pyCGNS.MAP/Manual

Release 4.0.1

Marc Poinot

September 09, 2010

CONTENTS

The MAP module is part of pyCGNS Python package MAP loads and saves CGNS/HDF5 files as Python trees.

CHAPTER

ONE

QUICK START

The MAPper is a module implementing the SIDS-to-Python CGNS mapping. The MAP module loads and saves CGNS/HDF5 files as Python trees.

A simple exemple to load a *CGNS/HDF5* file as a *CGNS/Python* tree:

```
import CGNS.MAP

(tree,links)=CGNS.MAP.load("./T0.cgns",CGNS.MAP.S2P_FOLLOWLINKS)
```

The `tree` value contains the actual CGNS/Python tree with linked-to files included (because the `S2P_FOLLOWLINKS` flag is *on*) and the `links` value is a list of links found during the *HDF5* file parse.

CHAPTER

TWO

USER INTERFACE

MAP is a lightweight module, its purpose is to be as small as possible in order to be embedded separatly in an application (see *Embbeded MAP* .

2.1 Functions

There are two functions: the `load` and the `save`. The `load` reads a CGNS/HDF5 file and produces a CGNS/Python tree. The `save` takes a CGNS/Python tree and writes the contents in a CGNS/HDF5 file:

```
(tree,links)=CGNS.MAP.load(filename,flags,threshold,depth,path)

status=CGNS.MAP.save(filename,tree,links,flags,threshold,depth,path)
```

The arguments and the return values are:

- tree The `tree` is the list representing the CGNS/Python tree. The structure of a `tree` list is detailled in *SIDS-to-Python*. There is no link information in this tree either for *load* or for *save*.

  During the *load*, the links are silently replaced by the linked-to tree they are referring. The `links` value keeps track of these link references found while parsing the CGNS/HDF5 file.

  During the *save*, the tree is splitted into separate files/nodes depending on the references found in the `links` value.

- links The `links` is a list with the link node information. It is returned by a *load* and used as command parameters during the *save*. You can write your own `links` list or change the list you obtain after a *load*. The structure of a `links` list is detailled in *SIDS-to-Python*.

2.2 Flags

The flags are integers that can be OR-ed or XOR-ed to set/unset specific behavior during the load and the save. The boolean operators are used for the flag settings:

```
flags=CGNS.MAP.S2P_FOLLOWLINKS|CGNS.MAP.S2P_TRACE

flags =flags&~CGNS.MAP.S2P_TRACE
flags&=~CGNS.MAP.S2P_TRACE
```

The table below gives the *CGNS.MAP* flags.

Flag variable	*Function*
S2P_NONE	Clear all flags, set to zero.
S2P_ALL	Set all flags, set to one.
S2P_TRACE	Set the trace on, messages are sent to 'stdout'
S2P_FOLLOWLINKS	Continue to parse the linked-to tree (1)
S2P_MERGELINKS	Forget all link specifications. (2)
S2P_COMPRESS	Sets the compress flag for 'DataArray_t' (2)
S2P_NOTRANSPOSE	No *dimensions* transpose during load and save. (5)
S2P_NOOWNDATA	Forces the *numpy* flag \~NPY_OWNDATA (1) (3)
S2P_NODATA	Do not load large 'DataArray_t' (2) (4)
S2P_UPDATE	not used
S2P_DELETEMISSING	not used

There is no requirements or check on which flag can or cannot be associated with another flag.

Remarks:

1. Only when you are *loading* a tree.
2. Only when you are *saving* a tree.
3. Which means all ''DataArray_t' actual memory zones will **NOT** be released by Python.
4. The term *large* has to be defined. The *save* will **NOT** check if the CGNS/Python tree was performed with the S2P_NODATA flag on, then you have to check by yourself that your *save* will not overwrite an existing file with empty data!
5. The default behavior is to transpose array and dimensions of an array if this is not a NPY_FORTRAN array. If you set this falg to 1, no transpose would be performed and the array and its dimensions would be stored without modification even if the NY_FORTRAN flag is not there.

CHAPTER

THREE

THE MAP API

The MAP module is designed so that you can re-use the lead/save function and put them into your own application. This allows you to create a *CGNS/HDF* tree from a *CGNS/Python* tree into your C code. The two function are very close the to Python level interface functions.

PyObject* **`s2p_loadAsHDF`**(char **filename*, int *flags*, int *threshold*, int *depth*, char **path*)