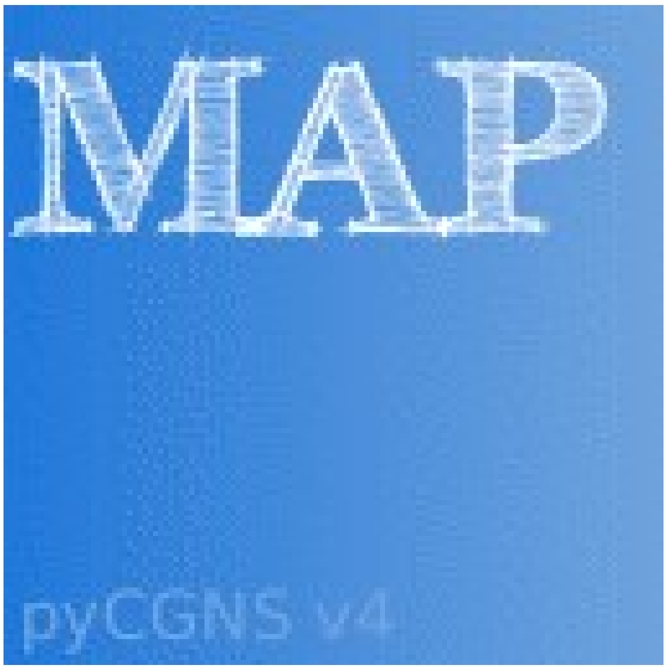

# *Guide to Python/CGNS*

Marc POINOT
ONERA/DSNA/CS2A

pyCGNS v4.0.1
MAP v1.0.0
June 2009

# FOREWORDS

The pyCGNS Python package is a set of tools for the CGNS standard. The tools are all released in the pyCGNS package but you can install and use each tool in a separate way, as far as dependancies for this tool allow you to do so.

| | |
|---|---|
| CGNS.WRA | **Wrapper**<br>A Python wrap to so-called midlevel library and adf library.<br>No dependancy to other pyCGNS modules, but it requires the CGNS and HDF5 libraries.<br>*Formerly was pyCGNS.MLL and pyCGNS.ADF* |
| CGNS.MAP | **Mapper**<br>Implements the SIDS-to-python mapping.<br>No dependancy to other pyCGNS modules, but it requires the HDF5 library.<br>*Formerly was CGNS.utils.saveAsADF and loadAsADF* |
| CGNS.PAT | **Patternater**<br>A set of functions to create or update complete SIDS sub-trees.<br>No dependancy to other pyCGNS modules.<br>*Formerly was pyCGNS.cgnslib* |
| CGNS.NAV | **Navigater**<br>A Tk based GUI for Python/CGNS trees.<br>Requires CGNS.MAP, CGNS.PAT, TkTreeControl and TkinterTreectrl.<br>*Formerly was pyS7* |
| CGNS.VAL | **Validater**<br>A customizable Python/CGNS tree parser to check tree compliance to SIDS or to user-defined rules.<br>Requires CGNS.MAP<br>*Formerly was pyC5* |
| CGNS.DAT | **Databaser**<br>A RDBMS-based application to store and manage set of CGNS files.<br>Requires CGNS.MAP and a RDBMS.<br>*Formerly was pyDAX* |
| CGNS.TRA | **Translater**<br>A set of file format translator to/from SIDS-to-Python or SIDS-to-HDF5.<br>Requires CGNS.MAP and CGNS.PAT<br>*Formerly was pyCRAB* |

## *CGNS Standard*

The Computational Fluid Dynamics community has a standard for data model definition and disk storage: CGNS (CFD General Notation System), it is an AIAA recommanded practice and it is in the process of being an ISO standard in the frame of ISO/STEP.

> △ In this document, we refer to the so-called official CGNS documents and libraries when we use the .org suffix. For example, the CGNS.org documentation means '*the official CGNS documentation or library or tools you can find on the web site www.cgns.org*'.

The reference document for CGNS is the so-called SIDS, which stands for *Standard Interface Data Structure*. The reference version is described in the document AIAA-R101A-2006 publicly available on the CGNS web site.

### About this document

The *pyCGNS-MAP* manual is, at the same time, a reference document and a users' guide. As we are supporting the CGNS standard, many informations related to the standard are not detailled here. The user should read the CGNS/SIDS to have more complete information about the CGNS data structures.

Each pyCGNS module has its own documentation, their may refer to each other.

The required modules and library versions, used by pyCGNS, can be detected at run-time using the pyCGNSconfig module. These are updated during the installation process.

# 1 SIDS MAPPING

The mapping describes how a CGNS node can be mapped to a Python object. You should refer to the CGNS.org official documents about the mappings, such as SIDS-to-ADF and SIDS-to-HDF5 mapping documents.

## 1.1 Basic node

A CGNS tree is composed of nodes, each node is a list of name, value, list of children and type (or label, which is same to CGNS type). The list of children itself is a list of node. The name and the type are Python strings. The value is a numpy array or a Python built-in type.

```
<CGNS-node> = [ <name>, <value>, [ <CGNS-node>+ ], <label> ]
```

A node cannot be an empty list, a list of children can be an empty list. The name should follow the CGNS/SIDS requirements as described in the Annex A. The type is one of the CGNS/SIDS type as a string, for example `CGNSBase_t` is “`CGNSBase_t`”.

There are some examples of simple nodes. We use there the ASCII representation of Python objects, in other words these are the objects as if were typing them in a Python file to be interpreted. We give more details about the internals in the section about the Python and numpy C APIs.

```
[ 'ZoneType', 'Structured', [], 'ZoneType_t' ]

[ 'Pressure', array([2.3, 2.4]), [], 'DataArray_t' ]

[ 'PointRange', array([[1,1,1],[5,7,9]], [], 'PointRange_t' ]
```

There is no information about the dimensions of the value, these can be deduced from the Python object supporting this value (i.e. The length for a string, the shape for an array, 1 for all other types). The shape of an array is ordered with the C indexing convention. For example, the PointRange above is of dimension 2x3 (the element (1,2) is 9). An empty value is `None`.

A CGNS sub-tree is any list of CGNS-nodes. A node without child has an empty list as child list.

There is no order in the list of children, the two examples below represent the same CGNS tree.

```
['Data',None,[
  ['Pressure',array([2.4]),[],'DataArray_t'],
  ['Mach',    array([0.8]),[],'DataArray_t']
],'UserDefinedData_t']

['Data',None,[
  ['Mach',    array([0.8]),[],'DataArray_t'],
  ['Pressure',array([2.4]),[],'DataArray_t']
], 'UserDefinedData_t']
```

The list can be replace by the tuple for the node container. The list and the tuple are Python sequences are most operation are the same. The tuple is cannot be changed, a CGNS node built with a tuple is a read-only node:

```
( 'ZoneType', 'Structured', [], 'ZoneType_t' )
```

There is no constraint about the use of a list or a tuple for the node itself and for the children, however it is strongly recommanded to use a list for the children.

## 1.2 Links

The idea of links is irrelevant in Python. The Python lists are pointers to the actual Python list objects, when you refer to a list you have a link. Then, as the default behavior, a lists are not copied when you refer to them. If you do not want to have the same object, you have to copy.

There is no link node in Python, but you have link nodes with the CGNS/HDF5 mapping. The SIDS-to-Python mapping defines the link as a separate information, not stored in the CGNS tree itself but in a companion list.

## 1.3 The CGNS Root

The is no Root node in the CGNS standard. Of course, there is one hidden in every implementation or mapping, but it doesn't really appear as a Root node type. The CGNS/Python defines its own CGNS root type, the CGNSTree_t, it allows the first node to be a normal node, in other words the root node has a name, a value, a list of children and a type.

The name is user defined name, there is no value, the type is CGNStree_t and the list of children contains the CGNSLibraryVersion_t and the CGNSBase_t nodes.

When you use CGNS.MAP functions, the module checks if you have the CGNStree node or not and, most of the time, the user ignores wether the root node is used or not.

## 1.4 Types mapping

The CGNS/SIDS doesn't define data types, they are defined by the mapping. The required types are listed in the table here after:

| SIDS SSL type | Numpy type | Plain Python type | Remark |
|---|---|---|---|
| C1 | s | string | |
| I4 | i | int | Single precision integers are changed to Python int (i.e. long) |
| I8 | l | int | |
| R4 | f | | Single precision reals are changed to Python float (i.e. double) |
| R8 | d | float | |

## 1.5 Atomic values vs numpy arrays

You may want to map a single value as an actual Python object instead of a numpy array. For example, it is easy to define a an integer value a

```
a=3
```

instead of

```
a=numpy.array([3])
```

So we accept the definition of such atomic values. However, the type mapping implies that all integers and all floating point values are stored as long integers and double floats. If you want to force a specific precision, you should use the numpy types:

```
CGNSLibraryVersion=numpy.array([2.4],dtype='f')
```

If you set a value with the string Python type, instead of the numpy array of chars, this string us then one dimensional.

# 2 MODULE INTERFACE

The MAP module has a very small interface. The idea is to provide a light module one could use as a simple translator without a large set of files.

## 2.1 Interface

The user interface defines no class. We really want the Python mapping to be library-independant, a CGNS tree representation in Python only requires the built-in types and the numpy module.

The two main function of MAP are `loadAsHDF` and `saveasHDF`. One reads a CGNS/HDF5 file and creates the corresponding CGNS/Python tree, the other performs the save from a CGNS/Python tree to a CGNS/HDF5 file.

The flags have a strong side effect on the way your CGNS/Python trees are read or write on the disk.

## 2.2 Flags

The flags are used as load and save parameters. Theses are bitfields you can set/unset in order to have the expected behavior of the MAP during load or save.

| | |
|---|---|
| S2P_NONE | Empty flag set, all flags set to OFF |
| S2P_ALL | Full flag set, all falgs set to ON |
| S2P_TRACE | Print textual trace messages on stdout. Used during load and save. |
| S2P_FOLLOWLINKS | Continue parsing the CGNS tree when encountering a link. Used during load and save. In the case of save, ON indicates that the link table should be used. In the case of load, OFF will force an empty link table as result. |
| S2P_MERGELINKS | Only used during save, ON will force all tree to be saved in a single file. |
| S2P_COMPRESS | Only used during save, ON will force data compression (for DataArray_t). |
| S2P_NOTRANSPOSE | Never transpose data |
| S2P_OWNDATA | Only used during load, ON forces numpy arrays to have the OWNDATA flag set. |
| S2P_NODATA | Load will force actual DataArray_t to empty, save will not change actual data if the flag S2P_UPDATE is ON. |
| S2P_UPDATE | Load will add new nodes to the Python tree when ON, see also S2P_DELETEMISSING. |
| S2P_DELETEMISSING | In the case of S2P_UPDATE, the load will remove nodes from the Python tree if they are not in the ADF tree, the save will delete ADF nodes not found in the Python tree. |

The flags can be stored as a Python object (actually, the bitfield is understood as an integer).

# 3 C API LEVEL

MAP can be used as pure Python module but it also has been designed to provide embedded-able files.

## *3.1 Single file translation*

A single file is required.

## *3.2 Python and numpy API remarks*

The numpy API should be initialized before any use.

## *3.3 Embedding SIDStoPython*

Your application can add the SIDStoPython file.