h5xplorer Documentation

Release 0.1

Nicolas Renaud

May 18, 2018

Contents:

h5xplorer is a modular interface for the vizualization of data stored in HDF5 files. Not only the defalut menu allows you to quickly plot the dataset stored in your hdf5 files but you can quickly create new menu items specfic to your dta and the way you want to visualize it !!! OMG !!!

CHAPTER 1

Installation

The code is hosted on GitHub (https://github.com/DeepRank/h5xplorer)

To install the code:

- clone the repository git clone https://github.com/DeepRank/h5xplorer
- go there `cd h5xplorer`
- install `pip install -e ./`

CHAPTER 2

Tutorial : The simplest app

h5xplorer allows to create an app very quickly thanks to default context menu that are automatically used if nothing else is specified. In a new file enters.

```
>>> #!/usr/bin/env python
>>> from h5xplorer.h5xplorer import h5xplorer
>>> app = h5xplorer()
```

Execute this file and you'll be able to explore our hdf5 file. The default menu is executed in the background. The default menu file teaches us how t build our own context menu. Let's look at it

```
>>> from PyQt5 import QtWidgets
>>> from h5xplorer.menu_tools import *
>>> from h5xplorer.menu_plot import *
>>>
>>> def default_context_menu(self, treeview, position):
>>>
>>>     """Generate a right-click menu for the items"""
>>>
>>>     all_item = get_current_item(self,treeview,single=False)
>>>
>>>     if len(all_item) == 1:
>>>
>>>         item = all_item[0]
>>>         data = get_group_data(get_current_hdf5_group(self,item))
>>>
>>>         if data is None:
>>>             list_operations = ['Print attrs']
>>>
>>>         elif data.ndim == 1:
>>>             list_operations = ['Print attrs','-','Plot Hist', 'Plot Line']
>>>
>>>         elif data.ndim == 2:
>>>             list_operations = ['Print attrs','-','Plot Hist', 'Plot Map']
>>>
```

(continues on next page)

(continued from previous page)

```
>>>         else:
>>>             list_operations = ['Print attrs']
>>>
>>>         action,actions = get_actions(treeview,position,list_operations)
>>>
>>>         if action == actions['Print attrs']:
>>>             send_dict_to_console(self,item,treeview)
>>>
>>>         if 'Plot Hist' in actions:
>>>             if action == actions['Plot Hist']:
>>>                 plot_histogram(self,item,treeview)
>>>
>>>         if 'Plot Line' in actions:
>>>             if action == actions['Plot Line']:
>>>                 plot_line(self,item,treeview)
>>>
>>>         if 'Plot Map' in actions:
>>>             if action == actions['Plot Map']:
>>>                 plot2d(self,item,treeview)
>>>
>>>     elif len(all_item) == 2:
>>>
>>>         item0,item1 = all_item
>>>
>>>         list_operations = ['Plot Scatter','Plot Line']
>>>         action,actions = get_actions(treeview,position,list_operations)
>>>
>>>         if action == actions['Plot Scatter']:
>>>             plot1D(self,item0,item1,treeview,plot='scatter')
>>>
>>>         if action == actions['Plot Line']:
>>>             plot1D(self,item0,item1,treeview,plot='line')
```

CHAPTER 3

Tutorial : A bit more advanced then

CHAPTER 4

Documentation

All the prototypes of the class/methods are here specified

4.1 Get items

Some functionalities are preimplemented to access the items and stuff

h5xplorer.menu_tools.**get_current_item**(*self*, *treeview*, *single=False*)
Get the item(s) that was selected

Parameters

- **treeview** (*HDF5TreeWidget*) – the treeview
- **single** (*bool, optional*) – if true only single item possible

Returns Description

Return type TYPE

h5xplorer.menu_tools.**get_current_hdf5_group**(*self*, *item*)
Get the HDF5 group of the selected item

Parameters **item** (HDF5TreeItem) – treeview item

Returns the corresponding group

Return type HDF5 group

h5xplorer.menu_tools.**get_group_data**(*group*)
Get the dataset of the item

Parameters **group** (*HDF5 group*) – group of the treeview item

Returns return np.array if the group has a dataset or None otherwise

Return type dataset or None

h5xplorer.menu_tools.**get_actions**(*treeview*, *position*, *list_action*)
Generate a singlelevel context menu of action and return the selected one

Example:

```
>>> list_operations = ['Print attrs','-','Plot Hist', 'Plot Map']
>>> action,actions = get_actions(treeview,position,list_operations)
>>> if action == actions['Print attrs']:
        send_dict_to_console(self,item,treeview)
```

Parameters

- **treeview** (*HDF5TreeWidget*) – the treeview
- **position** (*TYPE*) – Description
- **list_action** (*list*) – List of string giving the action names

Returns dict of QTMenu actions actions: the selected action

Return type actions

h5xplorer.menu_tools.**get_multilevel_actions**(*treeview*, *position*, *list_action*, *sub_list*)
Generate a multilevel context menu of action and return the selected one

Example:

```
>>> list_operations = ['Hit Rate','Av. Prec.']
>>> list_subop = [['Train','Valid','Test'],['Train','Valid','Test']]
>>> action,actions = get_multilevel_actions(treeview,position,list_operations,
↪list_subop)
```

Parameters

- **treeview** (*HDF5TreeWidget*) – the treeview
- **position** (*TYPE*) – Description
- **list_action** (*list*) – List of string giving the action names
- **sub_list** (*list*) – list of string giving the subactions

Returns dict of QTMenu actions actions: the selected action

Return type actions

h5xplorer.menu_tools.**send_dict_to_console**(*self*, *item*, *treeview*)
Send a dictionany to the QT console

Parameters

- **item** (HDF5TreeItem) – treeview item
- **treeview** (*HDF5TreeWidget*) – the treeview

h5xplorer.menu_tools.**print_attributes**(*self*, *item*, *treeview*)
Print the attribute in the console

Parameters

- **item** (HDF5TreeItem) – treeview item
- **treeview** (*HDF5TreeWidget*) – the treeview

h5xplorer.menu_tools.**get_user_values**(*varnames*, *vartypes='float'*, *windowtitle='Enter Values'*)

Get the values of variables from the users

Parameters

- **varnames** (*list*) – list of strings of all the desired variables
- **windowtitle** (*str, optional*) – Name of the window

Returns list of float of the desired variables

Return type list

4.2 Plotting routines

Some functionalities are preimplemented to plot dataset in the QT console

h5xplorer.menu_plot.**plot_histogram**(*self*, *item*, *treeview*, *nbins=10*)

Plot an histogram of the data

Example:

```
>>> def context_menu(self,treeview,position):
>>>     all_item = get_current_item(self,treeview,single=False)
>>>     item = all_item[0]
>>>     list_operations = ['Plot Hist']
>>>     action,actions = get_actions(treeview,position,list_operations)
>>>     if action == actions['Plot Hist']:
            plot_histogram(self,item,treeview)
```

Parameters

- **item** (HDF5TreeItem) – treeview item
- **treeview** (*HDF5TreeWidget*) – treeview
- **nbins** (*int, optional*) – number of bins in the histogram

h5xplorer.menu_plot.**plot_line**(*self*, *item*, *treeview*)

Plot a line plot of a single item VS its index

Example:

```
>>> def context_menu(self,treeview,position):
>>>     all_item = get_current_item(self,treeview,single=False)
>>>     item = all_item[0]
>>>     list_operations = ['Plot Line']
>>>     action,actions = get_actions(treeview,position,list_operations)
>>>     if action == actions['Plot Line']:
            plot_line(self,item,treeview)
```

Parameters

- **item** (HDF5TreeItem) – treeview item
- **treeview** (*HDF5TreeWidget*) – treeview

h5xplorer.menu_plot.**plot1D**(*self*, *item0*, *item1*, *treeview*, *plot='line'*)

Plot a XY line or scatter plot of two items

Note: You must be able to select multiple items to use this method. So you must create the app with:

```
>>> app = h5xplorer(extended_selection=True)
```

Example:

```
>>> def context_menu(self,treeview,position):
>>>     all_item = get_current_item(self,treeview,single=False)
>>>     item0 = all_item[0]
>>>     item1 = all_item[1]
>>>     list_operations = ['Plot1D']
>>>     action,actions = get_actions(treeview,position,list_operations)
>>>     if action == actions['Plot1D']:
          plot1D(self,item0,item1,treeview)
```

Parameters

- **item0** (HDF5TreeItem) – treeview item for the X data
- **item1** (HDF5TreeItem) – treeview item for the Y data
- **treeview** (*HDF5TreeWidget*) – treeview
- **plot** (*str, optional*) – 'line' or 'scatter'

h5xplorer.menu_plot.**plot2d**(*self*, *item*, *treeview*)

Plot a map of a 2D data array

Example:

```
>>> def context_menu(self,treeview,position):
>>>     all_item = get_current_item(self,treeview,single=False)
>>>     item = all_item[0]
>>>     list_operations = ['Plot2D']
>>>     action,actions = get_actions(treeview,position,list_operations)
>>>     if action == actions['Plot2D']:
          plot2D(self,item,treeview)
```

Parameters

- **item** (HDF5TreeItem) – treeview item
- **treeview** (*HDF5TreeWidget*) – treeview

4.3 Core design of the app

classes and stuff that the app is built from

4.3.1 Core App

class h5xplorer.h5xplorer.**DummyHDF5Group**(*dictionary*, *attrs={}*, *name='DummyHDF5Group'*)

Bases: dict

Dummy HDF5 group created sometimes ....

Parameters

- **dictionary** (*dict*) –
- **attrs** (*dict, optional*) –
- **name** (*str, optional*) – Name of the group

file = None

parent = None

class h5xplorer.h5xplorer.**HDF5TreeItem**(*data_file*, *parent*, *name*, *row*)

Bases: object

Create a new item for an HDF5 tree

Parameters

- **data_file** (*HDF5 data file*) – This is the file (NB must be the top-level group) containing everything
- **parent** (HDF5TreeItem) – The parent of the current item
- **name** (*string*) – The name of the current item (should be parent.name plus an extra component)
- **row** (*int*) – The index of the current item in the parent's children.

basename

has_children

children

purge_children()

Empty the cached list of children.

h5item

The underlying HDF5 item for this tree item.

class h5xplorer.h5xplorer.**h5xplorer**(*func_menu=<function default_context_menu>*, *baseimport=None*, *extended_selection=False*)

Bases: object

cleanup()

4.3.2 Dialog box

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index