Hierarchical Data Format, version 5 (HDF5)

Contents

References

Wikipedia
Main website for the HDF group
Tutorials
Examples

Training material

- Git repository - source code examples
- /ome/team/rleigh/hdf5/samples - sample images
- hdfview download

Introduction

Format features and goals

In 1987, HDF was created by the NCSA to solve the problem of moving data, originally image data, between different computing platforms. HDF is thus architecture independent by design. The format was influenced by file formats of that era, and their limitations, including TIFF and CGM . HDF was designed to be able to store and access large data objects efficiently, as well as being able to store multiple data objects within a single file “container”. It was also designed to be extensible so that it could store new types of data and metadata and so meet unanticipated future needs. C and Fortran were, and are, the primary languages for using HDF. HDF4 (the original format) was extended and simplified to result in HDF5, to scale from gigabyte to petabyte scale storage.

HDF has been adopted by many organisations, for diverse purposes other than imaging, including seismic data, astronomy and satellite imaging and mapping, simulation, weather data, and biological data of varied types.

HDF5 is used as the basis for several scientific imaging formats, including these supported by Bio-Formats:

- CellH5 (complex: grouped datasets and tables with attributes)
- BigDataViewer (XML view registration with HDF5 backend; other formats also supported)
- Imaris IMS (complex: grouped datasets and tables with attributes)

- Veeco AFM (simple: single dataset with attributes)
- Medical Image NetCDF (MINC) (simple: multiple datasets, no attributes)

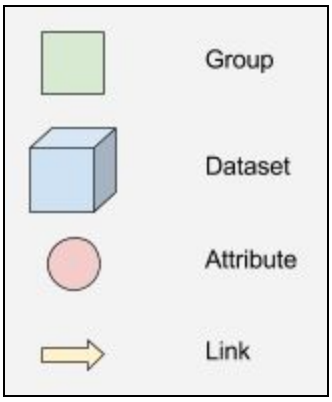
HDF is also used as the storage format for Matlab and PyTables.

HDF5 is a generalised format acting as a container of arbitrary data, with specialised support for multi-dimensional arrays. It can be used for a diverse range of applications, storing data with very different structures.

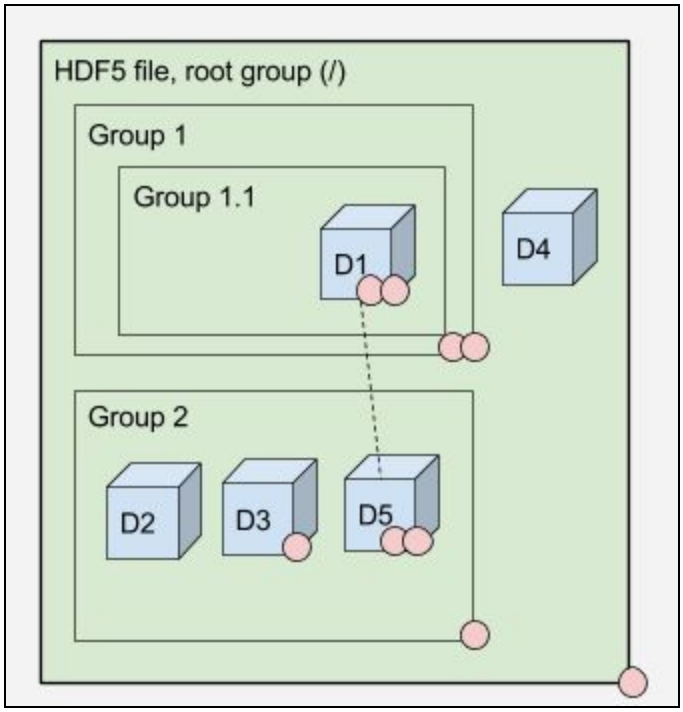
Format details

HDF5 was developed to store data in a structured binary data, primarily multi-dimensional arrays of numeric data, but also allowing for arbitrary binary data, metadata, and arbitrary data types. It provides a flexible means of storing large and complex scientific data in a binary container for fast and efficient access. The main structures in an HDF file are:

- Groups (can nest to form a hierarchy of containers)
- Datasets (multi-dimensional arrays)
- Property lists (control behaviour and storage of HDF objects)
- Datatypes (storage specification of stored data)
- Dataspaces (selections of dataset regions)
- Attributes (user metadata)
- Links (between objects within an HDF file, or between different HDF files)

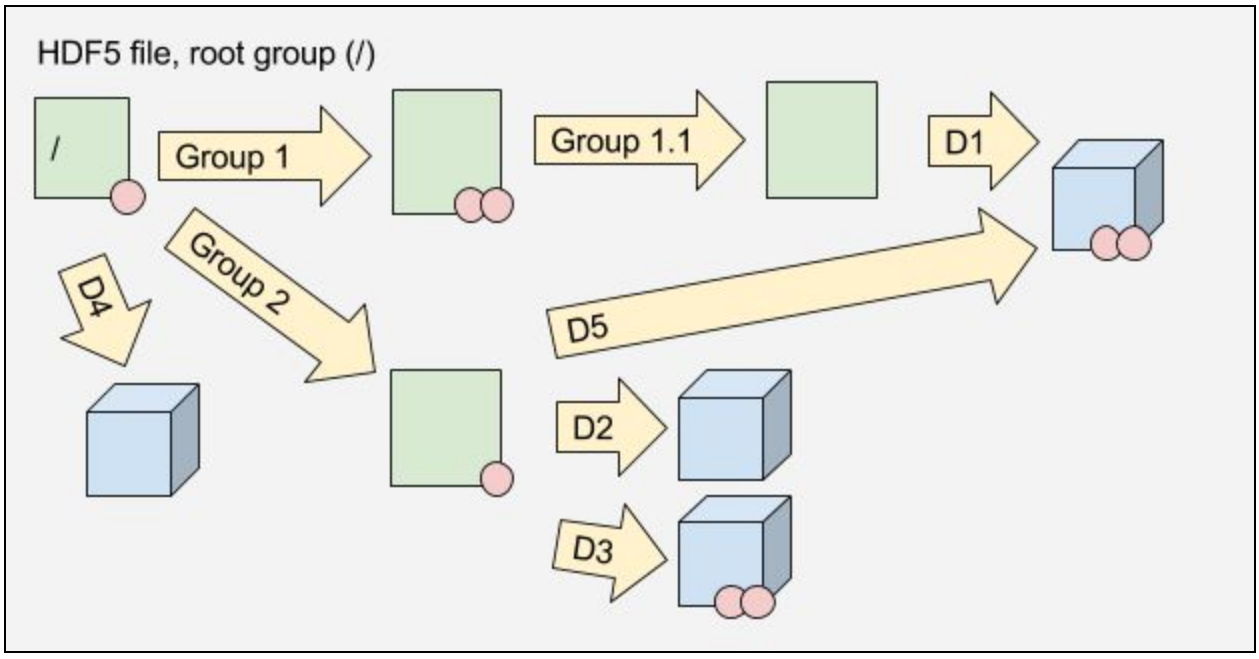


Datasets are contained within groups, which may be nested:



Like directories and files on a UNIX filesystem, groups and datasets don't have names. *Links* to groups and datasets have names. Links can be hard or symbolic, and can also be cyclic. Symbolic links can be absolute, relative or external (to a group or dataset in another HDF5 file).

As a graph:

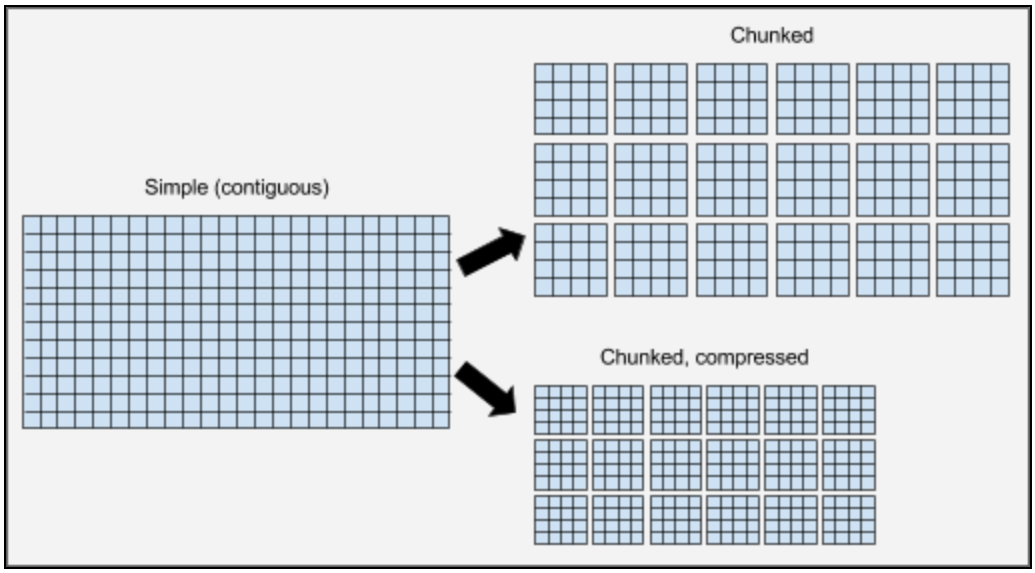


As a tree:

```
/
/D4
/Group1
/Group1/Group1.1
/Group1/Group1.1/D1
/Group2
/Group2/D2
/Group2/D3
/Group2/D5 (same dataset as /Group1/Group1.1/D1)
```

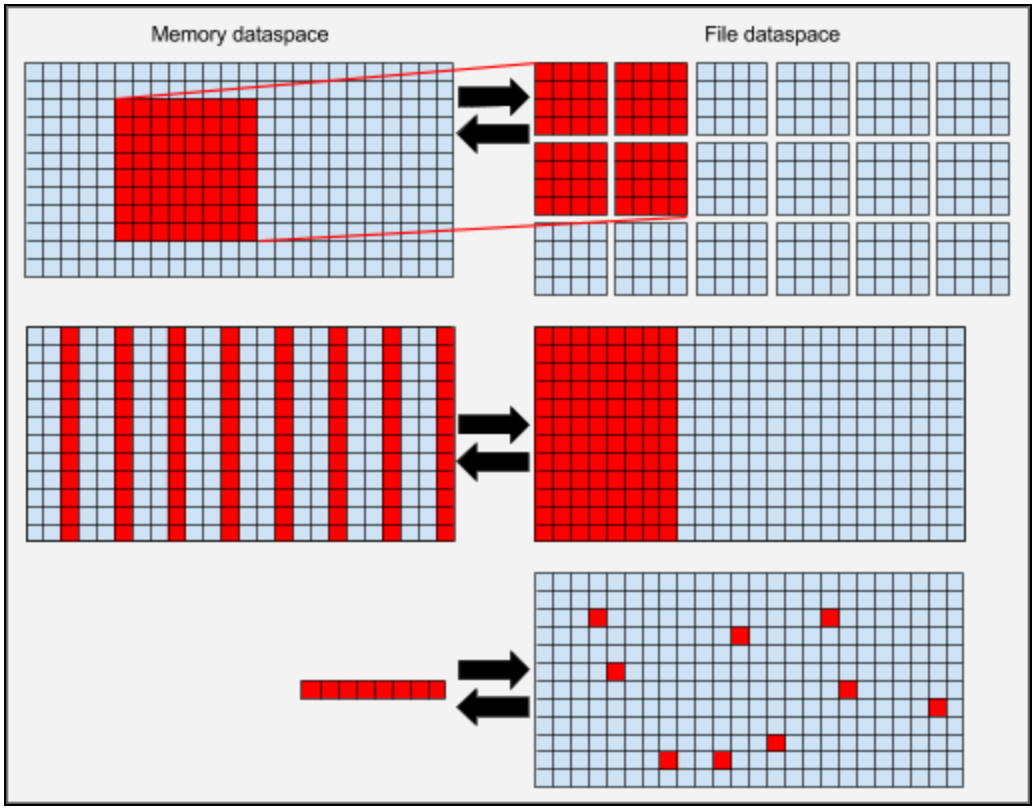
Data storage

- Rank (number of dimensions)
- Extents (size of each dimension)
- Chunks (splitting of each dimension)
- Compression



Data transfer:

A dataspace exists for both the data in memory and the dataset in the file. The data to transfer is selected by choosing the range and step of the data (`select_hyperslab`, `select_elements`), or by manually specifying individual element indexes. These may be combined to create complex selections. Data may then be transferred between the data in memory and the file dataset.


Memory dataspace
File dataspace

Tools

- Summary of available tools

Command Line

- `h5dump`
- `h5import`
- `h5copy`
- `h5diff`
- `h5ls`
- `h5ftotxt and h5fromtxt`

Graphical

- `Hdfview (download)`

Language bindings

C++

- `libhdf5_serial`
- `libhdf5_cpp`

Java

- JHDF5
- CDM (NetCDF)

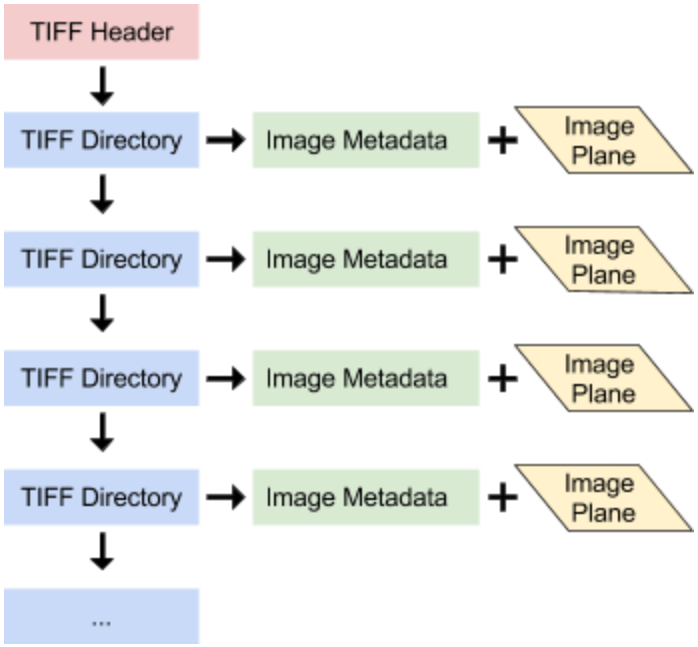
Python

- h5py
- PyTables
- Pandas

Examples (C++ and Python)

1. `write-array` - write contiguous array with groups and annotations
2. `read-array` - read arrray
3. `read-array-subset` - read subset of array
4. `write-chunked-array` - write chunked array

Comparison with TIFF



TIFF supports a linear list of image planes. Navigation requires starting at the first image directory and seeking along the chain of directory offsets. In contrast, HDF5 is much more flexible in its use of grouping and access by name rather than by index. Images are limited to single planes, i.e. two dimensions rather than an arbitrary rank. TIFF supports chunking and compression, but access is via tile index and requires writing in a linear order while HDF5 permits writing in any order and by using natural ranges rather than complex index calculations. Metadata is attached to TIFF directories as numeric tags with arbitrarily complex values. However, there is a central tag registry and so it is not as extensible as HDF5 in practice; if you create a new tag no other user will know the structure of the stored value. With HDF5 this is explicit.

Problems

- Single implementation (in C)
- Need for repacking to consolidate space
- Character encoding undefined
- Thread safety

Applications

OMERO.tables

A service to provide access to an HDF5 table. This is columnar data using a limited set of defined types, i.e. like an R data.frame, and not a general-purpose interface to HDF5. It provides access to a limited subset of HDF5 functionality. Useful for e.g. summary data from analysis but not for storing raw data in multi-dimensional arrays.

- Documentation
- Java example
- Python example

Potential use by OME

OME-HDF5?

- Store 5D images as 5D hypervolume; no more DimensionOrder/Interleaving/RGBChannelCount and 2D plane limitations
- Simplify fetching and ordering of image data from disc; replace openBytes with simple fetch of exactly the data requested
- Simplify image writing; ordering constraints are lifted
- OME-XML metadata can be stored as string attribute
- OME-XML metadata could alternatively be stored directly as attributes on datasets and groups:
    - More efficient to open, since unused metadata isn't read into memory
    - More efficient to read, since it's binary
    - Hard links can deduplicate repeated metadata such as channel and instrument metadata
- Analysis results can be stored in HDF5 and link back to the original datasets (including external references to the original datafiles)
- Interoperability with HDF5 libraries allows easy access to OME image data and metadata without the need for using our library APIs.
- The existing model and metadatastore APIs are not designed with HDF in mind; they would perform poorly unless they were updated to take advantage of HDF features.