HDF5gateway Documentation

Release 2012-11

Pete Jemian

November 25, 2012

CONTENTS

HDF5gateway makes it easy to read a HDF5 file into an IgorPro folder, including group and dataset attributes, such as a NeXus data file, modify it, and then write it back out.

Contents:

CHAPTER

ONE

HDF5GATEWAY: HDF5 FILE I/O SUPPORT

HDF5gateway makes it easy to read a HDF5 file into an IgorPro folder, including group and dataset attributes, such as a NeXus data file, modify it, and then write it back out.

The goal was to make it easy to read a HDF5 file into an IgorPro folder, including group and dataset attributes, such as a NeXus data file, modify it, and then write it back out. This file provides functions to do just that.

Starting with utilities provided in the *HDF5 Browser* package, this file provides these public functions:

- *H5GW_ReadHDF5(parentFolder, fileName, [hdf5Path])*
- *H5GW_WriteHDF5(parentFolder, newFileName)*
- *H5GW_ValidateFolder(parentFolder)*

and this function which is useful only for testing and development:

- *H5GW_TestSuite()*

Help is provided with each of these functions to indicate their usage.

1.1 Reading

An HDF5 file is read into an IgorPro data folder in these steps:

1. The groups and datasets are read and stored into an IgorPro folder.
2. Any attributes of these groups and datasets are read and assigned to IgorPro objects.

The data file is expected to be in the *home* folder (the folder specified by IgorPro's *home* path), or relative to that folder, or given by an absolute path name.

1.2 Writing

An IgorPro data folder is written to an HDF5 file in these steps:

1. The IgorPro folder is validated for correct structure.
2. The objects in the *HDF5___xref* text wave are written to the HDF5 file.
3. Any folder attributes or wave notes are written to the corresponding HDF5 data path.

The data file is expected to be in the *home* folder (the folder specified by IgorPro's *home* path), or relative to that folder, or given by an absolute path name.

1.3 Validating

Call *H5GW_ValidateFolder(parentFolder)* to test if the *parentFolder* in the IgorPro Data Browser has the proper structure to successfully write out to an HDF5 file by *H5GW_WriteHDF5(parentFolder, newFileName)*.

1.4 Structure of the *HDF5___xref* text wave

It is necessary to devise a method to correlate the name of the same object in the HDF5 file with its representation in the IgorPro data structure. In IgorPro, certain names are reserved such that objects cannot be named. Routines exist to substitute such names on data import to comply with these restrictions. The routine *HDF5LoadGroup* performs this substitution automatically, yet no routine is provided to describe any name substitutions performed.

The text wave, *HDF5___xref*, is created in the base folder of the IgorPro folder structure to describe the mapping between relative IgorPro and HDF5 path names, as shown in the next table. This name was chosen in hopes that it might remain unique and unused by others at the root level HDF5 files.

HDF5___xref wave column plan

column	description
0	HDF5 path
1	Igor relative path

Example

Consider the HDF5 file with datasets stored in this structure:

```
/
  /sasentry01
    /sasdata01
      I
      Q
```

The next table shows the contents of *HDF5___xref* once this HDF5 is read by *H5GW_WriteHDF5()*:

row	`HDF5___xref[row][0]`	`HDF5___xref[row][1]`
0	/	:
1	/sasentry01	:sasentry01
2	/sasentry01/sasdata01	:sasentry01:sasdata01
3	/sasentry01/sasdata01/I	:sasentry01:sasdata01:I0
4	/sasentry01/sasdata01/Q	:sasentry01:sasdata01:Q0

Remember, column 0 is for HDF5 paths, column 1 is for IgorPro paths.

On reading an HDF5 file, the *file_name* and *file_path* are written to the wave note of *HDF5___xref*. These notations are strictly informative and are not used further by this interface. When writing back to HDF5, any wave notes of the *HDF5___xref* wave are ignored.

About *HDF5___xref*:

- Only the folders and waves listed in the *HDF5___xref* text wave will be written to the HDF5 file.
- The *HDF5___xref* text wave is **not written** to the HDF5 file.

When writing an HDF5 file with these functions, based on the structure expected in an IgorPro data folder structure, the *HDF5___xref* text wave is required. Each IgorPro object described must exist as either an IgorPro folder or wave. A wave note is optional. For each such IgorPro object, a corresponding HDF5 file object will be created.

Note: Important! Any IgorPro data storage objects (folders or waves) not listed in *HDF5___xref* **will not be written** to the HDF5 file.

1.5 Groups and Folders

An HDF5 *group* corresponds to the IgorPro *folder*. Both are containers for either data or containers.

In HDF5, a group may have attached metadata known as *attributes*. In IgorPro, folders have no provision to store attributes, thus an optional *Igor___folder_attributes* wave is created. The folder attributes are stored in the wave note of this wave. For more information about attributes, see the discussion of *Attributes and Wave Notes* below.

1.6 Datasets and Waves

Data is stored in HDF5 datasets and IgorPro waves. Both objects are capable of storing a variety of data types with different shapes (rank and length). Of the two systems, IgorPro is the more restrictive, limiting the rank of stored data to four dimensions.

Keep in mind that all components of a single dataset (or wave) are of the same data type (such as 64-bit float or 8-bit int).

In HDF5, a dataset may have attached metadata known as *attributes*. HDF5 attributes are data structures in their own right and may contain data structures. In IgorPro, waves have a provision to store attributes in a text construct called the *wave note*. Of these two, IgorPro is the more restrictive, unless one creates a new wave to hold the data structure of the attributes. For more information about attributes, see the discussion of *Attributes and Wave Notes* below.

The HDF5 library used by this package will take care of converting between HDF5 datasets and IgorPro waves and the user need not be too concerned about this.

1.7 Attributes and Wave Notes

Metadata about each of the objects in HDF5 files and IgorPro folders is provided by *attributes*. In HDF5, these are attributes directly attached to the object (group or dataset). In IgorPro, these attributes are **stored as text** in different places depending on the type of the object, as shown in this table:

object	description
folder	attributes are stored in the wave note of a special wave in the folder named *Igor___folder_attributes*
wave	attributes are stored in the wave note of the wave

Note: IgorPro folders do not have a *wave note*

HDF5 allows an attribute to be a data structure with the same rules for complexity as a dataset except that attributes must be attached to a dataset and cannot themselves have attributes.

Note: In IgorPro, attributes will be stored as text.

An IgorPro wave note is a text string that is used here to store a list of *key,value* pairs. IgorPro provides helpful routines to manipulate such lists, especially when used as wave notes. The IgorPro wave note is the most natural representation of an *attribute* except that it does not preserve the data structure of an HDF5 attribute without additional coding. This limitation is deemed acceptable for this work.

It is most obvious to see the conversion of attributes into text by reading and HDF5 file and then writing it back out to a new file. The data type of the HDF5 attributes will likely be changed from its original type into "string, variable length". If this is not acceptable, more work must be done in the routines below.

1.7.1 IgorPro key,value list for the attributes

Attributes are represented in IgorPro wave notes using a list of *key,value* pairs. For example:

```
NX_class=SASdata
Q_indices=0,1
I_axes=Q,Q
Mask_indices=0,1
```

It is important to know the delimiters used by this string to differentiate various attributes, some of which may have a list of values. Please refer to this table:

separator	char	description
keySep	=	between *key* and *value*
itemSep	,	between multiple items in *value*
listSep	\r	between multiple *key,value* pairs

Note: A proposition is to store these values in a text wave at the base of the folder structure and then use these value throughout the folder. This can allow some flexibility with other code and to make obvious which terms are used.

1.8 Examples

1.8.1 Export data from IgorPro

To write a simple dataset *I(Q)*, one might write this IgorPro code:

```
// create the folder structure
NewDataFolder/O/S root:mydata
NewDataFolder/O sasentry
NewDataFolder/O :sasentry:sasdata

// create the waves
Make :sasentry:sasdata:I0
Make :sasentry:sasdata:Q0

Make/N=0 Igor___folder_attributes
Make/N=0 :sasentry:Igor___folder_attributes
Make/N=0 :sasentry:sasdata:Igor___folder_attributes

// create the attributes
Note/K Igor___folder_attributes, "producer=IgorPro\rNX_class=NXroot"
Note/K :sasentry:Igor___folder_attributes, "NX_class=NXentry"
Note/K :sasentry:sasdata:Igor___folder_attributes, "NX_class=NXdata"
Note/K :sasentry:sasdata:I0, "units=1/cm\rsignal=1\rtitle=reduced intensity"
Note/K :sasentry:sasdata:Q0, "units=1/A\rtitle=|scattering vector|"
```

```
20
21  // create the cross-reference mapping
22  Make/T/N=(5,2) HDF5___xref
23  Edit/K=0 'HDF5___xref';DelayUpdate
24  HDF5___xref[0][1] = ":"
25  HDF5___xref[1][1] = ":sasentry"
26  HDF5___xref[2][1] = ":sasentry:sasdata"
27  HDF5___xref[3][1] = ":sasentry:sasdata:I0"
28  HDF5___xref[4][1] = ":sasentry:sasdata:Q0"
29  HDF5___xref[0][0] = "/"
30  HDF5___xref[1][0] = "/sasentry"
31  HDF5___xref[2][0] = "/sasentry/sasdata"
32  HDF5___xref[3][0] = "/sasentry/sasdata:I"
33  HDF5___xref[4][0] = "/sasentry/sasdata:Q"
34
35  // Check our work so far.
36  // If something prints, there was an error above.
37  print H5GW_ValidateFolder("root:mydata")
38
39  // set I0 and Q0 to your data
40
41  print H5GW_WriteHDF5("root:mydata", "mydata.h5 ")
```

1.8.2 Read data into IgorPro

This is a simple operation, reading the file from the previous example into a new folder:

```
1  NewDataFolder/O/S root:newdata
2  H5GW_ReadHDF5("", "mydata.h5")  // reads into current folder
```

1.9 Public Functions

1.9.1 H5GW_ReadHDF5(parentFolder, fileName, [hdf5Path])

Read the HDF5 data file *fileName* (located in directory *data*, an IgorPro path variable) and store it in a subdirectory of IgorPro folder *parentFolder*.

At present, the *hdf5Path* parameter is not used. It is planned (for the future) to use this to indicate reading only part of the HDF5 file to be read.

String parentFolder Igor folder path (default is current folder)

String fileName name of file (with extension), either relative to current file system directory, or include absolute file system path

String hdf5Path path of HDF file to load (default is "/") :return String: Status: ""=no error, otherwise, error is described in text

1.9.2 H5GW_WriteHDF5(parentFolder, newFileName)

Starting with an IgorPro folder constructed such that it passes the *H5GW_ValidateFolder(parentFolder)* test, write the components described in *HDF5___xref* to *newFileName*.

String parentFolder Igor folder path (default is current folder)

String fileName name of file (with extension), either relative to current file system directory, or include absolute file system path

1.9.3 H5GW_ValidateFolder(parentFolder)

Check (validate) that a given IgorPro folder has the necessary structure for the function H5GW__WriteHDF5_Data(fileID) to be successful when writing that folder to an HDF5 file.

String parentFolder Igor folder path (default is current folder)

return String Status: ""=no error, otherwise, error is described in text

1.9.4 H5GW_TestSuite()

Test the routines in this file using the supplied test data files. HDF5 data files are obtained from the canSAS 2012 repository of HDF5 examples (http://www.cansas.org/formats/canSAS2012/1.0/doc/_downloads/simpleexamplefile.h5).

1.10 Private (static) Functions

Documentation of some, but not all, private functions is provided.

1.10.1 H5GW__OpenHDF5_RW(newFileName, replace)

1.10.2 H5GW__WriteHDF5_Data(fileID)

1.10.3 H5GW__SetHDF5ObjectAttributes(itemID, igorPath, hdf5Path)

1.10.4 H5GW__SetTextAttributeHDF5(itemID, name, value, hdf5Path)

1.10.5 H5GW__make_xref(parentFolder, objectPaths, group_name_list, dataset_name_list, base_name)

Analyze the mapping between HDF5 objects and Igor paths Store the discoveries of this analysis in the HDF5___xref text wave

String parentFolder Igor folder path (default is current folder)

String objectPaths Igor paths to data objects

String group_name_list

String dataset_name_list

String base_name

HDF5___xref wave column plan

column	description
0	HDF5 path
1	Igor relative path

1.10.6 H5GW__addPathXref(parentFolder, base_name, hdf5Path, igorPath, xref, keySep, listSep)

1.10.7 H5GW__addXref(key, value, xref, keySep, listSep)

append a new key,value pair to the cross-reference list

1.10.8 H5GW__appendPathDelimiter(str, sep)

1.10.9 H5GW__findTextWaveIndex(twave, str, col)

Wave/T twave correlation between HDF5 and Igor paths

String str text to be located in column *col*

int col column number to search for *str*

returns int index of found text or -1 if not found

1.10.10 H5GW__OpenHDF5_RO(fileName)

String fileName name of file (with extension), either relative to current file system directory or includes absolute file system path

returns int Status: 0 if error, non-zero (fileID) if successful

Assumed Parameter:

- ***home* (path): Igor path name (defines a file system** directory in which to find the data files) Note: data is not changed by this function

1.10.11 H5GW__HDF5ReadAttributes(fileID, hdf5Path, baseName)

Reads and assigns the group and dataset attributes. For groups, it creates a dummy wave *Igor___folder_attributes* to hold the group attributes.

All attributes are stored in the wave note

Too bad that HDF5LoadGroup does not read the attributes.

int fileID IgorPro reference number for this HDF5 file

String hdf5Path read the HDF5 file starting from this level (default is the root level, "/") Note: not implemented yet.

String baseName IgorPro subfolder name to store attributes. Maps directly from HDF5 path.

1.10.12 H5GW__HDF5AttributesToString(fileID, hdf5_Object, hdf5_Type, [keyDelimiter, keyValueSep, itemDelimiter])

Reads the attributes assigned to this object and returns a string of key=value pairs, delimited by ; Multiple values for a key are delimited by ,

All attributes are stored in the wave note

Too bad that HDF5LoadGroup does not read the attributes.

int fileID IgorPro reference number for this HDF5 file

String hdf5_Object full HDF5 path of object

int hdf5_Type 1=group, 2=dataset

String keyDelimiter between key=value pairs, default = "r"

String keyValueSep key and value, default = "="

String itemDelimiter between multiple values, default = ","

returns str key1=value;key2=value,value;key3=value, empty string if no attributes

1.10.13 H5GW__HDF5AttributeDataToString(fileID, hdf5_Object, hdf5_Type, attr_name, itemDelimiter)

Reads the value of a specific attribute assigned to this object and returns its value.

int fileID IgorPro reference number for this HDF5 file

String hdf5_Object full HDF5 path of object

int hdf5_Type 1=group, 2=dataset

String attr_name name of the attribute

String itemDelimiter if the attribute data is an array, this will delimit the representation of its members in a string

returns String value, empty string if no value

1.10.14 H5GW__SetStringDefault(str, string_default)

String str supplied value

String string_default default value

returns String default if supplied value is empty

1.10.15 H5GW__AppendString(str, sep, newtext)

String str starting string

String sep separator

String newtext text to be appended

returns String result

1.10.16 H5GW__FileExists(file_name)

String file_name name of file to be found

returns int 1 if exists, 0 if does not exist

1.11 Testing and Development

Examples to test read and write:

```
print H5GW_ReadHDF5("root:worker", "simpleexamplefile.h5")
print H5GW_ReadHDF5("root:worker", "simple2dcase.h5")
print H5GW_ReadHDF5("root:worker", "simple2dmaskedcase.h5")
print H5GW_ReadHDF5("root:worker", "generic2dqtimeseries.h5")
print H5GW_ReadHDF5("root:worker", "generic2dtimetpseries.h5")
print H5GW_WriteHDF5("root:worker:simpleexamplefile", "test_output.h5")
```

1.11.1 H5GW__TestFile(parentDir, sourceFile)

Reads HDF5 file sourceFile into a subfolder of IgorPro folder parentDir. Then writes the structure from that subfolder to a new HDF5 file: `"test_"+sourceFile` Assumes that sourceFile is only a file name, with no path components, in the present working directory. Returns the name (string) of the new HDF5 file written.

String parentDir folder within IgorPro memory to contain the HDF5 test data

String sourceFile HDF5 test data file (assumes no file path information prepends the file name)

CHAPTER

TWO

ADMINISTRATIVE MATTERS

2.1 Author

Author Pete R. Jemian jemian@anl.gov

2.2 Documentation

html http://subversion.xray.aps.anl.gov/admin_small_angle/HDF5gateway

pdf `HDF5gateway.pdf` (http://subversion.xray.aps.anl.gov/admin_small_angle/HDF5gateway/HDF5gateway.pdf)

2.3 Downloads

IgorExchange

`--tba--`

Just the IgorPro procedure file (text): http://subversion.xray.aps.anl.gov/admin_small_angle/HDF5gateway/HDF5gateway.ipf

subversion checkout

```
svn co https://subversion.xray.aps.anl.gov/small_angle/hdf5gateway/trunk hdf5gateway
```

2.4 Changes

version	date	comments
0.1	2012-11-24	development version for user testing
1.0	2012-11-26	initial production version

2.5 Building the documentation

The documentation for *HDF5gateway* is built from .rst files and from content in the *hdf5gateway.ipf* IgorPro procedure file by a Python script called *extractor.py*, located in the same directory.

The current documentation was built: November 25, 2012.

Required:

- Python
- Sphinx
- LaTeX

How to build the documentation

1. change to the directory with the file *extractor.py*
2. extract the docs from the .ipf file and build the HTML docs:

   ```
   python extractor.py
   ```

3. build the LaTeX and then PDF files:

   ```
   make latexpdf
   ```

4. copy the PDF file to the source directory and rebuild the HTML:

   ```
   cp _build/latex/HDF5gateway.pdf ./
   python extractor.py
   ```

5. copy the rebuilt HTML directory to the publishing space

   –tba–

INDEX