Description of the pencil-code HDF5 files `v0.1`

September 29, 2009

1 Data structure

In the following description, the full pathway for both subgroups and datasets is given for better readability. The subgroups are written with a trailing '/' to be distinguished from the datasets. The attributes relative to groups and datasets are written without the pathway. The different elements are presented in this order: after each subgroup follow its different attributes, then its datasets, then its embedded subgroups.

Please note that this is a first working version (thus the `v0.1`...), intended to be a proposal to be discussed. It should be considered as experimental, and thus shouldn't be expected to be stable (i.e. its format may evolve a lot).

Important : Please update this file immediately when you update the pencil-code HDF5 format, so that this description is always correctly describing the current implementation.

Notes and questions to be discussed are written in red. I suggest to increase the version number of the HDF5 file to `1.0` once we will have agreed and get rid of all these notes.

2 Example of use

All the following example are given in python, using the module `h5py`. This module was chose rather than the other implementation (`python-tables`) for being simple of use through the high-level component, while at the same time offering access to the almost entire HDF5 C API through the low-level components. There is nothing python-specific, so that the created HDF5 files will (should...) be readable by any other HDF5 implementation, independently of the chosen language. Note that for the moment, direct access to the HDF5 must be done (using high-level compounds of h5py module), but I intend to write a python interface to access and modify these data.

All the following example are supposed to be written on the python command line, after having imported the HDF5 module by:

```
>>> import h5py
>>>
```

The ">>>" corresponds to the python prompt, and thus indicate the commands typed by the user. The lines without the prompt are the answers returned by python.

2.1 Accessing a file

A file may be open by the command `h5py.File`. Different informations can be obtained from the returned object.

⟶ '/' : Root of the file

    ⟶ 'name' : String Attribute. Set to "PencilCode". This attribute is intended to describe the type of data stocked there, i.e. pencil code data. Any informations concerning the run itself are intended to be stocked in the respective subgroups.

    ⟶ 'ver' : String Attribute. Version of the pencil-code HDF5 file. The format described in this file corresponds to the version v0.1.

    ⟶ 'dateC' : String Attribute. Date of creation of the file. All dates are formatted as a string typeset as 'DD/MM/YY'. Maybe there are specific type format to stock dates, but I couldn't find it.

    ⟶ 'dateM' : String Attribute. Date of last modification of the file.

    ⟶ '/param/' : Subgroup intended to stock all the parameters data relative to the runs. I assume that all the relevant parameters can be found in params.log in datafile. The interest of reading the parameters there is that we have not only all the parameters obtained during init, but also before *each* run, so that all parameters changes made after each run can be found.

        ⟶ '/param/init/' : Subgroup containing the parameters passed during the init phase.

            ⟶ '/param/init/init_pars/' : Subgroup of the parameters of init_pars

                ⟶ '/param/init/init_pars/cvsid' : String dataset of dimension (1,), non-resizable

                ⟶ '/param/init/init_pars/ip' : Integer dataset of dimension (1,), non-resizable

                ⟶ '/param/init/init_pars/xyz0' : Float dataset of dimension (3,), non-resizable

                ⟶ '/param/init/init_pars/...' : All the parameters will be added. It has been chosen to list each parameter in separate datasets rather than in a single large dataset array, because it gather different types (and sometimes even arrays of data)... but maybe this should be discussed, and a single array would be better ?

            ⟶ '/param/init/hydro_init_pars/' : Subgroup of the parameters of hydro_init_pars

                ⟶ '/param/init/hydro_init_pars/inituu' : String dataset of dimension (1,), non-resizable

                ⟶ '/param/init/hydro_init_pars/...' : etc.

            ⟶ '/param/init/..._init_pars/' : etc. Do you think that we should create the subgroups in all cases, or only in the cases there are referred to in the pencil code files (i.e. when they are really needed for the run). In the present implementation, I chose to only add the relevant subgroups, rather than add empty one.

        ⟶ '/param/run/' : Subgroup containing the parameters passed during each run phase. in all the arrays, the first dimension corresponds to the number of the run.

            ⟶ '/param/run/timerun' : Float dataset of dimension (nbrun,1), first dimension resizeable, nbrun being the number of run that were realized. Note that this parameter do note appear as a pencil code parameter, but is written in the log file each time a new run is launched as a continuation from preceding runs.

            ⟶ '/param/run/run_pars/' : Subgroup of the parameters of run_pars

                ⟶ '/param/run/run_pars/nt' : Integer dataset of dimension (nbrun,1), first dimension resizable

                ⟶ '/param/init/run_pars/...' : etc.

            ⟶ '/param/run/chemistry_run_pars/' : Subgroup of the parameters of chemistry_run_pars

                ⟶ '/param/run/chemistry_run_pars/mobility' : Float dataset of dimension (nbrun,nbchem), first dimension resizable. nbchem is the number of chemical species.

                ⟶ '/param/init/chemistry_run_pars/...' : etc.

                ⟶ '/param/run/..._run_pars/' : etc.

    ⟶ '/data/' : Subgroup intended to stock all the data resulting from the runs.

        ⟶ '/data/time_series' : Float dataset of the time series, of dimension (timeslices, nbvar), timeslices corresponds to an output every /param/run/run_pars/it1 time steps, and nbvar corresponds to the number of diagnostic variables entered in the file print.in

        ⟶ '/data/slices' : Float dataset of the slices, of dimension (timeslices2,x,y), timeslices2 corresponds to an output every /param/run/run_pars/dvid unit of time, and x,y are the grid dimension in the chosen slice.

        ⟶ '/data/var' : Float dataset of the snapshots, of dimension (timeslices3,x,y,z), timeslices3 corresponds to an output every /param/run/run_pars/dsnap unit of time, and x,y,z is the dimension of the whole grid

    ⟶ '/notes/' : Subgroup intended to stock some additional informations about the runs.

        ⟶ '/notes/note' : Dataset of Variable Length Strings. Dimension: resizable (1,). This dataset is intended to stock written notes about the run. Each note can be of arbitrarily long size, and a arbitrary number of notes can be added. More precisely, each note is in HDF5 VL string format (VL standing for Variable Length), and the dataset is a resizable array of VL strings, of dimension one, initially created with one (empty) string element.

        ⟶ '...' : This subgroup should be left open to any personal modifications, so that any additional datasets or subgroups can be entered there as wanted.

Figure 1: Data structure of a pencil-code HDF5 file

```
>>> f=h5py.File("datafile.hdf5","a")
>>> f
<HDF5 file "datafile.hdf5" (mode a, 3 root members)>
>>> f.attrs
<Attributes of HDF5 object "/" (4)>
>>> f.keys()
['data', 'notes', 'param']
>>>
```

Be careful, all the modifications that are made to a file are buffered. It is necessary to either flush or close the file if one wants to be sure that everything has been written on the disk.

```
>>> f.flush()
>>> f.close()
>>>
```

2.2   Reading the file attribute

Attributes of the file can be directly read, so that we can check that this is a pencil code data file, which version of file format it is, and when it was created and last modified:

```
>>> f.attrs["name"]
'PencilCode'
>>>> f.attrs["ver"]
'v0.1'
>>> f.attrs["dateC"]
'28/09/09'
>>> f.attrs["dateM"]
'28/09/09'
>>>
```

Note that one can easily modify these attributes, or to create to one. It is advisable to not do so !

2.3   notes

Initially only f['notes/note'][0] exists, and it is set to the empty string '':

```
>>> f['/notes/note'][0]
''
>>> f['/notes/note'][1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  (... snip here verbose error messages ...)
ValueError: Index (1) out of range (0-0)
>>>
```

Any data can be entered in the note, and additional notes can be created:

```
>>> f['notes/note'][0]="This is a test file"
>>> f['notes/note'].resize((2,))
>>> f['notes/note'][1]="This is an additional note"
>>> f['notes/note'][1]
'This is an additional note'
```

```
>>> f['notes/note'][...]
array([This is a test file, This is an additional note], dtype=object)
>>>
```

Object can be used to directly access to the notes:

```
>>> note=f['notes/note']
>>> note[0]
'This is a test file'
>>> note
<HDF5 dataset "note": shape (2,), type "|O4">
>>>
```

3   Filters

Discussing the possible filters. Should be either possibility of manual tuning, or fastest (null=no compression, data in order), fast (shuffle+lzf=compression with optimization of access time) or small (shuffle+gzip=best compression available). See benchmark on `http://h5py.alfven.org/lzf/`.