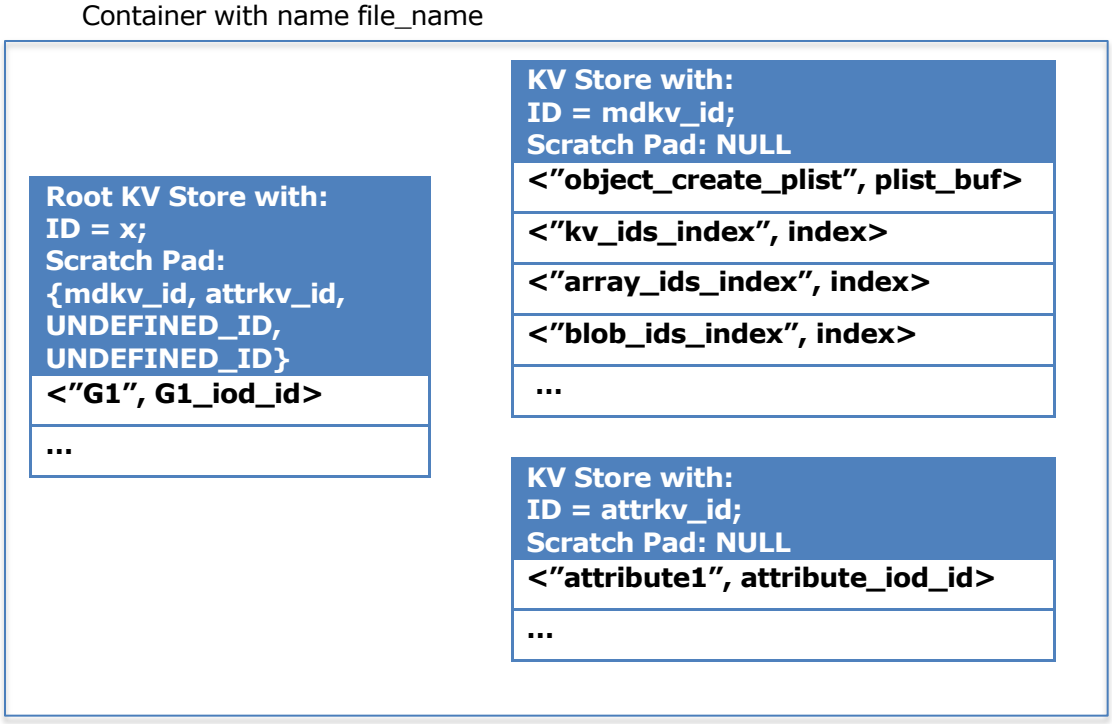
The HDF5 to IOD Object Mapping

The EFF storage software stack contains several components essential to the proper functioning and performance of the application I/O. The scope of this document is from the HDF5 library to the high level IOD API routines. We will not discuss here how IOD implements its API internally or its interaction with the DAOS library and underlying distributed storage.

1.1 HDF5 File -> IOD Container

An HDF5 file is mapped to an IOD container. The container is created/opened by the file name. In addition to that, the HDF5 library specifies that each file contains a root group. The root group always has a predefined ID (for now we will make that ID be 0). From that root ID, one can reach any object in the container by traversing the KV entries in the group KV store objects, retrieving the object names and corresponding IOD IDs of each group to reach the final object.

Since the root object is an actual HDF5 group, it will be stored in IOD the same way we store any group as described in 1.1. The following diagram shows an HDF5 file with a group /G1 under the root group, and an attribute "attribute1" attached to the file:



The creation property list can be created by calling *H5Pdecode()* on the binary buffer that is stored in the "object_create_plist" Key in the metadata KV store. This also applies to all creation property lists for all objects that are described next.

The indexes stored in the metadata KV for the file indicate the starting index for IDs that the client should use to generate IOD IDs for KV, Array, and BLOB objects.

1.2 HDF5 Group -> IOD Key Value Store Object

An HDF5 group maps to a IOD KV store object. To access the KV store object, one must have its IOD ID. From this ID, one can open that object and retrieve/add/update the KV pairs in that KV store object.

The KV pairs in the KV object describe links to the child objects that are linked to from this group. The key is a string that contains the link name and the value associated with that key contains that object's IOD ID. For example, a group "G1" that is a parent of dataset "G1/D1" and another group "G1/G2" will contain the following KV pairs:

<"D1", D1_iod_id>

<"G2", G2_iod_id>

In addition to the "links" to its children, an HDF5 group has its own metadata associated with it that it stored in the metadata KV. It includes the creation properties, the HDF type for the object (H5I_GROUP), and the number of links that point to this group. Attribute objects can be attached to an HDF5 group, all which need to be stored and retrieved from the file. To be able to store that information, we will use the scratch pad field that IOD provides for each object. In this 32-byte scratch pad we will store two IOD IDs for auxiliary KV store objects, KV1 and KV2, that we create to store metadata and attribute information for the object, respectively. (KV1's ID is stored in the upper 16 byte of the scratch pad and KV2's ID in the lower 16 bytes).

Here is an example that summarizes the mapping of HDF5 group with ID G1_iod_id with two child objects and three attribute objects in IOD:

Group KV Store with: ID = G1_iod_id; Scratch Pad: {mdkv_id, attrkv_id, UNDEFINED_ID, UNDEFINED_ID}
<"D1", D1_iod_id>
<"G2", G2_iod_id>
…

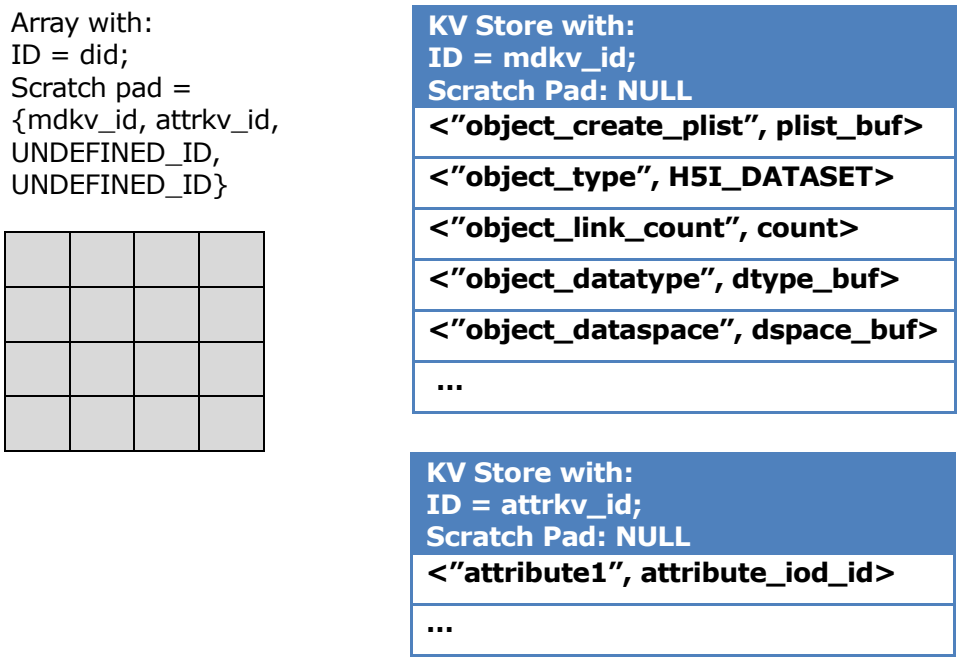
KV Store with: ID = mdkv_id; Scratch Pad: NULL
<"object_create_plist", plist_buf>
<"object_type", H5I_GROUP>
<"object_link_count", count>
…

KV Store with: ID = attrkv_id; Scratch Pad: NULL
<"attribute1", attribute_iod_id>
<"attribute2", attribute_iod_id>
<"attribute3", attribute_iod_id>
…

1.3 HDF5 Dataset -> IOD Array Object

An HDF5 dataset is stored as an IOD array object. The IOD array dimensions are determined from the HDF5 dataspace specified by the application. Similarly, the size of each IOD array element is determined by the HDF5 datatype. The IOD array creation operation allows us to chunk a dataset in the same way that HDF5 allows the application to store a chunked dataset.

Similarly to groups, HDF5 stores metadata associated with datasets (i.e. the datatype, dataspace, etc…) and allows users to attach attributes to a dataset. This is handled again with an IOD scratch pad field in the array object that contains IDs for an auxiliary metadata KV store object and attribute KV store object for each dataset, stored in the same order as for the group scratch pad information. Here is a diagram that describes a dataset in IOD with attribute "attribute1":



The datatype and dataspace objects can be retrieved from the binary buffers stored in the metadata KV store by calling *H5Tdecode()* and *H5Sdecode()* respectively.

Fixed Length Data

Datasets that are created with element types that are of fixed length (integers, floats, compound types of fixed length types, etc…) will have their data elements stored directly in the cells of the IOD array object.

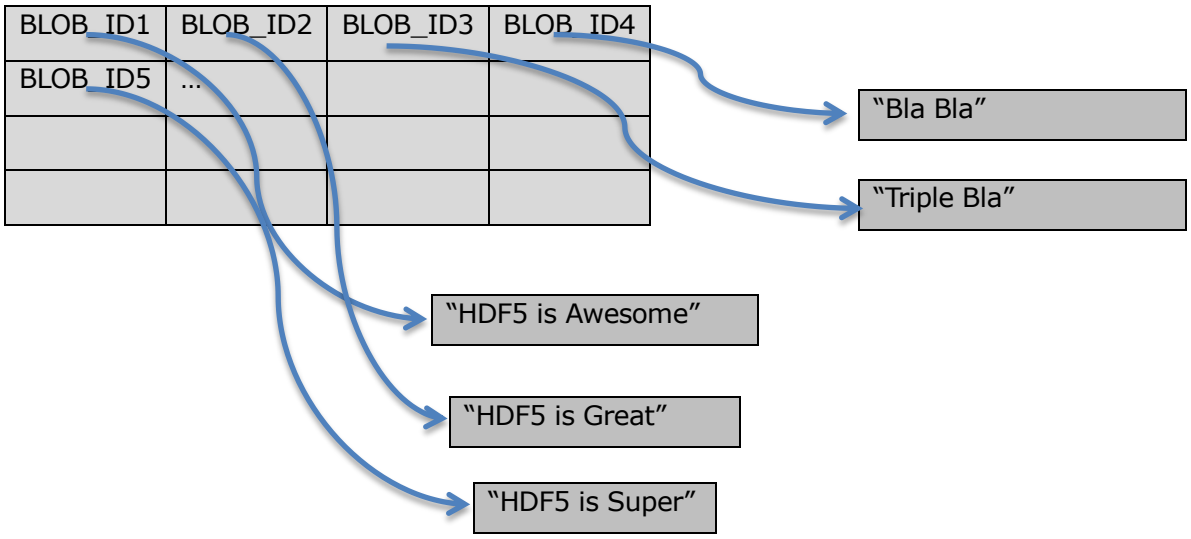
For example, if a dataset has been created with an integer type, the following figure shows how the IOD array that represents the dataset in IOD might look like:

0	25	6	8
7	90	26	78
87	676	87	56
46	67	87	56

Variable Length Data

The HDF5 library allows users to specify that elements of datasets be of variable length. This can be done by passing a VL datatype to the dataset creation operation. Since IOD does not have the capability to store variable length data in array cells, we instead create a blob object for each cell that is written to the IOD array object. The BLOB corresponding to every cell will hold the variable length data that was supposed to be stored in the array cell originally.

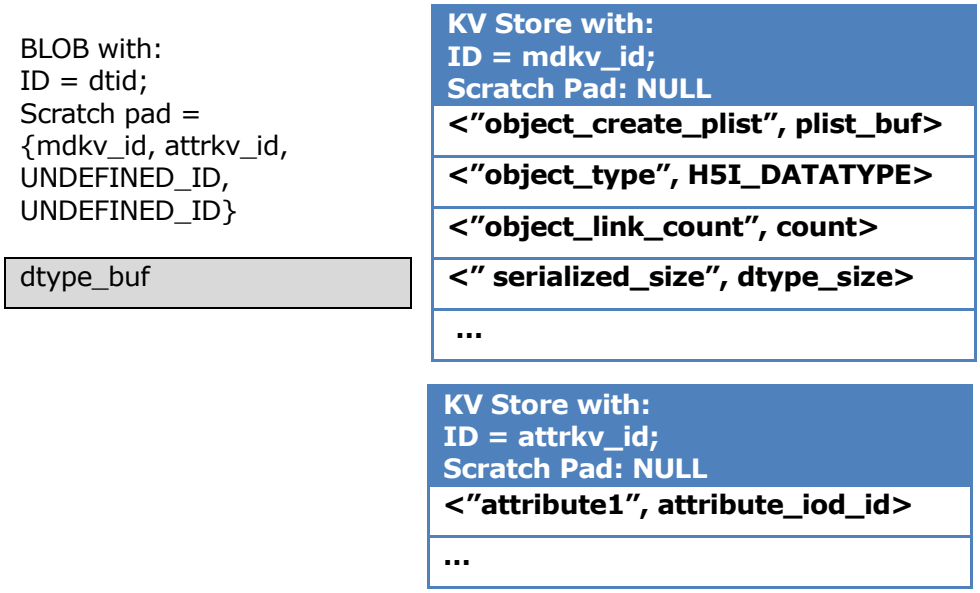
For example, if a dataset has been created with a variable length string type, here is what the representation in IOD might be:



1.4 HDF5 Named Datatypes

An HDF5 named datatype will be stored in IOD as a BLOB object. The BLOB will contain the binary encoded form of the datatype that can be obtained by calling *H5Tencode()* on the HDF5 datatype. Calling *H5Tdecode()* on that binary buffer will reconstruct the HDF5 datatype object.

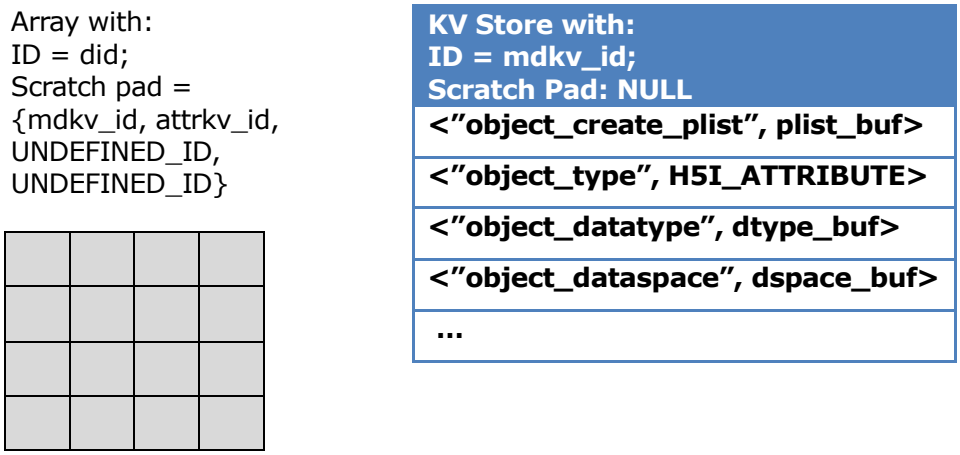
Similarly to group and dataset objects, HDF5 stores metadata associated with named datatypes and allows users to attach attributes to them. This is handled again with an IOD scratch pad field in the array object that contains IDs for an auxiliary metadata KV store object and attribute KV store object for each named datatype, stored in the same order as for the group scratch pad information.



1.5 HDF5 Attributes

An HDF5 attribute will be represented in IOD with an array object, the same as an HDF5 Dataset is stored. The IOD array dimensions are determined from the HDF5 dataspace specified by the application. Similarly, the size of each IOD array element is determined by the HDF5 datatype.

The metadata for an attribute is stored in an IOD scratch pad of the array object, which will contain a metadata KV object. Since attributes cannot have other attributes attached to them, the scratch pad will not include an attribute KV object for storing attribute IDs. Here is a diagram that describes an attribute in IOD:



Attributes always are created attached to an object (File, Group, Dataset, or Named Datatype). Objects can have multiple attributes attached to them. As we have shown previously, each type of those objects has a separate KV store for storing attributes in their scratch pad. The Key will hold the attribute name which has to be unique for a particular object (i.e. an object cannot have two attributes with the same name), and the value will hold the IOD ID for the actual attribute object.

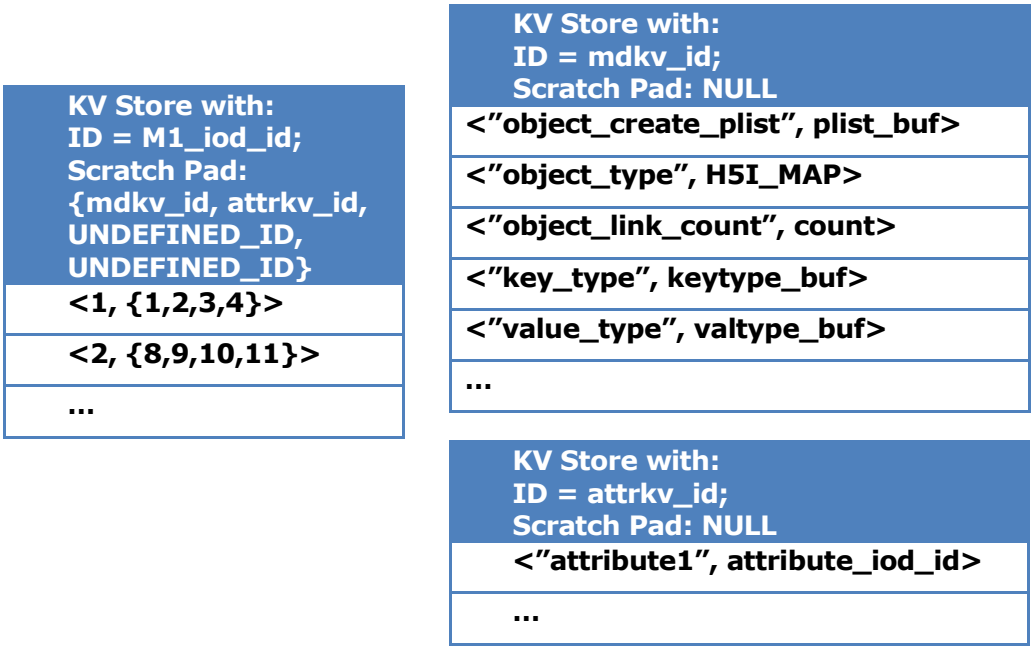
1.6 HDF5 Maps

An HDF5 map object maps directly to an IOD KV store object. To access the KV store object, one must have its IOD ID. From this ID, one can open that object and retrieve/add/update the KV pairs in that KV store object.

The KV pairs in the KV object are identical to the KV pairs the application sets to the MAP object.

In addition to the KV pairs, an HDF5 map has its own metadata associated with it that it stores in the metadata KV. It includes the creation properties, the datatype for Keys and Values, the HDF5 object type (H5I_MAP), and the number of links that point to this map. Attribute objects can be attached to an HDF5 map, all which need to be stored and retrieved from the file.

Here is an example that summarizes the mapping of an HDF5 map with ID M1_iod_id with two KV pairs and an attribute object in IOD:



1.7 Metadata format

The metadata stored in auxiliary KV objects for group and dataset objects is formatted in the same manner as specified in the HDF5 File Format Specification: http://www.hdfgroup.org/HDF5/doc/H5.format.html

Open Issues

The HDF5 library allows users to create VL datatypes nested within other VL datatypes. Furthermore it allows VL datatypes to be part of compound and array datatypes. We will not support such complex form of variable length data since their usage is not common in user applications.

Risks & Unknowns

The IOD implementation is not completely specified yet and details described above may still change if IOD's features change in an incompatible way.