HDF Compass Documentation

Release 0.6.0b7

The HDF Group

June 13, 2016

Contents

This document provide information about the HDF Compass application.

CHAPTER 1

In Brief

HDF Compass is an experimental viewer program for HDF5 and related formats, designed to complement other more complex applications like HDFView. Strong emphasis is placed on clean minimal design, and maximum extensibility through a plugin system for new formats.

HDF Compass is written in Python, but ships as a native application on Windows, OS X, and Linux, by using PyInstaller and Py2App to package the app.

CHAPTER 2

Contents

2.1 Requirements

TBD

2.2 License

2.2.1 Application License

Copyright Notice and License Terms for HDF Compass - Viewer for HDF5 and other file formats

---

Redistribution and use in source and binary forms, with or without modification, are permitted for any purpose (including commercial purposes) provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or materials provided with the distribution.
3. In addition, redistributions of modified forms of the source or binary code must carry prominent notices stating that the original code was changed and the date of the change.
4. All publications or advertising materials mentioning features or use of this software are asked, but not required, to acknowledge that it was developed by The HDF Group and credit the contributors.
5. Neither the name of The HDF Group, nor the name of any Contributor may be used to endorse or promote products derived from this software without specific prior written permission from The HDF Group or the Contributor, respectively.

DISCLAIMER: THIS SOFTWARE IS PROVIDED BY THE HDF GROUP AND THE CONTRIBUTORS “AS IS” WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. In no event shall The HDF Group or the Contributors be liable for any damages suffered by the users arising out of the use of this software, even if advised of the possibility of such damage.

2.2.2 Additional Copyright and License Information

Copyright and license terms for the following software can be found in the adjacent subdirectory Additional_Legal/. This information was obtained from sources provided by each software package provider at the location specified under "Original source:" and was current as of 19 December 2014.

HDF Compass uses selected icons from the following icon sets:

KDE Oxygen icon set Provided by: KDE Community Copyright and license information: additional_legal/KDE_Oxygen_Icon_Set_Copyright_and_License.txt Original source: http://www.gnu.org/copyleft/lesser.html (via link from https://techbase.kde.org/Projects/Oxygen/Licensing) License type: GNU LGPL version 3 license

HydroPro icon set Provided by: Ben Fleming via MediaDesign Copyright and license information: additional_legal/HydroPro_Icons_Terms.txt Original source: http://www.iconarchive.com/icons/media-design/hydropro/readme.txt License type: Freeware

Pre-built HDF Compass binaries include the following software. None of this software is present in whole or in part in the HDF Compass source code.

Python interpreter Provided by: Python Software Foundation Copyright and license information: additional_legal/Python_Copyright_and_License.txt Original source: https://docs.python.org/3/license.html License type: BSD-style license

wxPython GUI layer Provided by: Julian Smart, Vadim Zeitlin, Stefan Csomor, Robert Roebling, and other members of the wxWidgets team Copyright and license information: additional_legal/wxWidgets_Copyrights_and_Licenses.txt Original source: http://docs.wxwidgets.org/3.0/page_copyright.html License type: GNU LGPL version 2 license with exception

PyInstaller runtime support code (Windows & Linux only) Provided by: PyInstaller Development Team Copyright and license information: additional_legal/PyInstaller_Copyrights_and_Licenses.txt Original source: https://github.com/pyinstaller/pyinstaller/blob/develop/COPYING.txt License type: GNU GPL version 2 license with exception

HDF5 for Python (h5py) Provided by: Andrew Collette and contributors Copyright and license information: additional_legal/h5py_Copyrights_and_Licenses.txt Original source: http://docs.h5py.org/en/latest/licenses.html License type: BSD-style license

HydrOffice BAG (hydroffice.bag) Provided by: G.Masetti, B.R.Calder, and contributors Copyright and license information: additional_legal/hydroffice_bag_Copyrights_and_Licenses.txt Original source: https://bitbucket.org/ccomjhc/hyo_bag/raw/tip/COPYING.txt License type: BSD-style license

NumPy Provided by: NumPy Developers Copyright and license information: additional_legal/NumPy_Copyright_and_License.txt Original source: http://www.numpy.org/license.html License type: BSD-style license

matplotlib Provided by: Matplotlib Development Team Copyright and license information: additional_legal/matplotlib_Copyright_and_License.txt Original source: http://matplotlib.org/users/license.html License type: BSD-style license

PyDAP Provided by: Roberto De Almeida Copyright and license information: additional_legal/PyDAP_Copyright_and_License.txt Original source: http://www.pydap.org/license.html License type: BSD-style license

2.3 How to install

TBD

2.4 How to use

TBD

2.5 How to contribute *[developer]*

TBD

2.6 Data Model *[developer]*

2.6.1 Introduction

This document describes the publically accessible data model package which is used by HDFCompass to display objects in a file, OpenDAP server or other resource.

The data model is implemented as a collection of classes in a top-level Python package, `compass_model`, which is completely independent of the GUI code. It has no dependencies beyond the Python standard library. This makes it possible to develop and test new plugins independently of GUI development; in particular, the automated Python unit-testing framework can be used, which is impossible for code that depends on the GUI.

The classes in `compass_model` are abstract, and define a standard interface for objects like containers, regular multidimensional arrays, images, and key/value stores. "Plug-ins" consist of concrete implementations which satisfy this interface. For example, the built-in HDF5 plugin which ships with HDFCompass implements a `Group` class which inherits from `compass_model.Container`, and a Dataset class which inherits from `compass_model.Array`.

The GUI has a collection of viewers which can display any object following the interfaces defined in `compass_model`. For example, `compass_model.Container` implementations are displayed in a browser-style view, with list, icon, and tree displays possible. Likewise, `compass_model.Array` implementations are displayed in a spreadsheet-like view, with facilities for plotting data.

Multiple concrete classes can handle the same object in a file. For example, an HDF5 image is implemented as a dataset with special attributes. Three classes in the HDF5 plugin are capable of handling such an object, which inherit respectively from `compass_model.Image`, `compass_model.Array`, and `compass_model.KeyValue`; the last represents HDF5 attributes.

When an icon in the GUI is double-clicked, the default (last-registered) class is used to open to object in the file. The other classes are made available in a context menu, for example, if the user wants to open an image with the Array viewer or see the HDF5 attributes.

Numeric types (integers, floats, multidimensional data) are handled with the NumPy type model, which can represent nearly all formats. Python strings (byte and Unicode) are also supported.

2.6.2 Data stores

class `Store`

Represents a file or remote resource which contains objects. Objects within the store can be retrieved using *keys*,

which may be any hashable Python object. For example, an HDF5 store generally uses string keys representing paths in the file, although object and region reference objects are also valid.

Objects may be retrieved using the __getitem__ syntax (`obj = store[key]`). The retrieved object is a Node instance. The exact class depends on the order in which Node handlers were registered; see the *push* method below.

Methods related to the plugin system

Typically, a model will implement several classes based on the `compass_model` abstract classes such as `Container` or `Array`. This rasises the question: when an object is retrieved from the data store, which class should be used?

The answer is that each `Node` subclass you write should be "registered" with your `Store` subclass, and have a static method called `canhandle`. When an object is opened, by default the most recently registered class which reports it can understand the object is used.

All registered subclasses may be retrieved via the `gethandlers` function, which an optionally request that only subclasses capable of handling *key* be returned. This is the basis for the right-click "Open As" menu in the GUI.

classmethod `Store.`**`push`**(*nodeclass*)
Register a Node subclass. These are kept in a list inside the class; when an object is retrieved from the store, the first class for which *nodeclass.canhandle(store, key)* returns True is instantiated and returned.

`Store.`**`__getitem__`**(*key*)
Open an object in the store, using the last registered class which reports it can open the key.

`Store.`**`gethandlers`**(*key=None*)
Retrieve all registered Node subclasses, optionally filtering by those which report they can handle *key*.

`Store.`**`__contains__`**(*key*)
(Abstract) True if *key* is exists in the store, False otherwise.

Other methods & properties

static `Store.`**`canhandle`**(*url*)
(Abstract) Return True if this class can make sense of *url*, False otherwise. This method is used by the GUI when determining which Store implementation to use when opening a file. For example, the HDF5 plugin uses `h5py.is_hdf5(filename)`.

`Store.`**`__init__`**(*url*)
(Abstract) Create a new store instance from the data at *url*. URLs are given in the `scheme://locator` fashion. For example, an HDF5 file might be located by `file:///path/to/file.hdf5`.

`Store.close():`
(Abstract) Discontinue access to the data store.

`Store.`**`get_parent`**(*key*)
(Abstract) Return the object which contains *key*, or None if no such object exists.

`Store.`**`url`**
(Abstract) The URL used to open the store

`Store.`**`displayname`**
(Abstract) A short name used for the store (e.g. `basename(filepath)`).

`Store.`**`root`**
(Abstract) A Node instance representing the starting point in the file. For hierarchical formats, this would be the root container. For scalar formats (FITS, for example), this could be e.g. an Array or Image instance.

`Store.file_extensions`
For plugins that support local file access, this is a dictionary mapping file kinds to lists of extensions in "glob" format, e.g. `{'HDF5 File': ['.h5', '.hdf5']}`. This will be used to populate the filter in the file-open dialog, among other things.

2.6.3 Nodes

A "node" is any object which lives in the data store. The Node class defined below is the base class for more interesting abstract classes like containers and arrays. It defines much of the interface.

class `Node`
Base class for all objects which live in a data store.

You generally shouldn't inherit from Node directly, but from one of the more useful Node subclasses in this file. Direct Node subclasses can't do anything interesting in the GUI; all they do is show up in the browser.

`Node.icons`
Class attribute containing a dict for icon support. Keys should be integers giving icon size; values are a callable returning a byte string with PNG image data. Example: `icons = {16: get_png_16, 32: get_png_32}`. Since icons are a pain to handle, default icons are provided by `compass_model` and this attribute is optional.

`Node.classkind`
(Abstract) A short string (2 or 3 words) describing what the class represents. This will show up in e.g. the "Open As" context menu. Example: "HDF5 Image" or "Swath".

static `Node.canhandle`(*store*, *key*)
(Abstract) Determine whether this class can usefully represent the object. Keep in mind that keys are not technically required to be strings.

`Node.__init__(store, key):`
(Abstract) Create a new instance representing the object pointed to by *key* in *store*.

`Node.key`
(Abstract) Unique key which identifies this object in the store. Keys may be any hashable object, although strings are the most common.

`Node.store`
(Abstract) The data store to which the object belongs.

`Node.displayname`
(Abstract) A short name for display purposes (16 chars or so; more will be ellipsized).

`Node.description`
(Abstract) Descriptive string (possibly multi-line).

2.6.4 Containers

class `Container`(*Node*)
Represents an object which holds other objects, like an HDF5 group or a filesystem directory. Implementations will be displayed using the browser view.

`Container.__len__`()
(Abstract) Get the number of objects directly attached to the container.

`Container.__getitem__`(*index*)
(Abstract) Retrieve the node at *index*. Note this returns a Node instance, not a key.

2.6.5 Arrays

class **Array** (*Node*)
The array type represents a multidimensional array, using an interface inspired by Numpy arrays.

Implementations will be displayed in a spreadsheet-style viewer with controls for subsetting and plotting.

`Array.`**shape**
Shape of the array, as a Python tuple.

`Array.`**dtype**
NumPy data type object representing the type of the array.

`Array.`**__getitem__** (*indices*)
Retrieve data from the array, using the standard array-slicing syntax "data = array[idx1, idx2, idx3]. *indices* are the slicing arguments. Only integer indexes and slice objects (representing ranges) are supported.

2.6.6 Key-Value lists

class **KeyValue** (*Node*)
Represents an object which contains a sequence of key: value attributes. Keys must be strings. Values may be Python or NumPy objects. Implementations will be displayed using a list-like control.

`KeyValue.`**keys**
(Abstract) A list containing all (string) keys contained in the object.

`KeyValue.`**__getitem__** (*name*)
(Abstract) Retrieve the value associated with string *name*.

2.6.7 Images

class **Image** (*Node*)
Represents an image. The current interface supports only true-color RGB images with the origin at upper left, although this could easily be extended to more complex formats including RGBA or palette-based images.

Implementations are displayed in an image viewer.

`Image.`**width**
Image width in pixels

`Image.`**height**
Image height in pixels

`Image.`**data**
Image data. Currently RGB, pixel-interlaced.

2.6.8 Top-level functions

One public function is defined in `compass_model`:

push (*storeclass*)
Register a new Store subclass with HDFCompass. When a URL is being opened, the class will be queried (via `storeclass.canhandle`) to see if it can make sense of the URL.

2.7 How to release *[developer]*

2.7.1 Versioning

You need to install `bumpversion`.

Once installed, you can run something like: `bumpversion --allow-dirty --new-version 0.6.0.dev0 patch`.

The above release value must agree with the variable `version` present in the `conf.py` under the *docs* folder.

2.7.2 PyInstaller

For the *HDFCompass.1file.spec* file, you need to verify that the following parameters are passed to the `EXE()` function:

- `console=False`: to avoid that a console window is opened at run-time for standard I/O
- `debug=False`: to avoid that the boot-loader issues progress messages while initializing and starting the bundled app

2.8 How to freeze *[developer]*

TBD

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Symbols

A

C

D

F

G

H

I

K

N

P

R

S

U

W