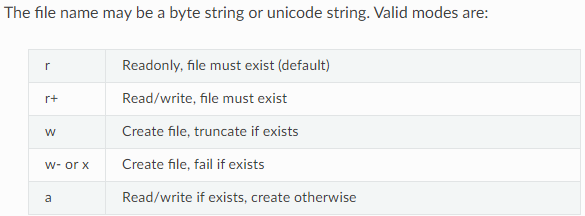
Library Functions Definitions

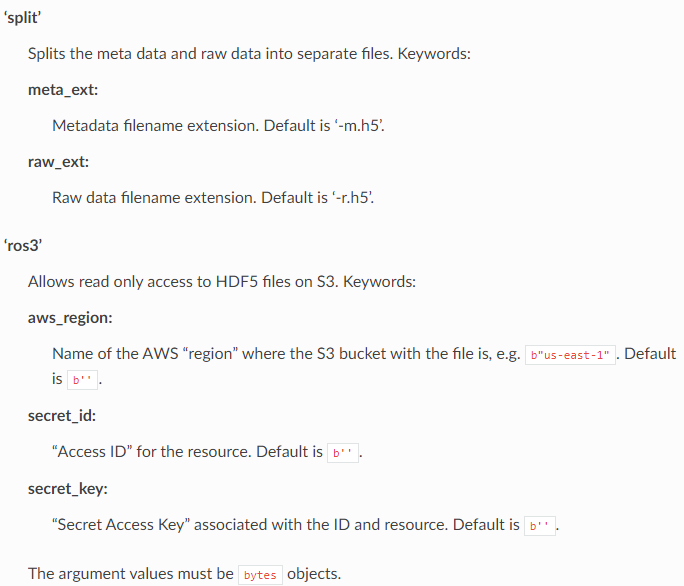
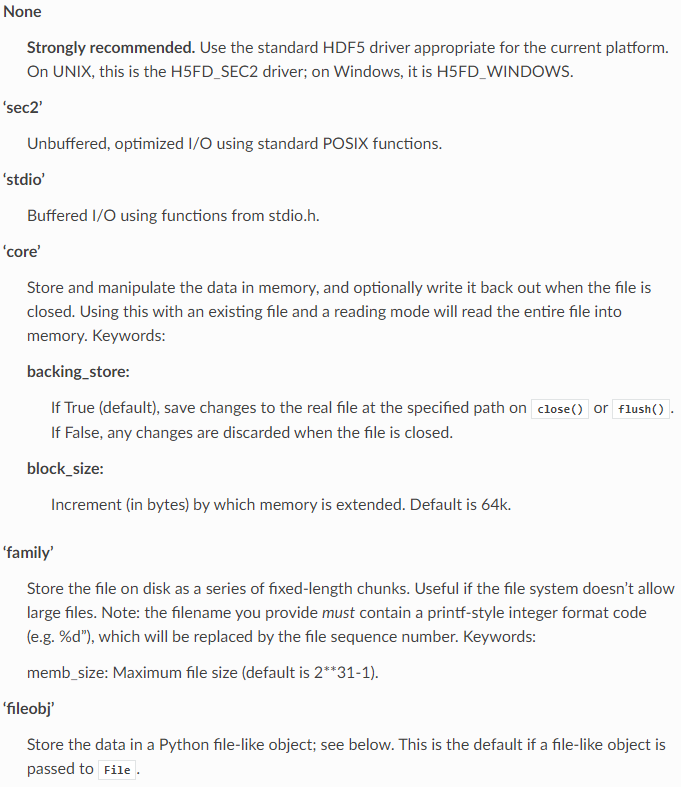
h5py (for reading in hdf)

Opening and creating files: f = h5py.File('myfile.hdf5','r')

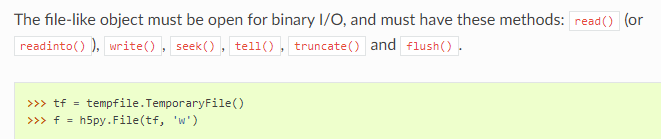


File drivers: f = h5py.File('myfile.hdf5', driver=<driver name>, <driver\_kwds>)

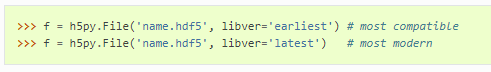
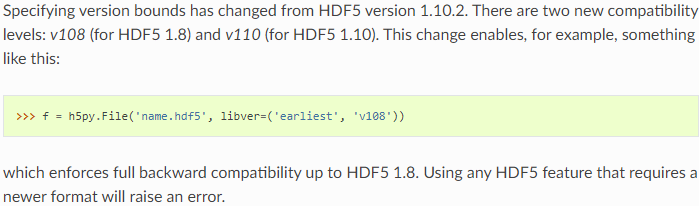
* HDF5 ships with a variety of different low-level drivers, which map the logical HDF5 address space to different storage mechanisms.
* You can specify which driver you want to use when the file is opened.
* For example, the HDF5 “core” driver can be used to create a purely in-memory HDF5 file, optionally written out to disk when it is closed.



Python file like objects:

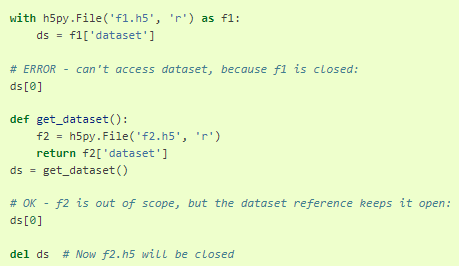
The first argument to [**File**](https://docs.h5py.org/en/latest/high/file.html#h5py.File) may be a Python file-like object, such as an **io.BytesIO** or **tempfile.TemporaryFile** instance. This is a convenient way to create temporary HDF5 files, e.g., for testing or to send over the network. 

Version:

* By default, the library will write objects in the most compatible fashion possible, so that older versions will still be able to read files generated by modern programs.
* However, there can be feature or performance advantages if you are willing to forgo a certain level of backwards compatibility.
* By using the “libver” option to [**File**](https://docs.h5py.org/en/latest/high/file.html#h5py.File), you can specify the minimum and maximum sophistication of these structures: 
* 

Closing files:

* If you call **[File.close()](https://docs.h5py.org/en/latest/high/file.html" \l "h5py.File.close" \o "h5py.File.close)**, or leave a with h5py.File(...) block, the file will be closed and any objects (such as groups or datasets) you have from that file will become unusable.
* This is equivalent to what HDF5 calls ‘strong’ closing.
* If a file object goes out of scope in your Python code, the file will only be closed when there are no remaining objects belonging to it.
* This is what HDF5 calls ‘weak’ closing.

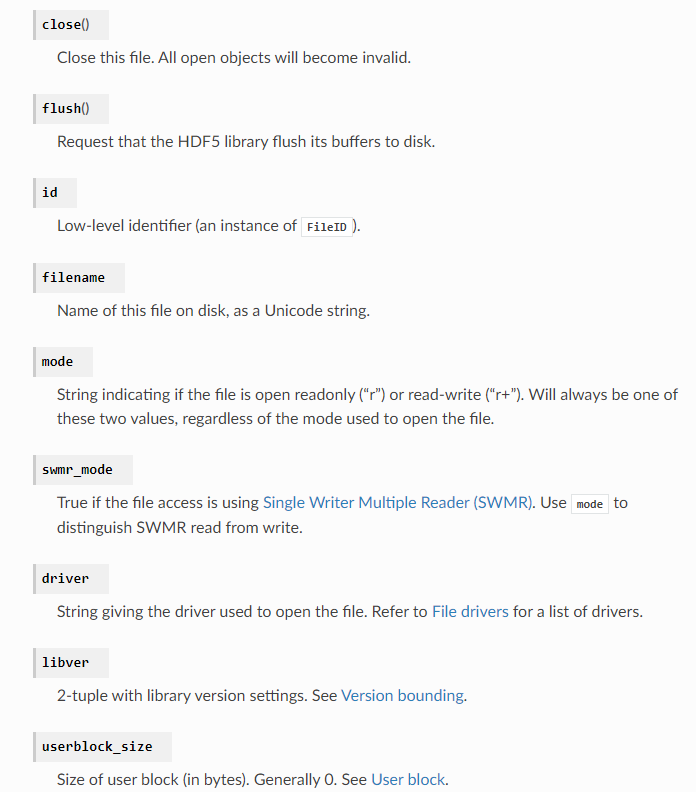


Userblock:

* HDF5 allows the user to insert arbitrary data at the beginning of the file, in a reserved space called the user block.
* The length of the user block must be specified when the file is created. It can be either zero (the default) or a power of two greater than or equal to 512.
* You can specify the size of the user block when creating a new file, via the userblock\_size keyword to File; the userblock size of an open file can likewise be queried through the File.userblock\_size property.
* Modifying the user block on an open file is not supported; this is a limitation of the HDF5 library. However, once the file is closed you are free to read and write data at the start of the file, provided your modifications don’t leave the user block region.

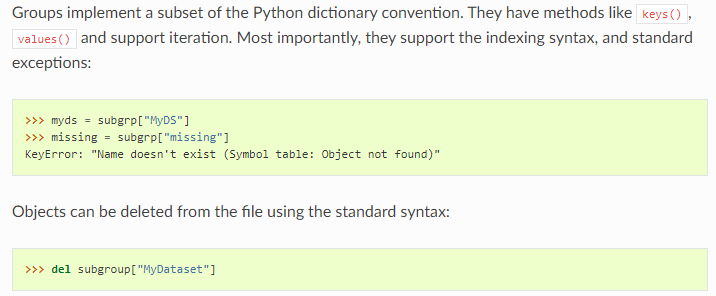
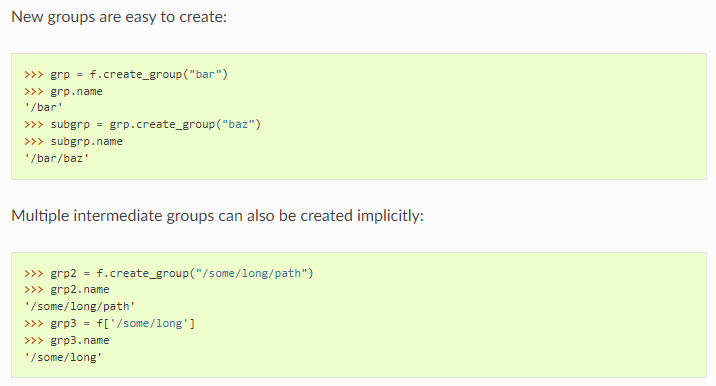
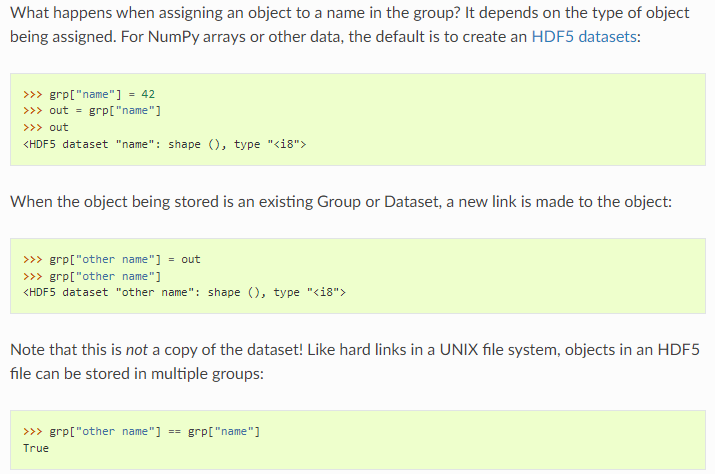
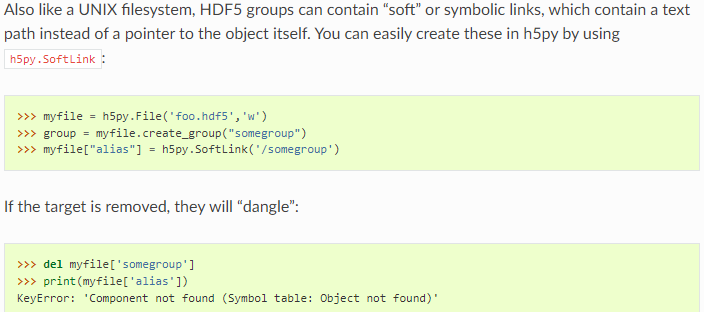
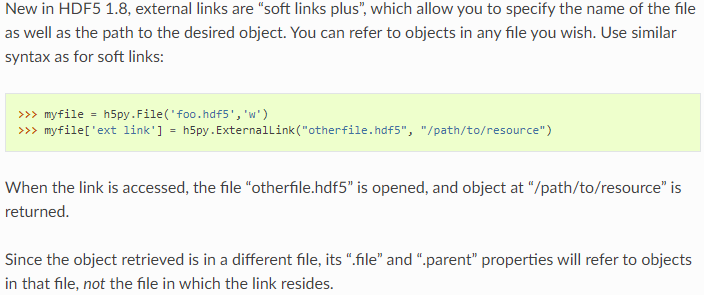
Filenames:

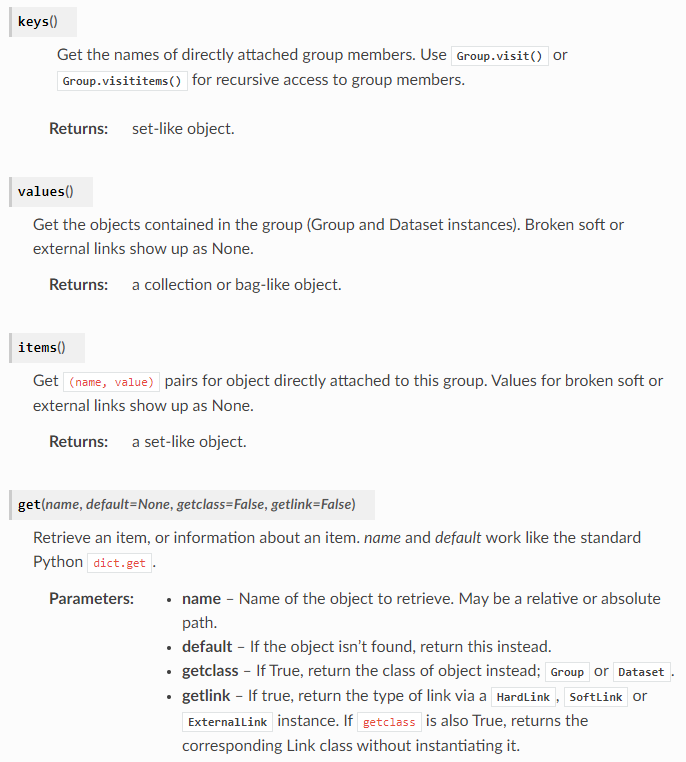
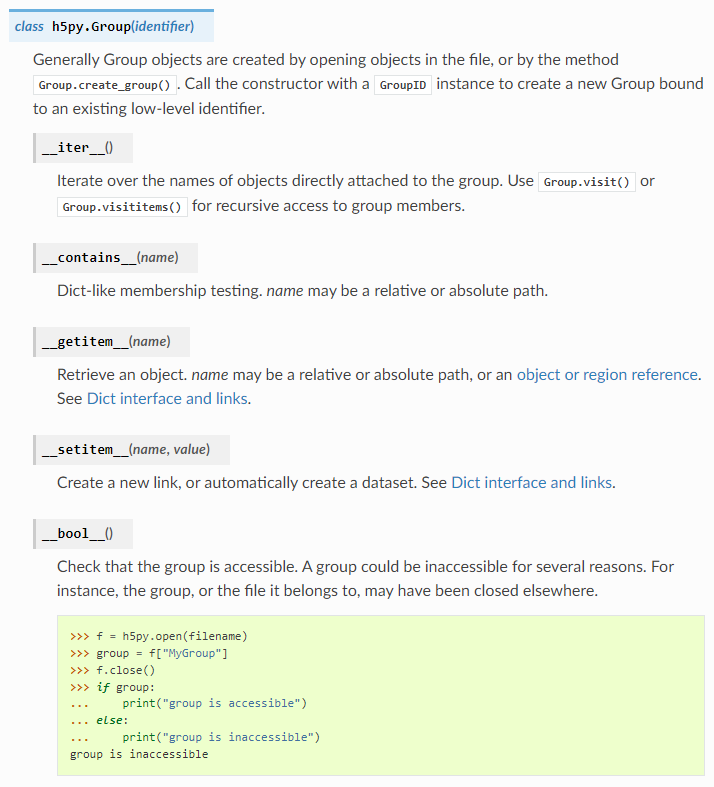
* Different operating systems (and different file systems) store filenames with different encodings. Additionally, in Python there are at least two different representations of filenames, as encoded bytes or as a Unicode string (str on Python 3).
* h5py’s high-level interfaces always return filenames as str, e.g. **[File.filename](https://docs.h5py.org/en/latest/high/file.html" \l "h5py.File.filename" \o "h5py.File.filename)**. h5py accepts filenames as either str or bytes. In most cases, using Unicode (str) paths is preferred (to be used on macOS and Windows), but there are some caveats.



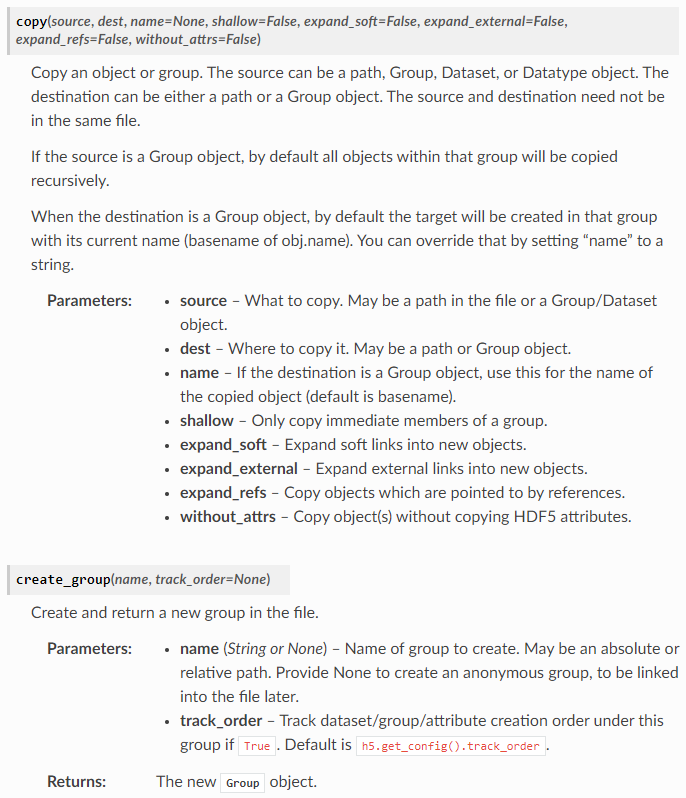
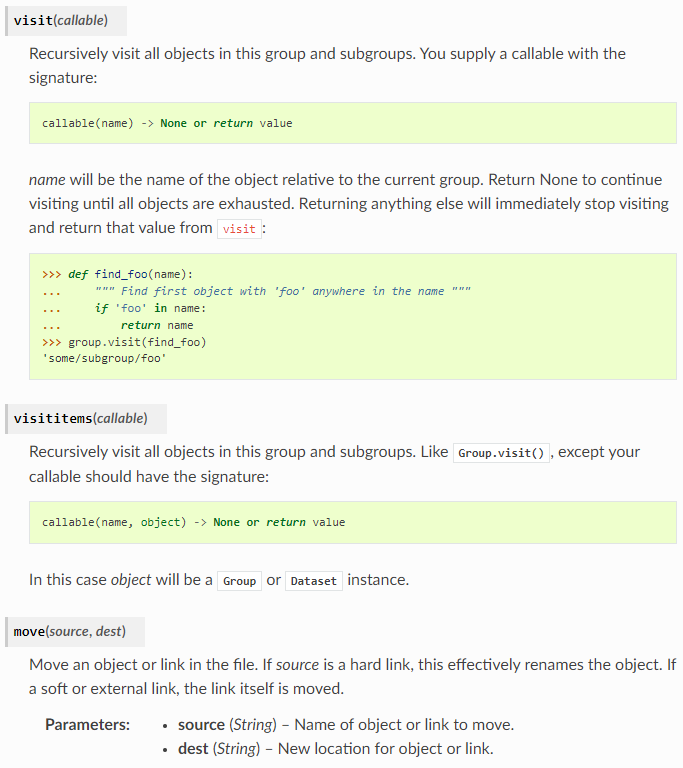


Groups:

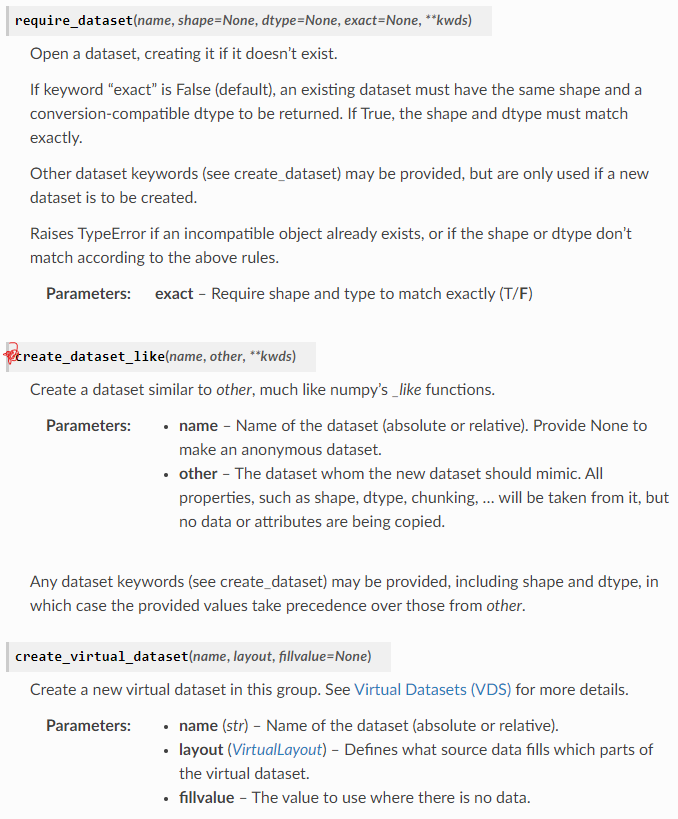
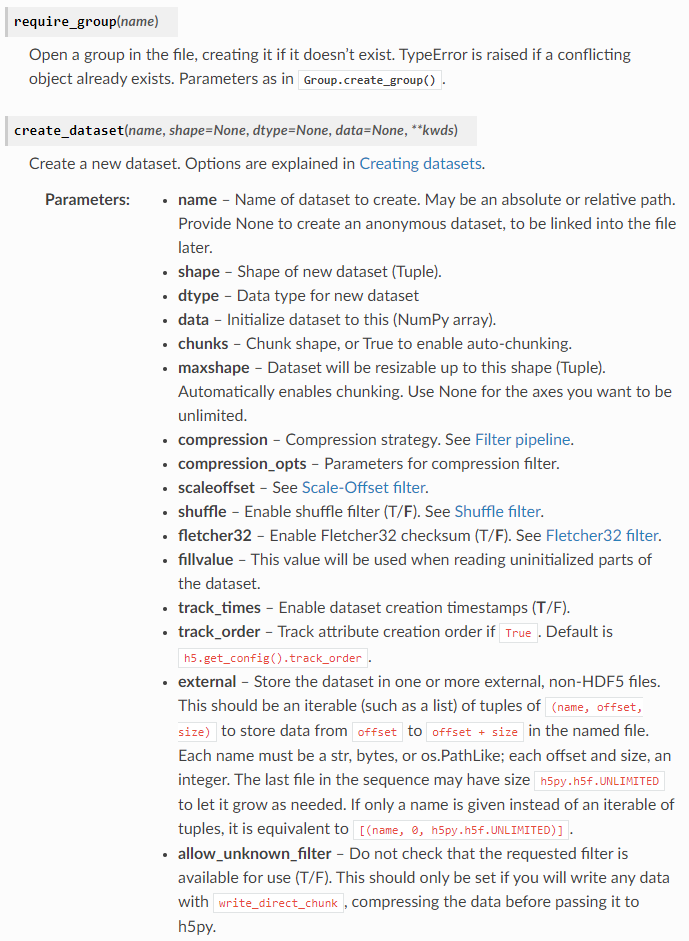
* Groups are the container mechanism by which HDF5 files are organized. From a Python perspective, they operate somewhat like dictionaries. In this case the “keys” are the names of group members, and the “values” are the members themselves ([**Group**](https://docs.h5py.org/en/latest/high/group.html#h5py.Group) and [**Dataset**](https://docs.h5py.org/en/latest/high/dataset.html#h5py.Dataset)) objects. 
* Assigning an object to a name in a group – hard links
* Soft links 
* 

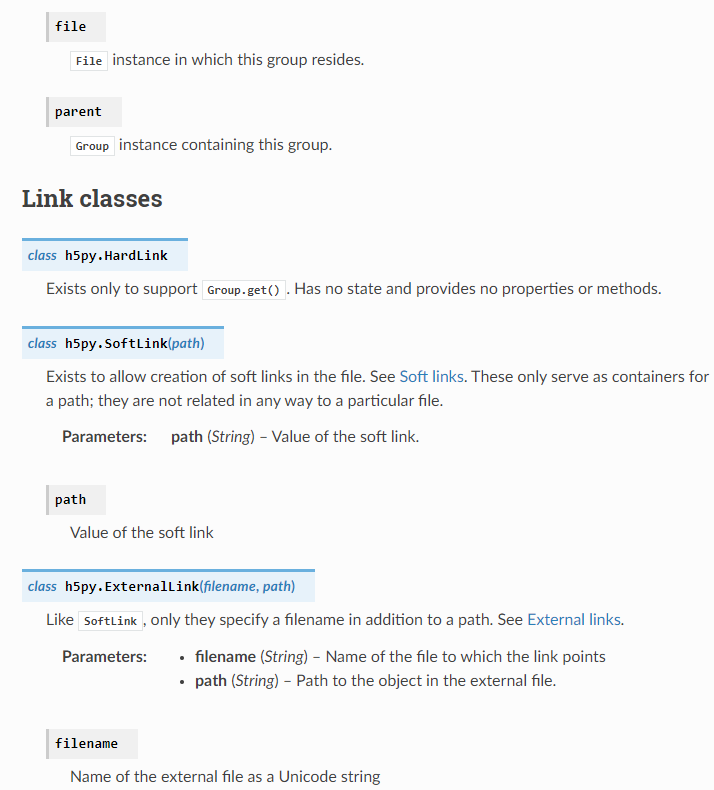
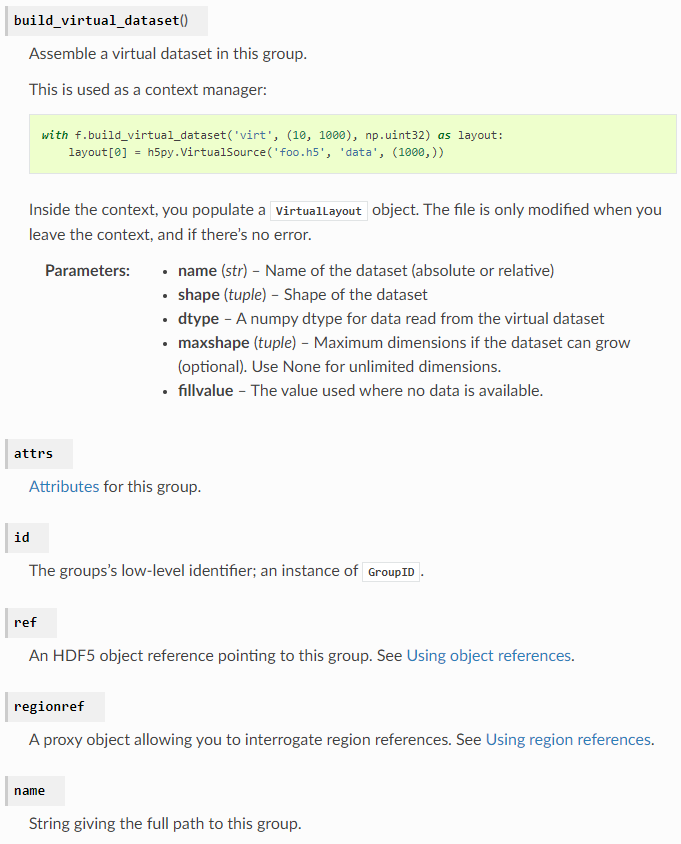










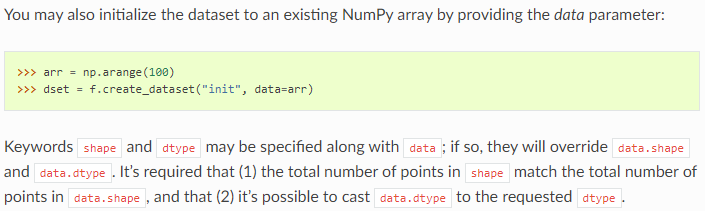


Datasets:

New datasets are created using either **[Group.create\_dataset()](https://docs.h5py.org/en/latest/high/group.html" \l "h5py.Group.create_dataset" \o "h5py.Group.create_dataset)** or **[Group.require\_dataset()](https://docs.h5py.org/en/latest/high/group.html" \l "h5py.Group.require_dataset" \o "h5py.Group.require_dataset)**. Existing datasets should be retrieved using the group indexing syntax (dset = group["name"]).

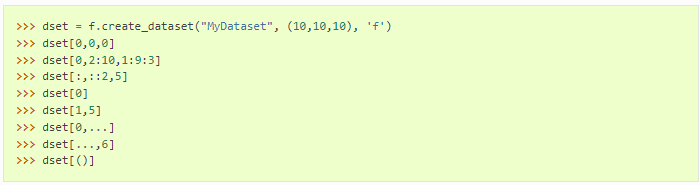
To initialise a dataset, all you have to do is specify a name, shape, and optionally the data type (defaults to 'f'):

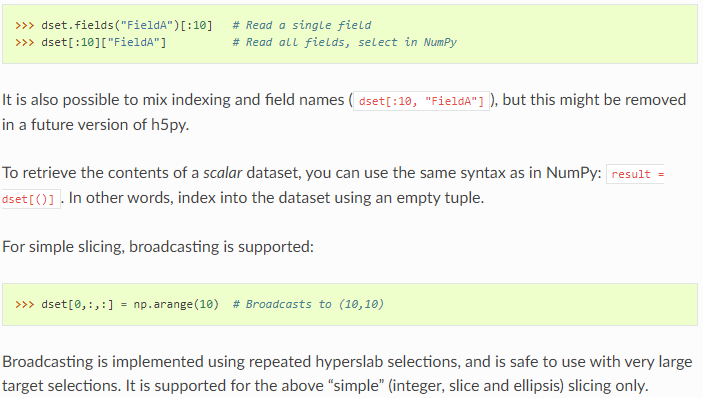


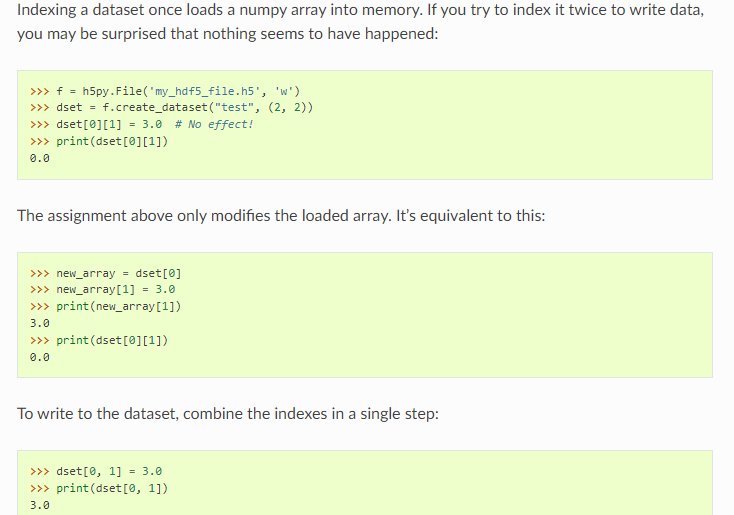


HDF5 datasets re-use the NumPy slicing syntax to read and write to the file. Slice specifications are translated directly to HDF5 “hyperslab” selections, and are a fast and efficient way to access data in the file. The following slicing arguments are recognized:

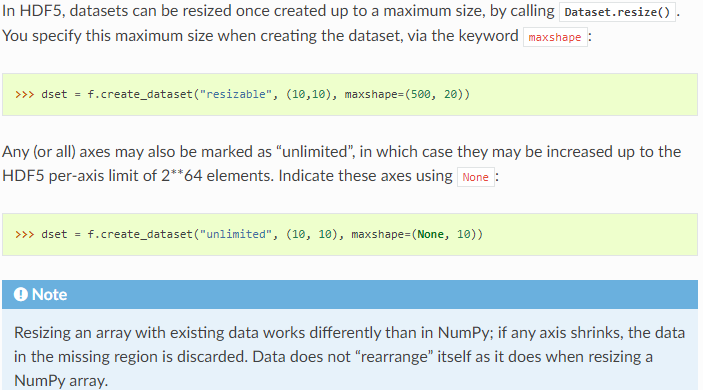
* Indices: anything that can be converted to a Python long
* Slices (i.e. [:] or [0:10])
* Field names, in the case of compound data
* At most one Ellipsis (...) object
* An empty tuple (()) to retrieve all data or *scalar* data



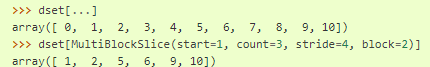




Chunked storage : see website for details

Resizable datasets: 

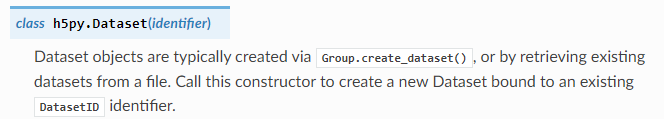
Filter pipeline: see website for compression/zip options

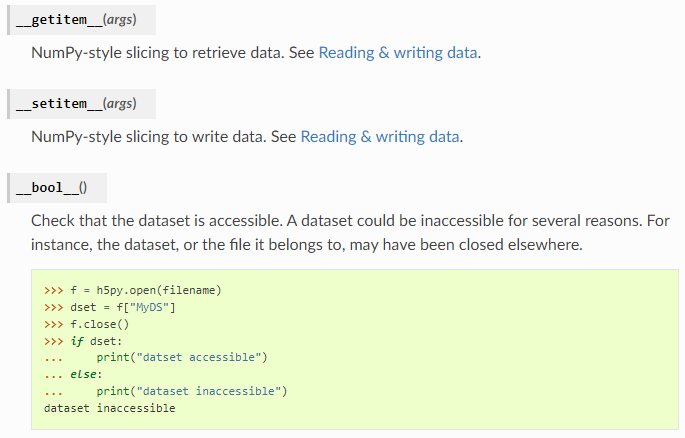
Multi-block selection: 

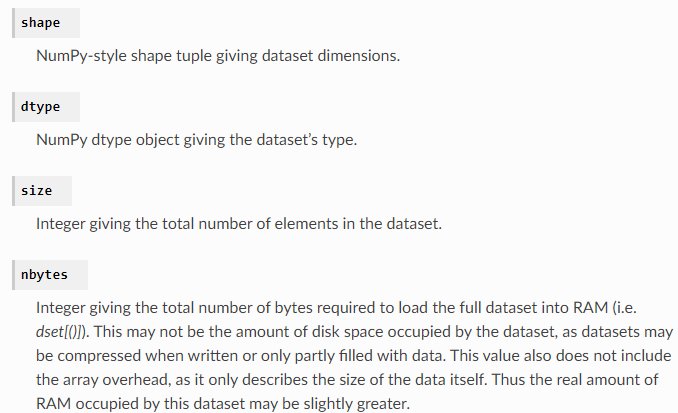
Empty/null datasets:

* HDF5 has the concept of Empty or Null datasets and attributes.
* These are not the same as an array with a shape of (), or a scalar dataspace in HDF5 terms.
* Instead, it is a dataset with an associated type, no data, and no shape.
* In h5py, we represent this as either a dataset with shape None, or an instance of h5py.Empty.
* Empty datasets and attributes cannot be sliced.

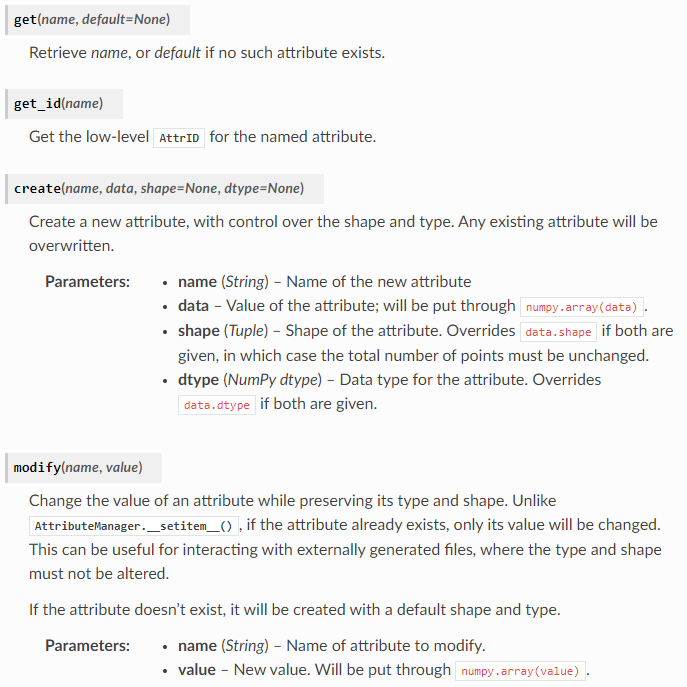
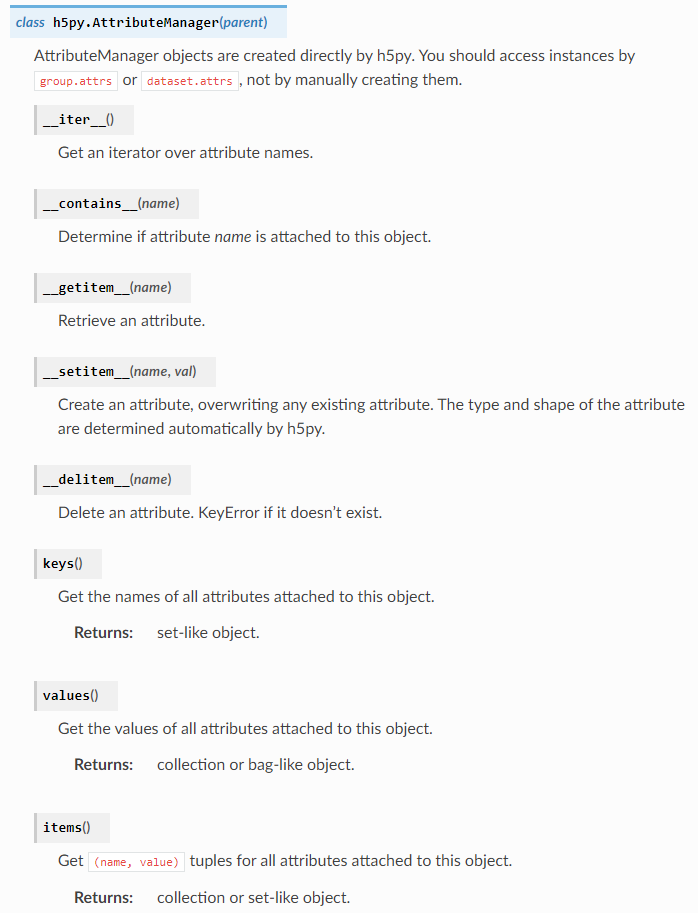




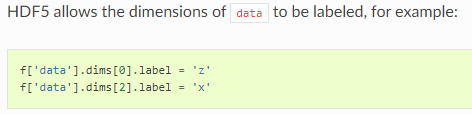




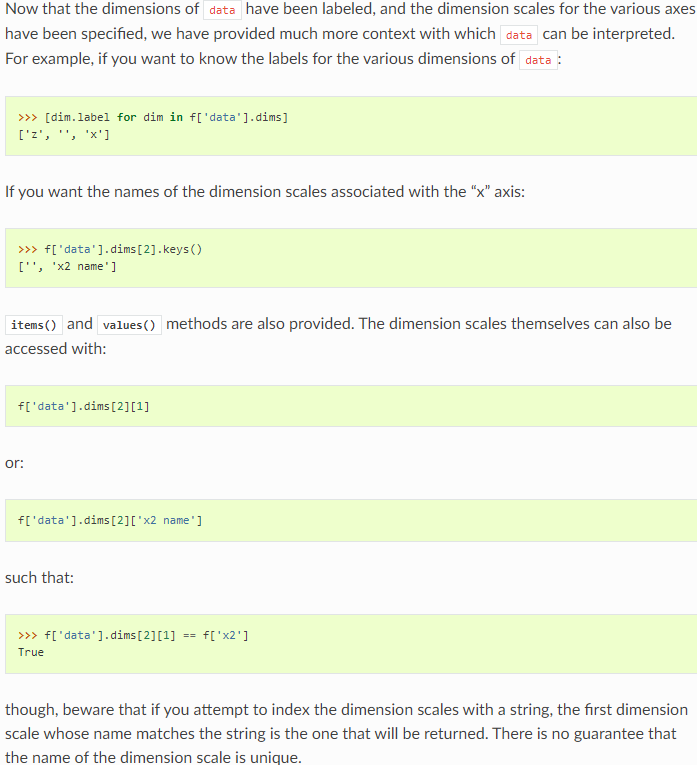
Attributes:



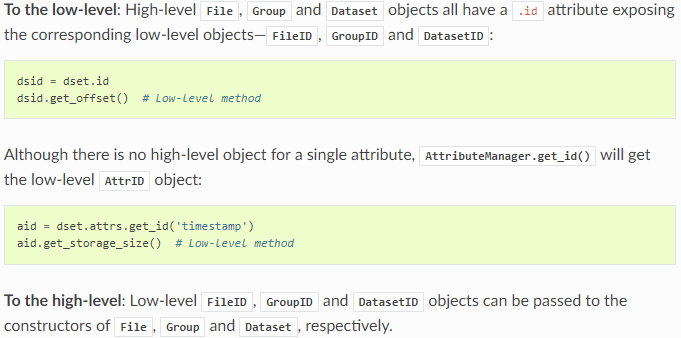
Dimension Scales:







API – id



Library configuration:

A few library options are available to change the behavior of the library. You can get a reference to the global library configuration object via the function **h5py.get\_config()**. This object supports the following attributes:

**complex\_names**

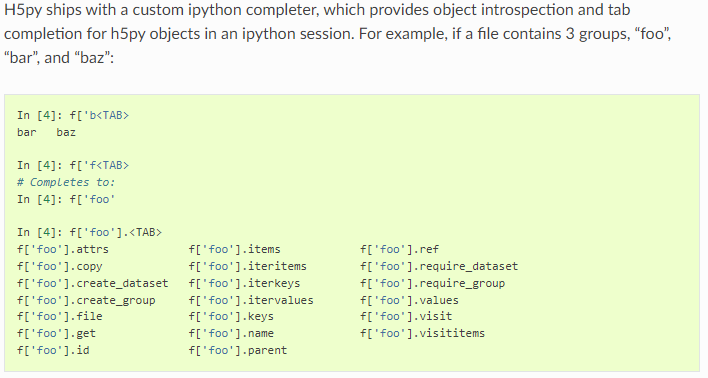
Set to a 2-tuple of strings (real, imag) to control how complex numbers are saved. The default is (‘r’,’i’).

**bool\_names**

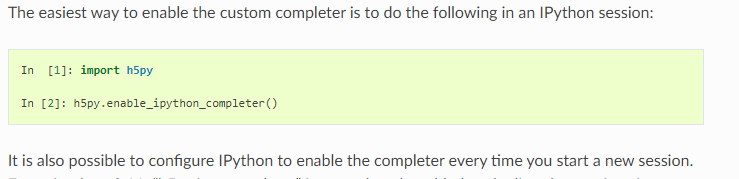
Booleans are saved as HDF5 enums. Set this to a 2-tuple of strings (false, true) to control the names used in the enum. The default is (“FALSE”, “TRUE”).

**track\_order**

Whether to track dataset/group/attribute creation order. If container creation order is tracked, its links and attributes are iterated in ascending creation order (consistent with dict in Python 3.7+); otherwise in ascending alphanumeric order. Global configuration value can be overridden for particular container by specifying track\_order argument to [**h5py.File**](https://docs.h5py.org/en/latest/high/file.html#h5py.File), [**h5py.Group.create\_group()**](https://docs.h5py.org/en/latest/high/group.html#h5py.Group.create_group), [**h5py.Group.create\_dataset()**](https://docs.h5py.org/en/latest/high/group.html#h5py.Group.create_dataset). The default is False.

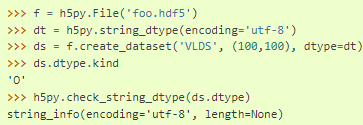


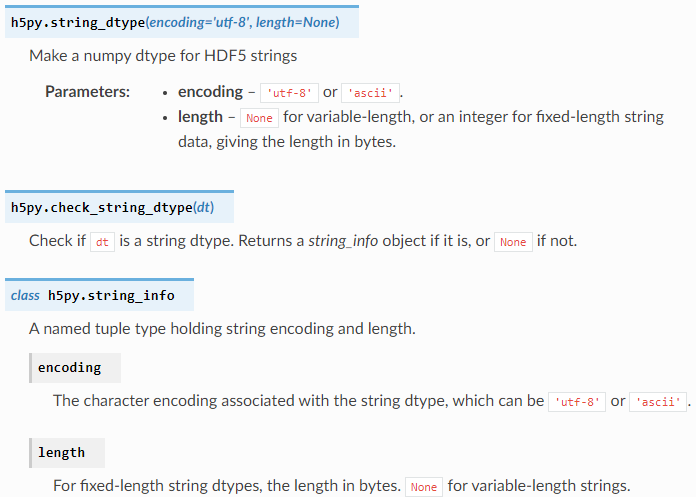
^gives you the options of what you could write

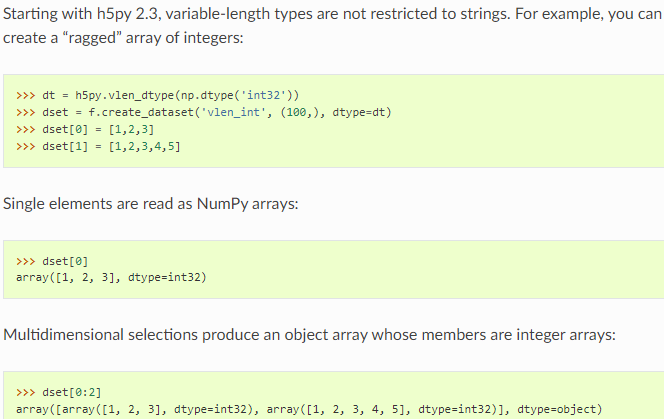


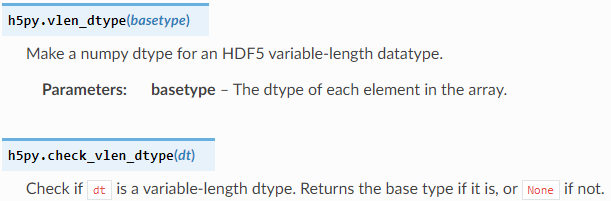
Special types:

Variable length strings

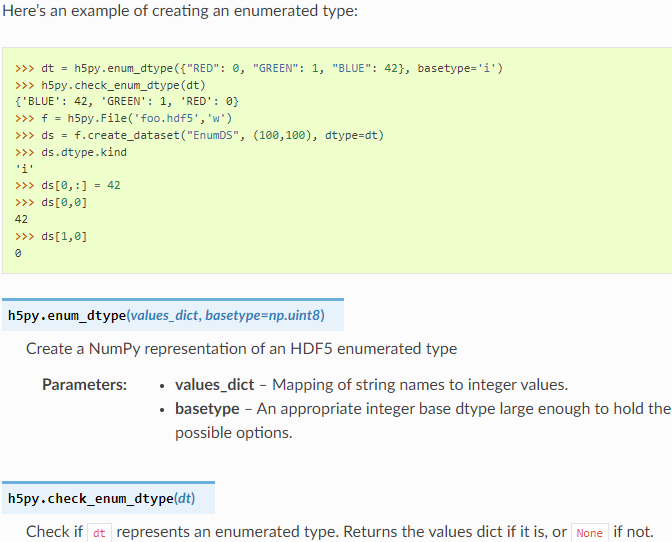
* In HDF5, data in VL format is stored as arbitrary-length vectors of a base type. In particular, strings are stored C-style in null-terminated buffers. NumPy has no native mechanism to support this. Unfortunately, this is the de facto standard for representing strings in the HDF5 C API, and in many HDF5 applications.
* Thankfully, NumPy has a generic pointer type in the form of the “object” (“O”) dtype. In h5py, variable-length strings are mapped to object arrays. A small amount of metadata attached to an “O” dtype tells h5py that its contents should be converted to VL strings when stored in the file.
* Existing VL strings can be read and written to with no additional effort; Python strings and fixed-length NumPy strings can be auto-converted to VL data and stored. 



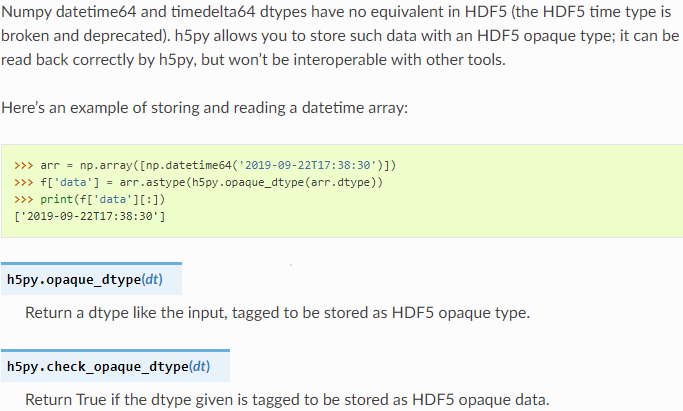




Enumerated type:



Date-time array:



Strings:

String data in HDF5 datasets is read as bytes by default: bytes objects for variable-length strings, or numpy bytes arrays ('S' dtypes) for fixed-length strings. Use **[Dataset.asstr()](https://docs.h5py.org/en/latest/high/dataset.html" \l "h5py.Dataset.asstr" \o "h5py.Dataset.asstr)** to retrieve str objects.

Variable-length strings in attributes are read as str objects. These are decoded as UTF-8 with surrogate escaping for unrecognised bytes.

When creating a new dataset or attribute, Python str or bytes objects will be treated as variable-length strings, marked as UTF-8 and ASCII respectively. Numpy bytes arrays ('S' dtypes) make fixed-length strings. You can use **[string\_dtype()](https://docs.h5py.org/en/latest/special.html" \l "h5py.string_dtype" \o "h5py.string_dtype)** to explicitly specify any HDF5 string datatype.

When writing data to an existing dataset or attribute, data passed as bytes is written without checking the encoding. Data passed as Python str objects is encoded as either ASCII or UTF-8, based on the HDF5 datatype. In either case, null bytes ('\x00') in the data will cause an error.

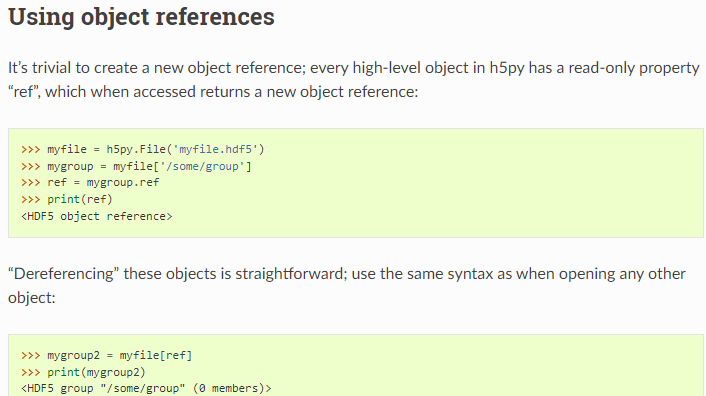
NumPy also has a Unicode type, a UTF-32 fixed-width format (4-byte characters). HDF5 has no support for wide characters. Rather than trying to hack around this and “pretend” to support it, h5py will raise an error if you try to store data of this type.

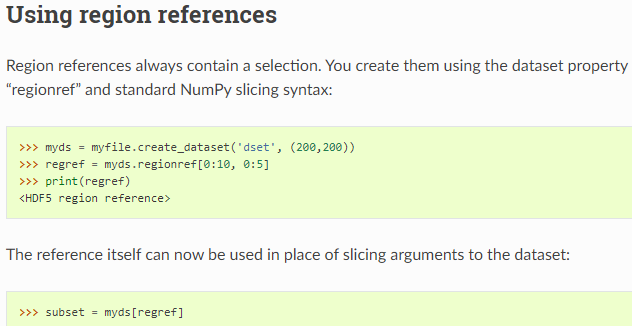
**How to store raw binary data**

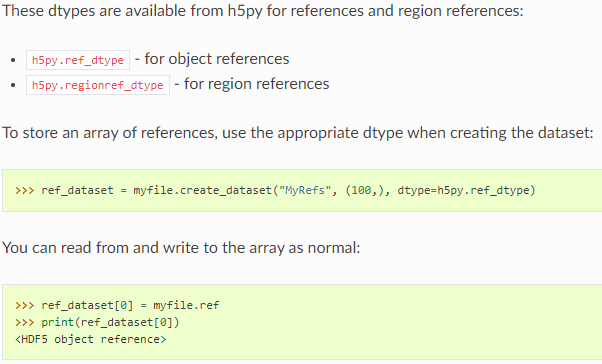
If you have a non-text blob in a Python byte string (as opposed to ASCII or UTF-8 encoded text, which is fine), you should wrap it in a void type for storage. This will map to the HDF5 OPAQUE datatype, and will prevent your blob from getting mangled by the string machinery.

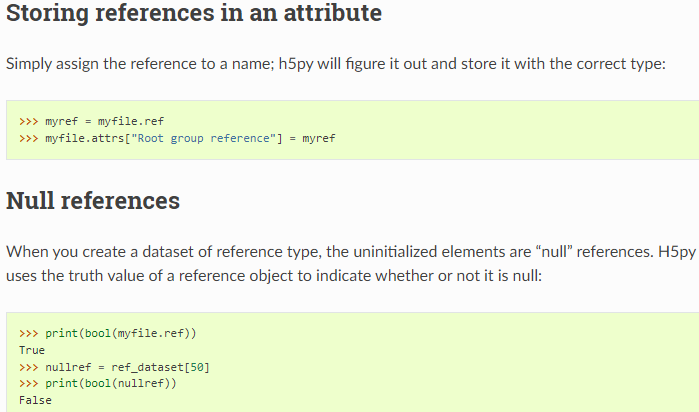
Here’s an example of how to store binary data in an attribute, and then recover it:







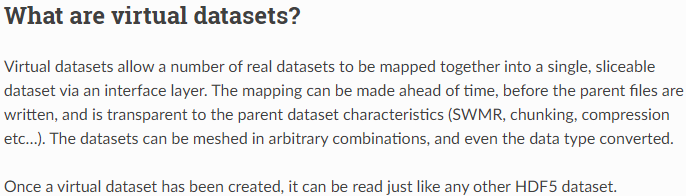


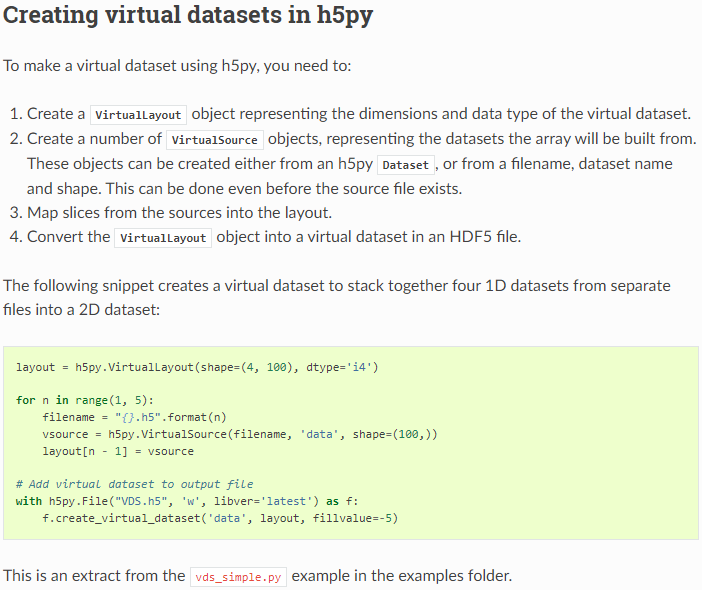


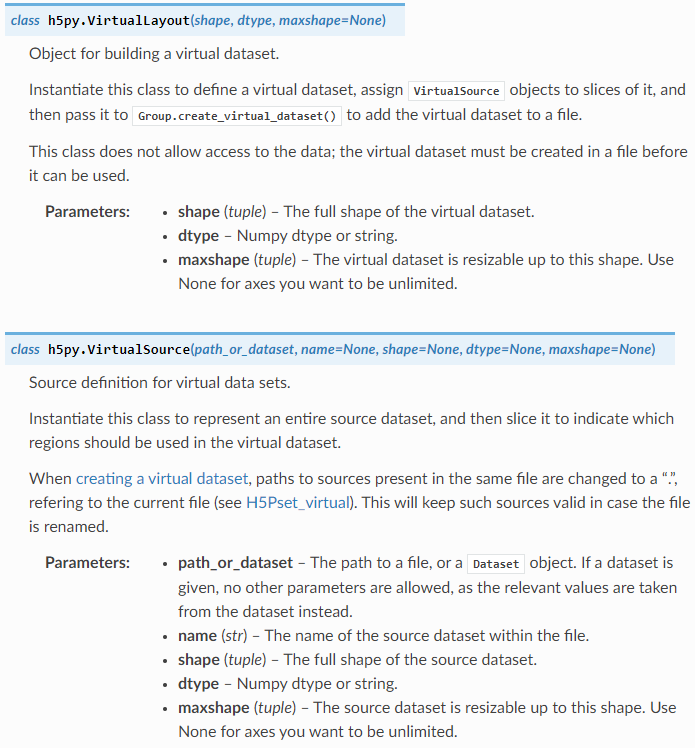
Parallel HDF5: see website for details

Single Writer multiple reader: ll

Virtual datasets:







\*ALL INFO ABOVE IS FROM THE FOLLOWING SOURCE:

Collette, A. (2014). *HDF5 for python¶*. HDF5 for Python - h5py 3.6.0 documentation. Retrieved May 16, 2022, from https://docs.h5py.org/en/latest/index.html

Copyright (c) 2008 Andrew Collette and contributors All rights reserved.