pyCGNS.intro/Manual

Release 4.2.0

Marc Poinot

August 27, 2007

CONTENTS

Release 4.2

- *CGNS.MAP* is now a wrapper to CHLone python module
- *CGNS.WRA* reborn, re-coding using *Cython*
- *CGNS.VAL* reborn
- completely new *CGNS.NAV* using *Qt*, *VTK* and *Cython*
- YouTube demos here

The package gathers various tools and libraries for CGNS end-users and Python application developpers. The main object of *pyCGNS* is to provide the application developpers with a Python interface to *CGNS/SIDS*, the data model. The *MAP* and *PAT* modules are dedicated to this goal: map the *CGNS/SIDS* data model the *CGNS/Python* implementation. The *WRA* module contains wrapper on *CGNS/MLL* and a *MLL-like* set of functions that uses the *CGNS/Python* mapping as implementation. The *NAV* module supports the *CGNS.NAV* graphical browser, with nice features about tree exploration, copy/paste and even global node changes. Then, the *VAL* module is a parser engine for CGNS/Python tree compliance checking. The *CGNS.VAL* tool can analyze your CGNS/HDF5 file and returns you a large panel of diagnostics.

The package uses *numpy*, *cython* and *HDF5* you should install before pyCGNS. The *CGNS.NAV* tools has an optional *VTK* viewer which requires the *VTK lib* and its python interface. The *CGNS/SIDS* data model has a mapping the *HDF5* for file storage. The *MAP* module uses *CHLone* instead of *CGNS/MLL* to map its *CGNS/Python* trees to *HDF5*.

Note: The *pyCGNS* python package is released under LGPL2 license and hosted by sourceforge where you can find source download, help forum, wiki and bug tracking.

The listed chapters are the introduction chapters, each MAP, PAT, WRA, NAV, VAL module has its own documentation which doesn't appear below. Click on the modules icons above to get their documentation.

CHAPTER

ONE

ABOUT PYCGNS

1.1 CGNS Standard

The purpose of these pages is not to describe the standard. There is the *cgns.org* web site with a lot of documentation. Anyway, there is a set of *How to's* using *pyCGNS* you can browse and learn a bit about this standard.

Welcome, you are now on '**CGNS How to?**', the Onera CGNS users web site. If you don't kown about CGNS or if you want an exhaustive information about it, you should see first the official CGNS web site (we refer to it as *cgns.org* in these pages).

- You are just looking for a practical *CGNS/SIDS* node use?
- Jump to *cross tables*.

1.1.1 CGNS How to?

The CGNS standard has been specified to cover a large part of the CFD data usage, a newcomer can be lost with all the possibilities CGNS offers. We have gathered here some practices of the Onera users of CGNS. Like a programming language, it is often useful to start by cloning existing simple examples, or finding out a pattern and copy it into another piece of code. The purpose of this site is to provide CGNS users with materials taken from our CGNS experience. You obviously have to change the examples to fit with your own applications parameters and requirements.

Of course, all code examples are in CGNS/Python...

Data model

The *CGNS/SIDS* is a **data model**. It describes the data you need to define (mandatory or optional data) the semantics of it, the way you organize it and its relationships with the other data. This describes a logical model of your application data.

Node and trees

A CGNS is a tree. A tree is a node which has node has children. The top node of a *CGNS/SIDS* tree has the CGNSBase_t type.

Actual top node

Node description

References

CGNSBase_t + +

Tree main structure

A CGNS tree has a topological structure. The branches of the tree are representing the actual topological divisions of your computation.

The skeleton

The support of your computation data is the grid (or grids). The grid can be structured and/or unstructured. The way your grid is cut into parts defines the skeleton of your tree. The branches that mainly holds the data are the *Base*, *Zone*, *Elements*, *Connectivity* and *Boundary conditions* nodes.

A CGNS is a tree. A tree is a node which has node has children. The top node of a *CGNS/SIDS* tree has the CGNSBase_t type.

Actual top node

Node description

Which patterns in which example?

The tables hereafter are giving you the relationships between the nodes or pattern of nodes and the corresponding examples.

1.2 Package contents

The pyCGNS Python module is a collection of 7 modules around the CGNS standard. Before v4, these modules were independent Python modules with more or less dependancies to each other. We gather all of them to have a common build/install/doc and test process, moreover this insures a better consistency between them.

The pyCGNS module now includes (former package names)

- MAP, the Mapper, new in v4 gives basic load/save function from/to *CGNS/SIDS* and *CGNS/HDF5*. This very sinple module is able to read/write GCNS/HDF5 files and translate them to CGNS/Python. This is the main feature of pyCGNS v4.0.
- PAT, the PatterMaker, a full *CGNS/SIDS* patterns using the *CGNS/Python* mapping. This is pure python module, it creates and modify CGNS/Python trees without the help of any HDF5 or even ADf calls.
- NAV, the Navigater (pyS7), a graphical browser that can handle *CGNS/Python*, *CGNS/HDF5* and *CGNS/ADF* file formats. It is slightly different to *cgnsviewer* because it actually is a tree editor, you can copy/cut/paste CGNS/Python trees and quickly draft or modify your CGNS tree.
- WRA, the Wrapper (pyCGNS), is a *CGNS/MLL* and *CGNS/ADF* Python wrapping. All the CGNS/MLL functions are mapped to their Python clone.
- VAL,the Validater (pyC5), an XML grammar based validation of a *CGNS/Python* tree, for example produced using *MAP* or *PAT*. *Unsuable in v4.0*
- DAT, the DataTracer (pyDAX), some DBMS services for *CGNS/HDF5* files. *Unsuable in v4.0*
- APP, the Applicater, a set of applications using other modules. *Unsuable in v4.0*

1.3 Quick start

1.3.1 Loading a CGNS/HDF file with MAP

The *CGNS.MAP* module implements the *CGNS/Python* mapping. You can load/save a *CGNS/HDF5* file using the simple *MAP* functions (below, the >>> string is the python interpreter prompt).

Now `tree` is a Python list with the whole‘‘ ./001Disk.hdf‘‘ CGNS tree into, with the data structure as described in *SIDS-to-Python*.

1.3.2 Using PAT to modify a CGNS tree

The previously loaded *CGNS/Python* tree is modified using plain Python functions and types. The *CGNS.APP* module contains utilities, we use the `getNodeByPath` function which returns a *CGNS/Python* node with the target tree and the target node path as parameters.:

```
import CGNS.APP.path_utils as CGU

node=CGU.getNodeByPath("/Disk/zone1/ZoneBC/ext1/PointRange",tree)
```

The returned node is a list of 4 Python values, the name (a string), the value of the node (a *Numpy* array), the list of children and the CGNS type of the node (string).

1.3.3 Using PAT to create a CGNS tree

We want to create a *CGNS/HDF5* file with a simple *base* and a *reference state*:

```
import CGNS.PAT.cgnslib as CGL

T=CGL.newCGNSTree()
CGL.newBase(T,'Test Case 01',3,3) # name, physical dim, topological dim
CGL.newSimulationType(T)
CGL.newReferenceState(T)
```

1.3.4 Re-using your CGNS/MLL scripts

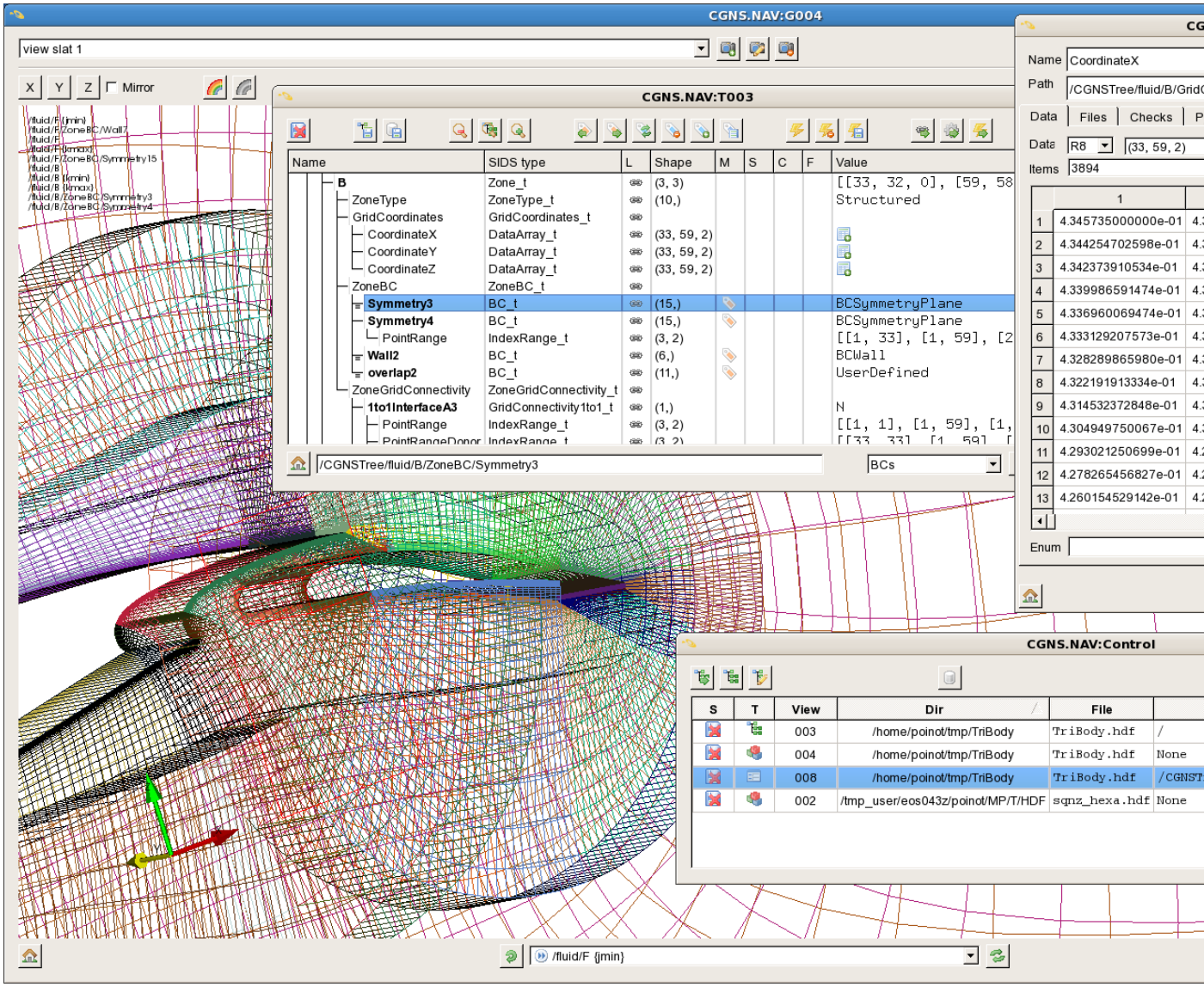
If you really want to use CGNS/MLL, for example if you have a large toolbox with old pyCGNS scripts you have to change your imports from *CGNS.WRA* to {CGNS.WRA.wrapper‘ for functions:

```
import CGNS.WRA.wrapper

filesol=CGNS.WRA.wrapper.pyCGNS('testfile.cgns',CGNS.WRA.MODE_WRITE)
filesol.basewrite('Base',3,3)
filesol.close()
```

1.3.5 Browsing your CGNS tree with NAV

The CGNS.NAV tool is a CGNS tree brower. You start it typing `CGNS.NAV` and you can open several views on your file.


CGNS.NAV:G004
view slat 1
X
Y
Z
Mirror
CGNS.NAV:T003
Name
SIDS type
L
Shape
M
S
C
F
Value
B
Zone_t
(3, 3)
[[33, 32, 0], [59, 58
ZoneType
ZoneType_t
(10,)
Structured
GridCoordinates
GridCoordinates_t
CoordinateX
DataArray_t
(33, 59, 2)
CoordinateY
DataArray_t
(33, 59, 2)
CoordinateZ
DataArray_t
(33, 59, 2)
ZoneBC
ZoneBC_t
Symmetry3
BC_t
(15,)
BCSymmetryPlane
Symmetry4
BC_t
(15,)
BCSymmetryPlane
PointRange
IndexRange_t
(3, 2)
[[1, 33], [1, 59], [2
Wall2
BC_t
(6,)
BCWall
overlap2
BC_t
(11,)
UserDefined
ZoneGridConnectivity
ZoneGridConnectivity_t
1to1InterfaceA3
GridConnectivity1to1_t
(1,)
N
PointRange
IndexRange_t
(3, 2)
[[1, 1], [1, 59], [1,
/CGNSTree/fluid/B/ZoneBC/Symmetry3
BCs
Name
CoordinateX
Path
/CGNSTree/fluid/B/GridC
Data
Files
Checks
Data
R8
(33, 59, 2)
Items
3894
1
4.345735000000e-01
4.344254702598e-01
4.342373910534e-01
4.339986591474e-01
4.336960069474e-01
4.333129207573e-01
4.328289865980e-01
4.322191913334e-01
4.314532372848e-01
4.304949750067e-01
4.293021250699e-01
4.278265456827e-01
4.260154529142e-01
Enum
CGNS.NAV:Control
S
T
View
Dir
File
003
/home/poinot/tmp/TriBody
TriBody.hdf
004
/home/poinot/tmp/TriBody
TriBody.hdf
None
008
/home/poinot/tmp/TriBody
TriBody.hdf
002
/tmp_user/eos043z/poinot/MP/T/HDF
sqnz_hexa.hdf
None
/fluid/F {jmin}

CHAPTER

TWO

BUILD AND INSTALL

The installation described here is for UNIX platforms only.

The installation process is easy in the case you already have the required libraries and a standard configuration.

2.1 Required libraries

The first step of the installation is to make sure you have the required libraries. We have now a quite large set of libraries, we are trying hard to use well known and easy to install libraries but we must admit this could be a difficult task to install all by yourself...

The mandatory libs are *Python*, *numpy*, *HDF5* and *CHLone*, *VTK* and *Qt*. The *cython* compiler and *scons* (for CHLone) are also required. Then, if you want to build *CGNS.WRA* (we recommend to do so), you need *libcgns*.

> **Warning:** The *libcgns* (*CGNS/MLL*) is **NOT** required for pyCGNS. You need it if you want the **WRA** module, all other pyCGNS tools are using *CHLone*.

The complete set of required tools/modules/libraries contains Python, HDF5 and other graphical toolkits:

- Python (starting from v2.4)
- numpy (v1.1 +)
- cython (v0.16 +)
- hdf5 (v1.8.5 +)
- CHLone (v0.4 +)
- VTK (v5.8 +)
- Qt (v4.7 +)
- pySide (v4.7 +)

The *VTK* and the *Qt* graphical toolkits need to have a Python binding. The important point is to use the **native** Python binding of these toolkits, not any other (see installation details hereafter).

No Python binding is required for HDF5.

2.2 Optional libraries

The so-called *mid-level* library is not mandatory, the *WRA module* is the only one to have dependancies on.

- CGNS/MLL (libcgns) (starting v3.0)

2.3 Installation process

Once you have these installed you can proceed with pyCGNS. You go into the top directory and you edit the `pyCGNSconfig.py.in` (see *Configuration file contents*). You have to set the correct paths and various values such as directory search libs or flags.

Then you run:

```
python setup.py build
```

and then:

```
python setup.py install
```

or:

```
python setup.py install --prefix=/local/tools/installation
```

All the modules of the pyCGNS package are installed and you can now proceed with tutorial examples.

2.4 Single module installation

You can ask for a single module installation:

```
python setup.py build --single-module=MAP
python setup.py install
```

You have to check that this installation doesn't overwrite an existing installation with the other pyCGNS modules.

2.5 Configuration file contents

The `pyCGNSconfig_user.py` should work with no modification if you have a standard installation. All you have to declare is the directory in which we can find *Python/numpy/hdf5/CHLone/cgns* libraries.

If you have specific installations you can change some paths/flags for each external library: *hdf5*, *numpy*, *CGNS/MLL* and *CHLone*. The configuration file is a Python file, it is imported after the default configuration. The changes you make in the configuration file will overwrite the defaults:

```
# --- stuff to add for HDF5

#HDF5_VERSION          = ''
HDF5_PATH_INCLUDES    = ['/home/myself/hdf5/include']
HDF5_PATH_LIBRARIES   = ['/home/myself/hdf5/lib']
#HDF5_LINK_LIBRARIES   = []
#HDF5_EXTRA_ARGS       = []
```

To avoid overwriting, use Python to update the config:

```
# --- stuff to add for HDF5

#HDF5_VERSION          = ''
HDF5_PATH_INCLUDES    = ['/home/myself/hdf5/include']
HDF5_PATH_LIBRARIES   = ['/home/myself/hdf5/lib']
#HDF5_LINK_LIBRARIES   = []
HDF5_EXTRA_ARGS       = HDF5_EXTRA_ARGS + ['-DMYFLAG']
```

You do not have to specify the VTK nor the Qt libraries, there is no direct link to these libraries: the Python modules will do that.

2.5.1 Module dependancies

The pyCGNS modules have dependancies with their brothers. The list below gives you the required modules (or optional) for each of them.

- MAP : None
- PAT : MAP
- WRA : PAT MAP
- APP : PAT MAP
- NAV : PAT MAP APP (WRA)

2.6 NAV depends

The CGNS.NAV tool uses two graphical toolkits, one for the GUI (Qt) and another one for the graphical view window (VTK). The VTK toolkit must have its Python binding installed. You should not use *pyVTK* but the native Python binding of VTK. To do so, edit the VTK configuration file to set the Python wrapping and all associated paths. And example of production could be:

```
cd VTK
mkdir B
cd B
cmake ..
```

Then edit the *CMakeCache.txt* file:

```
//Wrap VTK classes into the Python language.
VTK_WRAP_PYTHON:BOOL=ON
```

And build/install...

The Qt toolkit is used with the PySide Python binding, which is its official bindingwith this langage. We do not use pyQt. You can check you have the correct bindings by typing:

```
python
>>>import vtk
>>>import PySide
```

Both should not fail...

2.7 MAP depends

The *CHLone* librarie is required and thus *HDF5* is required.

2.8 WRA depends

CGNS/MLL and *CGNS/ADF* libraries are required. You should build the CGNS libraries with the `CG_BUILD_SCOPE` set to `True`. To do so, edit your `CMakeCache.txt` file and set the following variable to `ON`:

```
//Enable or disable scoping of enumeration values
ENABLE_SCOPING:BOOL=ON

//Build the CGNSTools package
BUILD_CGNSTOOLS:BOOL=ON
```

```
//Build a shared version of the library
CGNS_BUILD_SHARED:BOOL=ON

//Enable or disable the use of Fortran
ENABLE_FORTRAN:BOOL=ON

//Enable or disable HDF5 interface
ENABLE_HDF5:BOOL=ON
```

The WRA and NAV modules (and CHLone by the way) are using cython. You can check your cython is present using:

```
cython --version
```

That should not fail...

CHAPTER

THREE

GLOSSARY

cgns.org The official CGNS web site, by extension any document on this web site has an *official* taste...

CGNS The specific purpose of the CFD General Notation System (CGNS) project is to provide a standard for recording and recovering computer data associated with the numerical solution of the equations of fluid dynamics. See also the *How to?*.

CGNS/SIDS The Standard Interface Data Structure is the specification of the data model. This public document describes the syntax and the semantics of all tree-structured data required or proposed for a CFD simulation.

CGNS/MLL The Mid-Level Library is an example implementation of *CGNS/SIDS* on top of *CGNS/ADF* and *CGNS/HDF5* mappings. This library has a C and a Fortran API.

CGNS/ADF The Advanced Data Format *CGNS/SIDS* implementation. A binary storage format and its companion library, developped by *Boeing*.

CGNS/HDF5 The Hierarchical Data Format *CGNS/SIDS* implementation. A binary storage format and its companion library (see below).

CGNS/Python The Python programming language *CGNS/SIDS* implementation.

CHLone A *CGNS/HDF5* compliant implementation. The CHLone library is available on SourceForge.

HDF5 A powerful storage system for large data. The HDF5 library should be seen as a middleware system with a lot of powerful features related to efficient, portable and trustable storage mean.

python An object oriented interpreted programming language.

cython A compiler tool that translate Python/Numpy into C code for performance purpose.

numpy The numerical library for Python. *Numpy* is used to store the data in Python arrays which have a direct memory mapping to actual C or Fortran memory.

VTK A visualization toolkit used to display 3D objects ni *CGNS.NAV*.

PySide The Python interface for the Qt toolkit. PySide

Qt A powerful graphical toolkit available under GPL v3, LGPL v2 and a commercial license. The current use of Qt is under LGPL v2 in pyCGNS.

CHAPTER

FOUR

PDF DOCS

- Introduction document
- MAP manual
- PAT manual
- NAV manual
- WRA manual
- VAL manual